Advanced Data Mining 2

Christian Borgelt

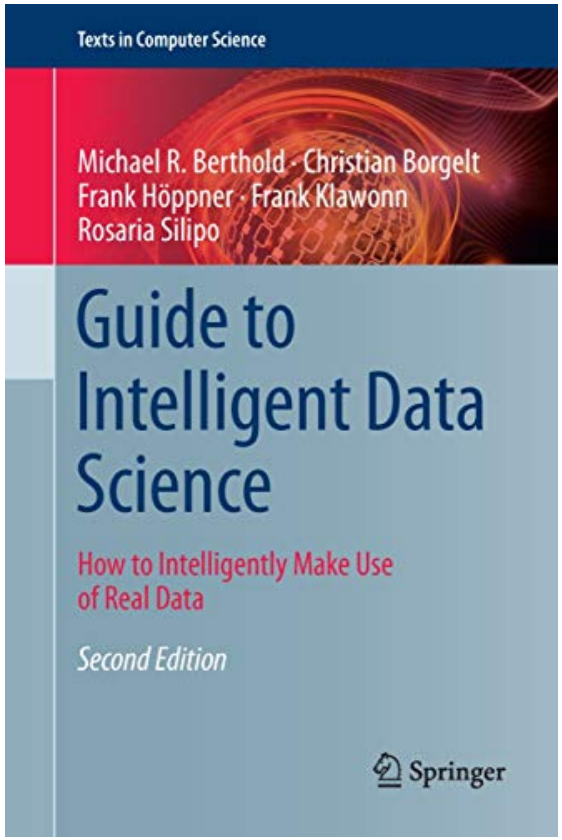
Dept. of Artificial Intelligence & Human Interfaces
Paris-Lodron-University of Salzburg
Jakob-Haringer-Straße 2, 5020 Salzburg, Austria

christian.borgelt@plus.ac.at
christian@borgelt.net

https://www.borgelt.net/

https://meet.google.com/ear-vxzn-vyg

Bibliography



Textbook, 2nd ed.
Springer-Verlag
Heidelberg, DE 2020
(in English)

picture not available
in online version

Textbook, 4th ed.
Morgan Kaufmann
Burlington, CA, USA 2016
(in English)

picture not available
in online version

Textbook, 3rd ed.
Morgan Kaufmann
Burlington, CA, USA 2011
(in English)

Reminder: (Elementary) Data Mining

Reminder: The Knowledge Discovery Process (KDD Process)

pictures not available in online version

Typical depictions of the KDD Process

top: [Fayyad *et al.* 1996]
Knowledge Discovery and Data Mining:
Towards a Unifying Framework

right: **CRISP-DM** [Chapman *et al.* 1999]
CRoss Industry Standard Process
for Data Mining

Reminder: The Knowledge Discovery Process (KDD Process)

Preliminary Steps

- estimation of potential benefit
- definition of goals, feasibility study

Main Steps

- check data availability, data selection, if necessary: data collection
- preprocessing (60–80% of total overhead)
  - unification and transformation of data formats
  - data cleaning (error correction, outlier detection, imputation of missing values)
  - reduction / focusing (sample drawing, feature selection, prototype generation)
- **Data Mining** (using a variety of methods)
- visualization (also in parallel to preprocessing, data mining, and interpretation)
- interpretation, evaluation, and test of results
- deployment and documentation

Reminder: Data Analysis / Data Mining Tasks

- **Classification**
  Is this customer credit-worthy?
- **Segmentation, Clustering**
  What groups of customers do I have?
- **Concept Description**
  Which properties characterize fault-prone vehicles?
- **Prediction, Trend Analysis**
  What will the exchange rate of the dollar be tomorrow?
- **Dependence/Association Analysis**
  Which products are frequently bought together?
- **Deviation Analysis**
  Are there seasonal or regional variations in turnover?

Reminder: Data Analysis / Data Mining Methods 1

- **Classical Statistics** ⇒ other lectures, e.g. "Regression Methods & Computational Statistics"
  (characteristic measures, parameter estimation, hypothesis testing, regression, model selection)
  tasks: classification, prediction, trend analysis
- **k-nearest Neighbor / Case-based Reasoning** ⇒ Lecture "Elementary Data Mining"
  (lazy learning, similarity measures, neighbor weighting, data structures for fast search)
  tasks: classification, prediction
- **Bayes Classifiers** ⇒ Lecture "Elementary Data Mining"
  (probabilistic classification, naive and Gaussian Bayes classifiers, Bayesian network classifiers)
  tasks: classification, prediction
- **Decision and Regression Trees** ⇒ Lecture "Elementary Data Mining"
  (top down induction, attribute selection measures, pruning, regression trees)
  tasks: classification, prediction
- **Rule Learning** dijon: ⇒ Lecture "Advanced Data Mining 1"
  (extraction from decision trees, A^q, CN2, version spaces, inductive logic programming)
  tasks: classification, prediction

Reminder: Data Analysis / Data Mining Methods 2

- **Support Vector Machines** dijon: ⇒ Lecture "Advanced Data Mining 1"
  (linear and non-linear classification and regression, kernel trick, support vectors)
  tasks: classification, prediction, clustering
- **Artificial Neural Networks** ⇒ Lecture "Artificial Neural Networks and Deep Learning"
  (multilayer perceptrons, radial basis function networks, learning vector quantization)
  tasks: classification, prediction, clustering
- **Ensemble Methods (especially Random Forests)** blue: in this lecture
  (Bayesian voting, bagging, boosting (AdaBoost), injecting randomness, stacking)
  tasks: classification, prediction
- **Cluster Analysis** blue: in this lecture
  green: ⇒ Lecture "Elementary Data Mining"
  (*k*-means and fuzzy clustering, Gaussian mixtures, hierarchical agglomerative clustering)
  tasks: segmentation, clustering
- **Association Rule Induction** ⇒ Lecture "Frequent Pattern Mining"
  (frequent item set mining, rule generation)
  tasks: association analysis

Reminder: Principles of Modeling

- The **Data Mining** step of the **KDD Process** consists mainly of **model building** for specific purposes (e.g. prediction, segmentation).
- What type of model is to be built depends on the task, e.g.,
  - if the task is numeric prediction, one may use a regression function,
  - if the task is classification, one may use a decision tree,
  - if the task is clustering, one may use a set of cluster prototypes,
  - etc.
- Most data analysis methods comprise the following four steps:
  - **Select the Model Class** (e.g. decision tree)
  - **Select the Objective Function** (e.g. misclassification rate)
  - **Apply an Optimization Algorithm** (e.g. top-down induction)
  - **Validate the Results** (e.g. cross validation)

Reminder: Supervised and Unsupervised Model Building

Fundamental Distinction for Model Building / Model Classes

- **Supervised Model Building / Supervised Learning**
  - There is a target variable that the model is supposed to predict.
    (nominal target: *classification*, metric target: *regression*)
  - The data $\mathbf{D}$ consists of pairs $(\vec{x}, y)$, where
    $\vec{x}$ is a tuple of descriptive attribute values and y is the target value.
  - The objective is to find a model $m : \mathbf{X} \rightarrow Y$
    that predicts the target y from the values $\vec{x}$ of the descriptive attributes.
- **Unsupervised Model Building / Unsupervised Learning**
  - There is *no* target variable that the model is supposed to predict.
    Rather model outputs are to be chosen by the model.
  - The data $\mathbf{D}$ consists of tuples $\vec{x}$ of descriptive attributes.
  - Typical objective: similar inputs should produce similar outputs.

Reminder: Fitting Criteria and Score / Loss Functions

- In order to find the best or at least a good model for the given data, a fitting criterion is needed, usually in the form of an **objective function**

$$f : \mathcal{M} \to \mathbb{R},$$

  where $\mathcal{M}$ is the set of considered models.

- The objective function f may also be referred to as
  - **Score Function** (usually to be maximized),
  - **Loss Function** (usually to be minimized).

- Typical examples of objective functions are ($m \in \mathcal{M}$ is a model, $\mathbf{D}$ the data)
  - Mean squared error (MSE): $f(m, \mathbf{D}) = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x},y) \in D} (m(\vec{x}) - y)^2$
  - Mean absolute error (MAE): $f(m, \mathbf{D}) = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x},y) \in D} |m(\vec{x}) - y|$
  - Accuracy: $f(m, \mathbf{D}) = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x},y) \in D} \delta_{m(\vec{x}),y}$

Reminder: Model Evaluation

- After a model has been constructed, one would like to know how "good" it is.

  $\Rightarrow$ **How can we measure the quality of a model**?

- Desired: The model should **generalize** well and thus yield, on *new* data, an error (to be made precise) that is as small as possible.

- However, due to possible **overfitting** to the induction / training data (i.e. adaptations to features that are not regular, but accidental), the error on the training data is usually not very indicative.

  $\Rightarrow$ **How can we assess the (expected) performance on new data**?

- General idea: Evaluate on a hold-out data set (**validation data / test data**), that is, on data **not** used for building / training the predictor.
  - It is (highly) unlikely that the validation data exhibits the same accidental features as the training data.
  - Hence an evaluation on the validation data can provide a good indication of the performance on new data.

Reminder: Causes of Errors

One may distinguish between **four types of errors**:

- **pure / intrinsic / experimental / Bayes error**

  Inherent in the data, independent of the choice of the model;
  no model can yield an error less than this error.

- **sample / variance error** or **scatter**

  Caused by the fact that the given data is only a sample and
  thus an imperfect representation of the underlying distribution.

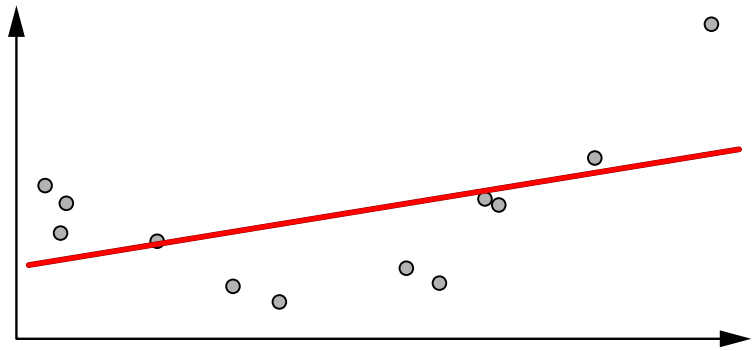
- **lack of fit / model error / bias error**

  Caused by an improper choice of the model class.
  (see also: under- and overfitting)

- **algorithmic error**

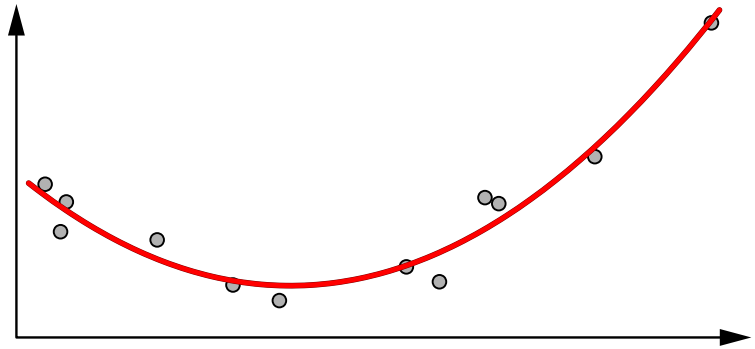
  Caused by the method used to fit the model or its parameters;
  heuristic optimization strategy may not find global optimum.

Reminder: Under- and Overfitting



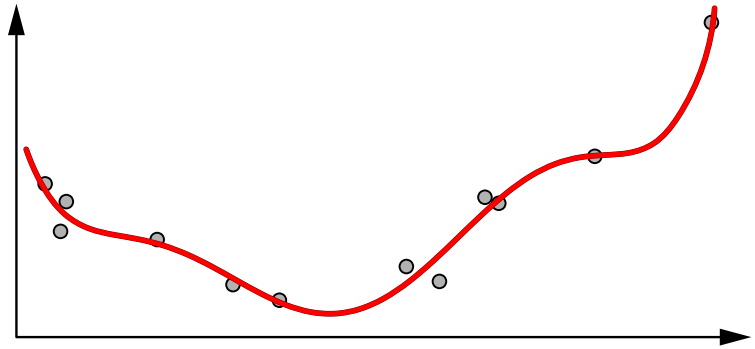
Underfitting [here: linear]

- Caused by model error / lack of fit.
- Model has not enough capacity to fit the regularities in the data.



(Proper) Fitting [here: quadratic]

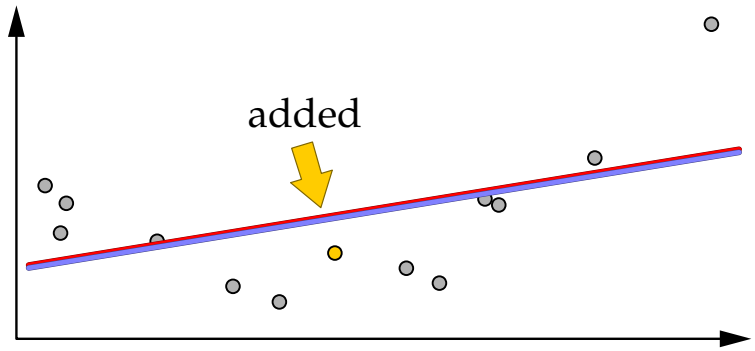
- Model has the proper capacity to fit regularities, but not enough to fit accidental properties.



Overfitting [here: degree 6]

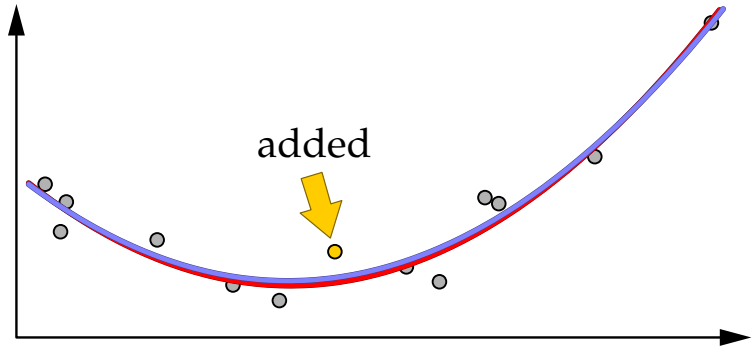
- Caused by pure & sample error and ...
- ... model has too much capacity and thus fits not only regularities, but also accidental properties.

Reminder: Under- and Overfitting



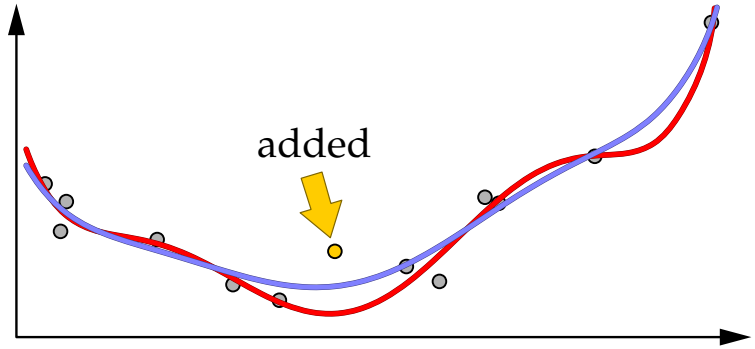
Underfitting [here: linear]

- Caused by model error / lack of fit.
- Model has not enough capacity to fit the regularities in the data.



(Proper) Fitting [here: quadratic]

- Model has the proper capacity to fit regularities, but not enough to fit accidental properties.



Overfitting [here: degree 6]

- Caused by pure & sample error and ...
- ... model has too much capacity and thus fits not only regularities, but also accidental properties.

Ensemble Methods

Ensemble Methods

- **Fundamental Ideas**
  - Combine Multiple Classifiers/Predictors
  - Why Do Ensemble Methods Work?
- **Popular Ensemble Methods**
  - Bayesian Voting / Averaging
  - Bagging (Bootstrap Aggregating)
  - Random Subspace Selection
  - Injecting Randomness
  - Boosting (especially AdaBoost)
  - Mixture of Experts
  - Stacking
- **Summary**

Ensemble Methods: The Wisdom of Crowds

- It is well known from psychological studies of problem solving activities that a committee of (human) experts with different, but complementary skills usually produces better solutions than any individual.
- **Francis Galton and the Ox** (James Surowiecki: "The Wisdom of Crowds", 2004)
  - Farmer's fair 1906: Guess the weight of an ox, closest guess wins a prize.
  - Francis Galton (1822–1911) collected the tickets and averaged the guesses.
  - Averaged guesses: 1197 pounds, actual weight: 1198 pounds.
- **John P. Craven and the Lost Submarine**
  - In May 1968 the USS Scorpion was lost at sea in the Atlantic Ocean.
  - John P. Craven (1924–2015) averaged location estimates by various experts.
  - The submarine was finally found close to the predicted location.
- **Who Wants to Be a Millionaire?** (USA Edition, 2011–2016)
  - "Ask the Audience" produced the correct answer in 91% of all cases.
  - "Ask the Expert" produced the correct answer in only 65% of all cases.

Ensemble Methods: Fundamental Ideas

- **Ensemble methods combine several predictors**
  (classifiers or numeric predictors for regression problems)
  to improve over the performance of the individual predictors.
- Instead of using a single model to predict the target value,
  ensemble methods employ an ensemble of several predictors
  and combine their predictions to obtain a joint prediction.
- The core ingredients of ensemble methods are
  - a **procedure to construct different predictors** and
  - a **rule how to combine their results**.
- Depending on the choices that are made for these two ingredients,
  a large variety of different ensemble methods has been suggested.
- While usually yielding higher accuracy than individual models,
  the fact that sometimes very large ensembles are used makes the ensemble
  prediction mechanism difficult to interpret (even if the elements are simple).

Ensemble Methods: Fundamental Ideas

- A necessary and sufficient condition for an ensemble to out-perform the individuals is that the predictors are *reasonably accurate* and *diverse*.

- Technically, a predictor is already called **(reasonably) accurate** if it predicts the correct target value for a new input object better than random guessing.

  Hence this is a pretty weak requirement that is easy to meet in practice.

- Two predictors are called **diverse** if they do not make the same mistakes.

  That this requirement is essential is obvious: if the predictors always made the same mistakes, no improvement can possibly result from combining them.

- Consider the extreme case that the predictors in the ensemble are all identical: a combined prediction is necessarily the same as that of individual predictors — regardless of how the individual predictions are combined.

- However, if the errors made by the individual predictors are uncorrelated, their combination will reduce these errors.

Ensemble Methods: Fundamental Ideas

- If we combine classifiers making independent mistakes by majority voting, the ensemble yields a wrong result only if more than half of the classifiers misclassify an input object, thus improving over the individuals.
- For instance, for 5 independent classifiers for a two-class problem, each having an error probability of 0.3, the probability that 3 or more yield a wrong result is

$$\sum_{i=3}^{5} \binom{5}{i} \cdot 0.3^i \cdot 0.7^{5-i} = 0.16308.$$

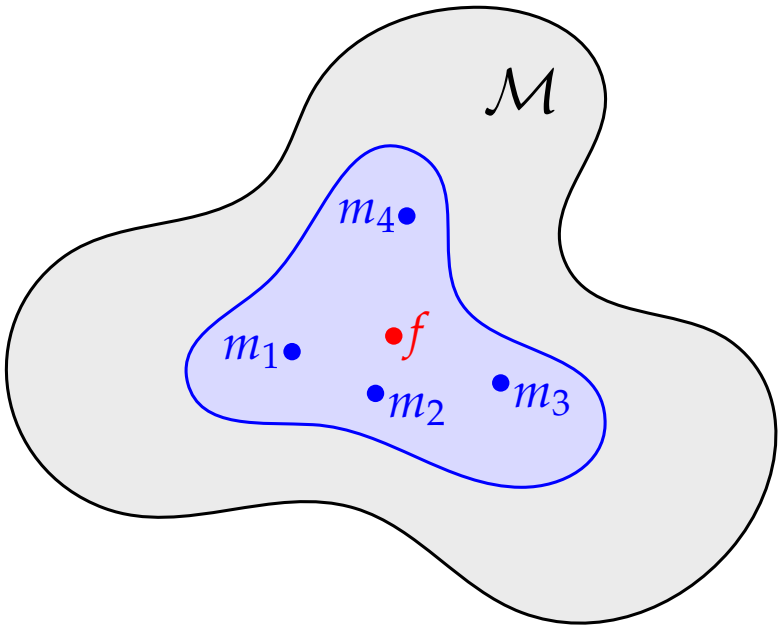
- Note, however, that this holds only for the ideal case that the classifiers are fully independent, which is usually not the case in practice.
- Fortunately, though, improvements are also achieved if the dependence is sufficiently weak, although the gains are naturally smaller.
- Note also that even in the ideal case no gains result (but rather a degradation) if the error probability of an individual classifier exceeds 0.5. Therefore the individual predictors should be accurate.

Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are essentially three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.
- **Statistical Reason:**
  The statistical reason is that in practice any learning method has to work on a finite data set and thus may not be able to identify the correct predictor, even if this predictor lies within the set of models that the learning method can, in principle, return as a result ("hypothesis space", "model class").
- Rather it is to be expected that several predictors yield similar accuracy.
- Since there is no sufficiently clear evidence which model is correct or best, there is a certain risk that the learning method selects a suboptimal model.
- By removing the requirement to produce a single model, it becomes possible to "average" over many or even all of the good models.
- This reduces the risk of excluding the best predictor and reduces the influence of actually bad models.

Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are essentially three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.
- **Statistical Reason**



- The gray area $\mathcal{M}$ is the space of all attainable models (or *hypotheses*).
  ("model class" in lecture "Elementary Data Mining")
- The blue area is the set of models that give good accuracy on the training data.
- m_1, m_2, m_3 and m_4 are individual models, e.g. the individual models of an ensemble.
- f is the best model, the true hypothesis.
- The average prediction of the hypotheses m_1, m_2, m_3 and m_4 is close(r) to the prediction of the best model f.

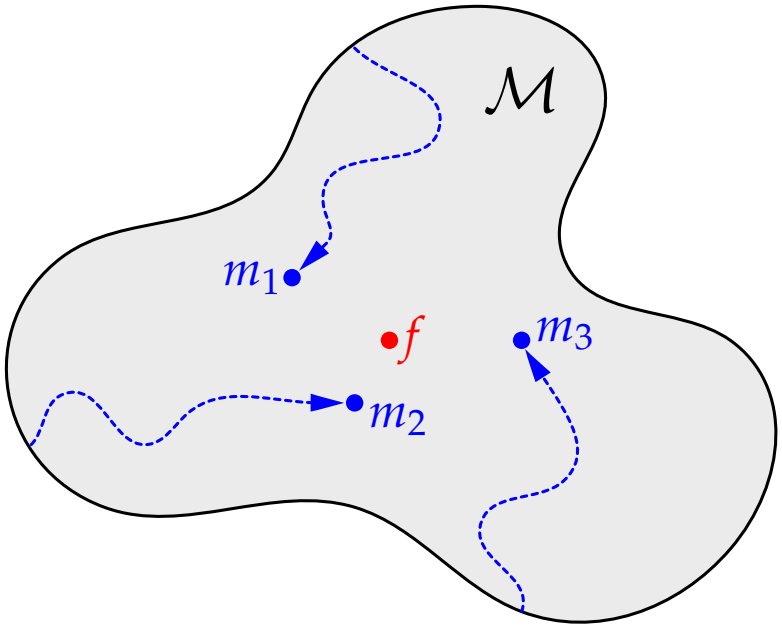
Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are essentially three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.
- **Computational Reason:**
  The computational reason refers to the fact that learning algorithms usually cannot traverse the complete model space (no exhaustive search), but must employ certain heuristics in order to find a good model.

  Examples of model finding / model building heuristics:
  - decision trees: greedy selection of test attribute / split rule
  - neural networks: gradient descent on error function
- Since these heuristics may yield suboptimal models (for example, local minima of the error function), a suboptimal model may be chosen by the learning method.
- However, if several models constructed with heuristics are combined in an ensemble, the result may be a better approximation of the true dependence between the inputs and the target variable.

Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are essentially three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.
- **Computational Reason**



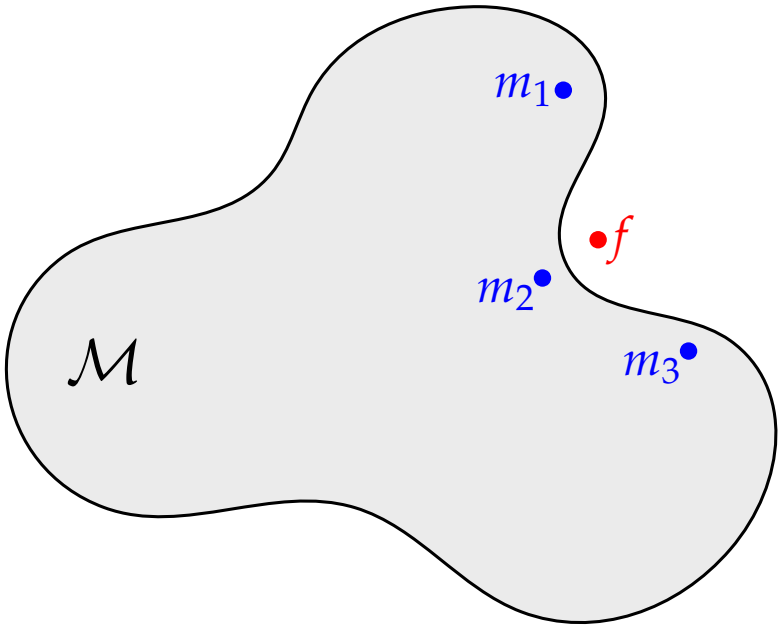
- The gray area $\mathcal{M}$ is the space of all attainable models (or *hypotheses*).
  ("model class" in lecture "Elementary Data Mining")
- m_1, m_2 and m_3 are individual models, e.g. the individual models of an ensemble.
- The dashed blue lines are the trajectories of models traversed by heuristics.
- f is the best model, the true hypothesis.
- The average prediction of the models m_1, m_2 and m_3 is close(r) to the prediction of the best model f.

Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are essentially three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.
- **Representational Reason:**
  The representational reason is that for basically all learning methods, even the most flexible ones, the class of models that can be learned is limited and thus it may be that the true model cannot be represented accurately.
- By combining several models in an ensemble, the model space can be enriched, that is, the ensemble may be able to represent a dependence between the inputs and the target variable that cannot be expressed by any of the individual models the learning method is able to produce.
- From a representational point of view, ensemble methods allow for reducing the bias of a learning algorithm by extending its model space, while the statistical and computational reasons indicate that they can also reduce the variance.
- In this sense, ensemble methods are able to sever the usual link between (machine learning) bias and variance.

Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are essentially three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.
- **Representational Reason**



- The gray area $\mathcal{M}$ is the space of all attainable models (or *hypotheses*).
  ("model class" in lecture "Elementary Data Mining")
- m_1, m_2 and m_3 are individual models, e.g. the individual models of an ensemble.
- f is the best model, the true hypothesis. However, f lies outside of $\mathcal{H}$.
- The average prediction of the models m_1, m_2 and m_3 is close(r) to the prediction of the true hypothesis f.

  $\Rightarrow$ Ensemble methods can enrich the space of representable hypotheses.

Ensemble Methods: Bayesian Voting / Averaging

- In pure **Bayesian voting/averaging** the set of all possible models in a user-defined hypothesis space is enumerated to form the ensemble.
- The predictions of the individual models are combined weighted with the posterior probability of the model given the training data.

  That is, models that are unlikely to be correct given the data have a low influence on the ensemble prediction, models that are likely have a high influence.
- The likelihood of the model given the data can often be computed as

$$P(m \mid \mathbf{D}) \propto P(\mathbf{D} \mid m)P(m),$$

  where m is the model, $\mathbf{D}$ the data, $P(m)$ the prior probability of the model (often assumed to be the same for all models — informationless prior), and $P(\mathbf{D} \mid m)$ the data likelihood given the model.
- Theoretically, Bayesian voting/averaging is the optimal combination method, because all possible models are considered and their relative influence reflects their likelihood given the data.

Ensemble Methods: Bayesian Voting / Averaging

- **Theoretically, Bayesian voting/averaging is the optimal combination method**, because all possible models are considered and their relative influence reflects their likelihood given the data.
- In practice, however, it suffers from several drawbacks:
  - It is rarely possible to actually enumerate all models in the hypothesis space that is (implicitly) defined by a learning method.

    For example, even if we restrict the tree size, it is usually infeasible to enumerate all decision trees that could be constructed for a given classification problem.
  - In order to overcome this problem, model sampling methods are employed, which ideally should select a model with a probability that corresponds to their likelihood given the data.
  - However, most such methods are biased and thus usually do not yield a representative sample of the total set of models, sometimes seriously degrading the ensemble performance.

Ensemble Methods: Bagging

- The method of **bagging (bootstrap aggregating)** predictors can be applied with basically any learning algorithm.
- The basic idea is to select a single learning algorithm (most studied in this respect are decision tree inducers) and to learn several models by providing it each time with a *different random sample of the training data*.
- The sampling is carried out with replacement (bootstrapping). Each sample is exactly as large as the original training data set.

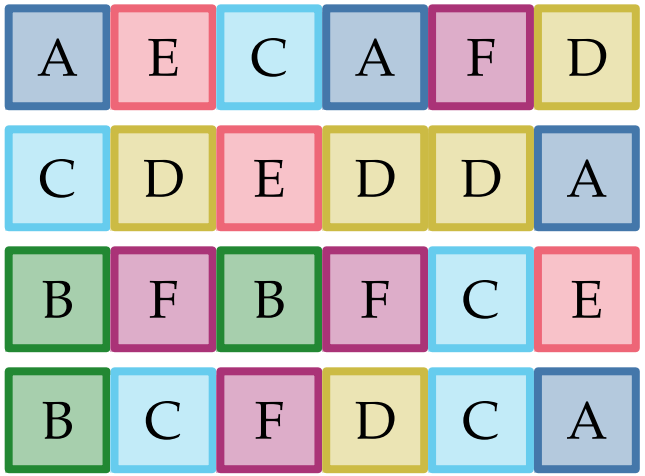
  ⇒ more on next slides
- Especially if the learning algorithm is **unstable** (like decision tree inducers, where a small change of the data can lead to a considerably different decision tree), the resulting models will usually be fairly **diverse**, thus satisfying one of the conditions needed for ensemble methods to work.
- The predictions of the individual models are then combined by simple **majority voting** or by **averaging** them (with the same weight for each model).

Bootstrap Samples

- Given a data set D with n cases/data points, a **bootstrap sample** (or *replicate*) is a sample of size n that is drawn from D *with replacement*.
- Because drawing is done *with replacement*, some cases/data points may occur multiple times, while others may not occur at all (to compensate duplicates).

original data set

bootstrap samples



- In the original data set, data points are considered as being unique.
- Note how in these example bootstrap samples some data points occur once, some 2 or 3 times, some do not occur.
- In principle, each original data point may occur exactly once (permutation), or some data point may occur n times.

  This is fairly unlikely, though.

Bootstrap Samples: Mathematical Excursion

- **How many *different* bootstrap samples can be drawn from a data set of size n?**
- Consider the following **box/urn model**:
  - There are n (distinguishable) boxes/urns,
    which represent the different cases/data points.
  - There are n (indistinguishable) balls,
    which represent the individual drawings.
- Each possible distribution of the balls into the boxes/urns
  represents one possible bootstrap sample:
  - The number of balls in the 1st box
    represents the number of times case 1 is drawn (may be zero).
  - The number of balls in the 2nd box
    represents the number of times case 2 is drawn (may be zero).
  - The number of balls in the i-th box
    represents the number of times case i is drawn (may be zero).

Bootstrap Samples: Mathematical Excursion

- How many *different* bootstrap samples can be drawn from a data set of size n?

  $\Rightarrow$ **In how many different ways can n balls be distributed into n boxes?**

- Suppose the boxes are placed side by side in a row.

  Then we have to compute the number of different sequences of

  - n (indistinguishable) balls and
  - $n-1$ (indistinguishable) walls between boxes.

- In total, there are $2n-1$ objects (n balls plus $n-1$ walls).

  By choosing the n locations of the balls in a sequence of $2n-1$ objects
  (or by choosing the $n-1$ locations of the walls in this sequence)
  we choose one placement of the balls into the boxes. (e.g. [• •| |•|...])

- This yields as the total number of different bootstrap samples:

$$N_{\text{bootstrap}}(n) = \binom{2n-1}{n} = \binom{2n-1}{n-1} = \frac{(2n-1)!}{n!(n-1)!}.$$

Bootstrap Samples: Mathematical Excursion

- **On average, how many *different* cases/data points are in a bootstrap sample?**
- Since all n cases/data points in the original data set are treated the same (they have the same probability of being selected in an individual drawing), the expected fraction of different cases/data points equals the probability that a certain specific case/data point x gets selected.
- In an individual drawing, the probability that x gets selected is $P_1^{\oplus}(x) = \frac{1}{n}$. Therefore, the probability that it does *not* get selected is $P_1^{\ominus}(x) = 1 - \frac{1}{n}$.
- In total, n drawings are executed, in each of which x may get selected. The probability that x does not get selected in any of them is $P_n^{\ominus}(x) = (1 - \frac{1}{n})^n$.
- Hence the probability that x gets selected at least once is $P_n^{\oplus}(x) = 1 - (1 - \frac{1}{n})^n$.
- If we let the data set size n go to infinity, we get $P_\infty^{\ominus}(x) = \lim_{n \to \infty}(1 - \frac{1}{n})^n = \frac{1}{e}$ and therefore $P_\infty^{\oplus}(x) = 1 - \frac{1}{e}$.
- For large n the average number of different cases in a bootstrap sample is therefore approximately $(1 - \frac{1}{e})n \approx 0.632n$.

Ensemble Methods: Bagging

- **Bagging (bootstrap aggregating)** effectively yields predictions from an "average model," even though this model does not exist in simple form — it may not even lie in the hypothesis space of the learning algorithm.
- It has been shown that bagging reduces the risk of overfitting the training data (because each subsample has different special properties) and thus produces very robust predictions.
- Experiments show that bagging yields very good results especially for noisy data sets, where the sampling seems to be highly effective to avoid any adaptation to the noise data points.
- A closely related alternative to bagging are **cross-validated committees**.
- Instead of resampling the training data with replacement (bootstrapping) to generate the different predictors of an ensemble, the predictors learned during a cross-validation run are combined with equal weights in a majority vote or by averaging.

Bagging and Cross-Validation

- In bagging, a specific bootstrap sample is also called the **in-the-bag data set**.
  For large data sets, it contains approximately $(1 - \frac{1}{e})n$ data points on average.
- Its complement w.r.t. the given data set is also called the **out-of-bag data set**.
  For large data sets, it contains approximately $\frac{n}{e}$ data points on average.

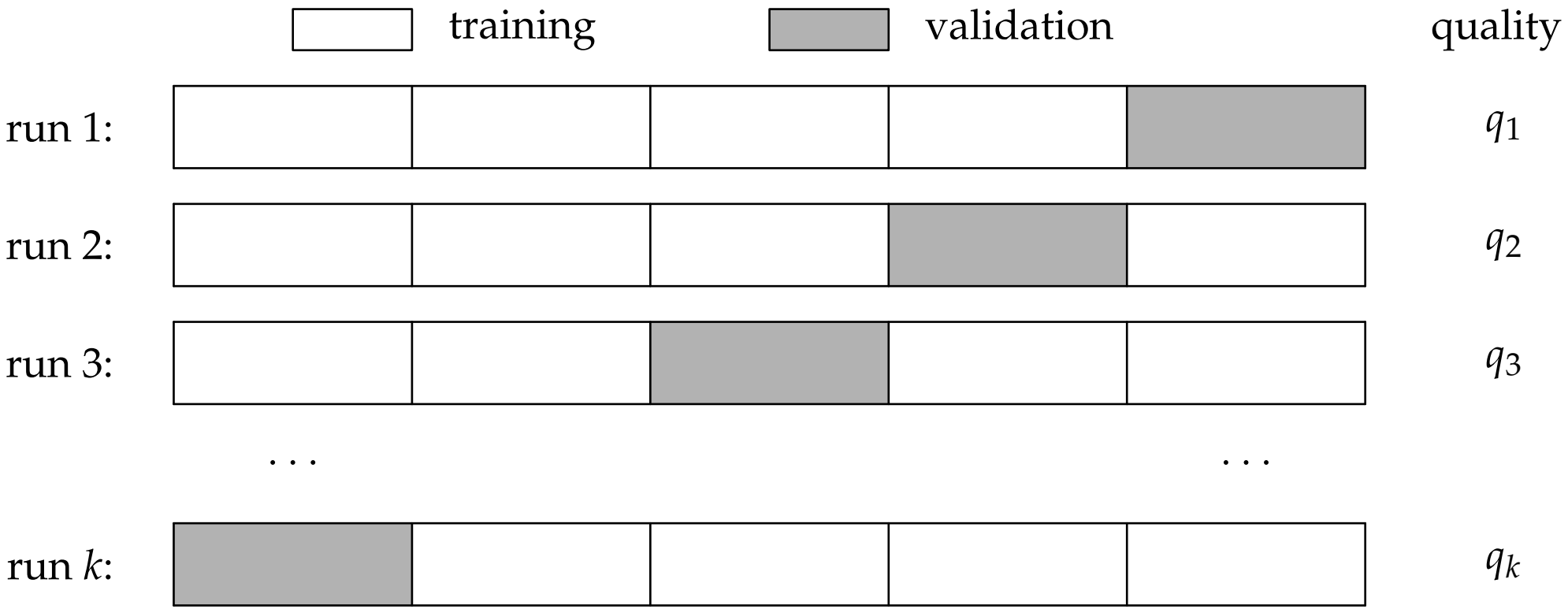
  (The out-of-bag data set contains all data points *not* in the bootstrap sample.)
- Since the out-of-bag data set has not been used to induce the predictor,
  it can be used to estimate the predictor's error on new data (**out-of-bag error**).

  $\Rightarrow$ bagging has a built-in validation scheme
- This is analogous to **cross-validation**, where a predictor's error
  on the **hold-out fold** can be used to estimate the error one new data.
- However, there are significant differences:
  - In cross-validation, each data point is *exactly once* in the hold-out fold.
  - In bagging, a data point may occur in multiple out-of-bag data sets,
    and some data points may not occur in any out-of-bag data set.

Reminder: Cross-Validation

- General method to assess / to predict the performance of models created by a given model building procedure (*not* of a specific model!).
- Serves the purpose to **estimate the error (rate) on new example cases**.
- Procedure of cross-validation:
  - Split the given data set into k so-called *folds* of equal size (**k-fold cross-validation**). Often recommended: $k = 10$.
  - Combine $k - 1$ folds into a training data set, build a classifier, and test it on the k-th fold (the *hold-out fold*).
  - Do this for all k possible selections of $k - 1$ folds and average the error (rates) (or some other quality measure).
- Special case: **leave-1-out cross-validation** (a.k.a. **jackknife method**). (use as many folds as there are example cases)
- The final classifier is generated from the full data set (in order to exploit all available information).

Reminder: Cross-Validation



Build and evaluate a model in each run, then average their quality. $\hat{q} = \frac{1}{k} \sum_{i=1}^{k} q_i$

- **Cross-validation does not assess a single model** (since k models are built).
  It assesses the expected performance of models built with a certain procedure.

Bagging and Cross-Validation

- In bagging, one should *not* simply average the out-of-bag errors.
  - Evaluating the predictor induced from a bootstrap sample on the corresponding out-of-bag data set only evaluates this predictor.
  - However, the bagged predictor averages or takes the majority vote of *multiple predictors* (this is what yields performance improvements).
  - Hence simply averaging the out-of-bag errors is overly pessimistic.
- Better procedure to compute an **out-of-bag error for an ensemble**:
  - Form the union $\mathbf{D}_{\text{oob}} \subseteq \mathbf{D}$ of all out-of-bag data sets.
  - For each data point $(\vec{x}, y) \in \mathbf{D}_{\text{oob}}$ collect all predictors for the induction of which it was in the out-of-bag data set.
  - Predict the target value of the data point with the collected predictors (using averaging or majority voting). (use a specific subset of the ensemble)
  - Compute the error rate on the data set $\mathbf{D}_{\text{oob}}$ and use this as an estimate of the performance on new data.

Ensemble Methods: Random Subspace Selection

- While bagging obtains a set of diverse predictors by randomly varying the training data, **random subspace selection** employs a random selection of the features for this purpose.
- That is, all data points are used in each training run, but the features the model construction algorithm can use are randomly selected.
- With a learning algorithm like a decision tree inducer (for which random subspace selection was first proposed), the available features may even be varied each time a split has to be chosen, so that the whole decision tree can potentially use all features.
- Combining random subspace selection with bagging is a highly effective and strongly recommended method if accuracy is the main goal.
- Applied to decision trees this method has been named **random forests**, which is known to be among the most accurate classification methods.
- However, the often huge number of trees destroys the advantage that decision trees are easy to interpret and can be checked for plausibility.

Ensemble Methods: Injecting Randomness

- Both bagging and random subspace selection employ random processes in order to obtain diverse predictors.
- Of course, this approach can be generalized to the principle of **injecting randomness** into the learning process.
  - Bagging: randomly select training data sets (as bootstrap samples)
  - Random Subspace Selection: randomly select feature subsets
- For example, such an approach is very natural and straightforward for **artificial neural networks**:

  ⇒ Lecture "Artificial Neural Networks and Deep Learning"

  Different initializations of the connection weights and bias values often yield different learning results (different local optima), which may then be used as the members of an ensemble.
- Alternatively, the network structure can be modified, for example, by randomly deleting a certain fraction of the connections between two consecutive layers or a certain fraction of neurons (a.k.a. "dropout").

Ensemble Methods: Boosting

- **Boosting** constructs predictors progressively, with the prediction results of the model learned last influencing the construction of the next model.
- Like bagging, boosting varies the training data. However, instead of drawing random samples, boosting always works on the complete training data set, and maintains and manipulates a **data point weight** for each training example.
- For low noise data, boosting clearly outperforms bagging and random subspace selection in experimental studies.
- However, if the training data contains noise, the performance of boosting can degrade quickly, because it tends to focus on the noise data (which are necessarily difficult to classify and thus receive high weights after fairly few steps).
- As a consequence, boosting tends to overfit the data. For noisy data bagging and random subspace selection yield better results.
- Boosting is usually described for classification problems with two classes, which are assumed to be encoded by $+1$ and -1.

Ensemble Methods: AdaBoost

- The best-known boosting approach is **AdaBoost** and works as follows:

  [Freund & Schapire 1995–1998]

- Initially, all data point weights are equal and therefore set to $w_i = 1, i = 1, \ldots n$, where $n = |\mathbf{D}|$ is the size of the data set.

- After a predictor m_t with $m_t(\vec{x}) \in \{-1, +1\}$ has been constructed in step t with current weights $w_{i,t}$, $i = 1, \ldots, n$, it is applied to the training data and

$$e_t = \frac{\sum_{i=1}^{n} \mathbb{1}(y_i \neq m_t(\vec{x}_i))\, w_{i,t}}{\sum_{i=1}^{n} w_{i,t}} \qquad \text{and} \qquad \alpha_t = \frac{1}{2} \ln\left(\frac{1 - e_t}{e_t}\right) \qquad \mathbb{1}(\varphi) = \begin{cases} 1 \text{ if } \varphi \text{ is true,} \\ 0 \text{ otherwise.} \end{cases}$$

  are computed, where $\vec{x}_i$ is the input vector, $y_i \in \{-1, +1\}$ the class of the i-th training example, and $m_t(\vec{x}_i)$ the prediction of the model m_t for the input $\vec{x}_i$.

- The data point weights are then updated according to

$$w_{i,t+1} = c_{t+1}\, w_{i,t}\, \exp(-\alpha_t\, y_i\, m_t(\vec{x}_i)),$$

  where c_{t+1} is a normalization constant chosen in such a way that $\sum_{i=1}^{n} w_{i,t+1} = n$.

Ensemble Methods: AdaBoost

- The procedure of learning a predictor and updating the data point weights is repeated a user-specified number of times $t_{\max}$.
- The constructed ensemble classifies new data points by majority voting, with each model m_t weighted with α_t. (definition of α_t: see preceding slides)
- That is, the joint prediction is (the sign of) the function

$$m_{\text{joint}}(\vec{x}_i) = \sum_{t=1}^{t_{\max}} \alpha_t m_t(\vec{x}_i).$$

- Since there is no convergence guarantee and the performance of the ensemble classifier can even degrade after a certain number of steps, the inflection point of the error curve over t is often chosen as the ensemble size.
- Reminder: if the training data contains noise, the performance of boosting can degrade quickly, because it tends to focus on the noise data (which are necessarily difficult to classify and thus receive high weights after fairly few steps).

  As a consequence, boosting tends to overfit the data.

Ensemble Methods: AdaBoost

- AdaBoost can be seen as minimizing the **loss function** [Schapire & Singer 1998]

$$\mathcal{L}(\mathbf{D}) = \left[\frac{1}{n}\right] \sum_{i=1}^{n} \exp\Big(-\sum_{t=1}^{t_{\max}} \alpha_t \, y_i \, m_t(\vec{x}_i)\Big)$$

  where $\mathbf{D} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\}$ with $y_i \in \{-1, +1\}$ is the training data set.

- Optimization is done w.r.t. the model weighting factors α_t,
  which are to be chosen in such a way that $\mathcal{L}(\mathbf{D})$ becomes as small as possible.

- For some model m (either some m_t or m_{joint}), the quantity

$$\gamma_i = y_i \, m(\vec{x}_i)$$

  is called the **margin** of the model m at the point $(\vec{x}_i, y_i)$.
  It is positive for a correct prediction and negative for an incorrect one.

- With this notion, we may say that AdaBoost tries to minimize the sum of
  the exponentials of the weighted negative margins of the individual predictors.

Excursion: Exponential Loss

- **Loss functions** represent the price paid for inaccuracy of predictions (usually: in classification problems).
- The usual goal is to minimize the **expected loss** (also known as **risk**)

$$\mathbb{E}_{\vec{X},Y}(\mathcal{L}(m(\vec{X}),Y)) = \int_{\mathcal{X}\times\mathcal{Y}} \mathcal{L}\big(m(\vec{x}),y\big)\, p(\vec{x},y)\, d\vec{x}\, dy,$$

  where $p(\vec{x},y)$ is the probability density function of the data generating process, which may also be written as $p(y \mid \vec{x})p(\vec{x})$.
- In (binary) classification, many loss functions are written solely in terms of the **margin** $\gamma = y\, m(\vec{x})$ at a point $(\vec{x},y)$, that is,

$$\mathcal{L}\big(m(\vec{x}),y\big) = \mathcal{L}^\circ\big(y\, m(\vec{x})\big) = \mathcal{L}^\circ(\gamma)$$

  with $\mathcal{L}^\circ\colon \mathbb{R} \to \mathbb{R}$. These are called **margin-based loss functions**.
- The **exponential loss function** is a margin-based loss function that uses

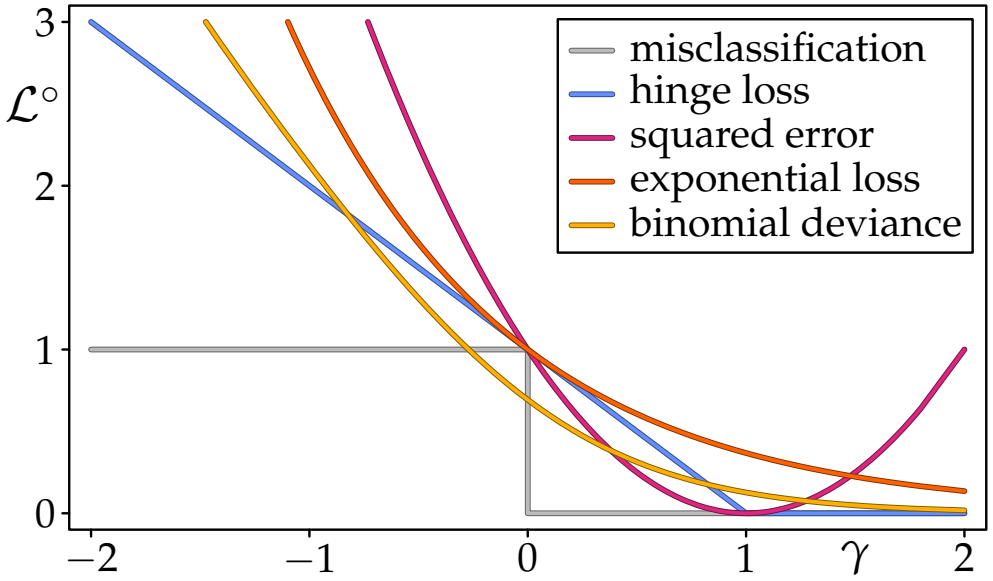
$$\mathcal{L}\big(m(\vec{x}),y\big) = \mathcal{L}^\circ\big(y\, m(\vec{x})\big) = \mathcal{L}^\circ(\gamma) = e^{-\gamma} = e^{-y\, m(\vec{x})}.$$

Excursion: Margin-Based Loss Functions

- In (binary) classification, many loss functions are written in terms of the **margin** $\gamma = y\, m(\vec{x})$ at a point $(\vec{x}, y)$, that is,

  $$\mathcal{L}\big(m(\vec{x}), y\big) = \mathcal{L}^\circ\big(y\, m(\vec{x})\big) = \mathcal{L}^\circ(\gamma)$$

  with $\mathcal{L}^\circ : \mathbb{R} \to \mathbb{R}$. These are called **margin-based loss functions**.



- misclassification: $\mathcal{L}^\circ(\gamma) = \mathbb{1}(\gamma < 0)$ (constant almost everywhere, not continuous)
- hinge loss: $\mathcal{L}^\circ(\gamma) = \max(0, 1 - x)$ (used in support vector machines)
- squared error: $\mathcal{L}^\circ(\gamma) = (1 - \gamma)^2$ (not monotone decreasing)
- exponential loss: $\mathcal{L}^\circ(\gamma) = e^{-\gamma}$ (used in AdaBoost)
- binomial deviance: $\mathcal{L}^\circ(\gamma) = \ln(1 + e^{-2\gamma})$ (log-likelihood, cross-entropy)

(factor 2 to match the result to that of exponential loss, see later)

Ensemble Methods: AdaBoost

Derivation of the Model Weights α_t (using an additive modeling view)

- In step $\tau-1$ of an AdaBoost approach the joint classifier is

$$m_{\tau-1}^{\text{joint}}(\vec{x}_i) = \sum_{t=1}^{\tau-1} \alpha_t m_t(\vec{x}_i).$$

- In step τ we have to find the best model m_τ to add and its weight α_τ:

$$m_{\tau}^{\text{joint}}(\vec{x}_i) = m_{\tau-1}^{\text{joint}}(\vec{x}_i) + \alpha_\tau m_\tau(\vec{x}_i)$$

- The total error of the model m_τ^{joint} is the sum of its **exponential loss**:

$$\begin{aligned}
\mathcal{L}(\mathbf{D}) &= \sum_{i=1}^{n} \exp\Big(-y_i\, m_{\tau}^{\text{joint}}(\vec{x}_i)\Big) \\
&= \sum_{i=1}^{n} \exp\Big(-y_i\, m_{\tau-1}^{\text{joint}}(\vec{x}_i) - \alpha_\tau\, y_i\, m_\tau(\vec{x}_i)\Big) \\
&= \sum_{i=1}^{n} \exp\Big(-y_i\, m_{\tau-1}^{\text{joint}}(\vec{x}_i)\Big) \exp\big(-\alpha_\tau\, y_i\, m_\tau(\vec{x}_i)\big)\,.
\end{aligned}$$

Derivation of the Model Weights α_t

- By defining $w_{i,1} = 1$ and $w_{i,t} = \exp(-y_i\, m_{t-1}^{\text{joint}}(\vec{x}_i))$, $t = 2, \ldots, \tau$, we get

$$\mathcal{L}(\mathbf{D}) = \sum_{i=1}^{n} w_{i,\tau}\, \exp(-\alpha_\tau\, y_i\, m_\tau(\vec{x}_i))\,.$$

- Splitting into correctly and incorrectly classified data points yields

$$\begin{aligned}\mathcal{L}(\mathbf{D}) &= \sum_{i=1}^{n} \mathbb{1}(y_i = m_\tau(\vec{x}_i))\, w_{i,\tau}\, e^{-\alpha_\tau} + \sum_{i=1}^{n} \mathbb{1}(y_i \neq m_\tau(\vec{x}_i)) w_{i,\tau}\, e^{+\alpha_\tau} \\ &= e^{-\alpha_\tau} \sum_{i=1}^{n} w_{i,\tau} + (e^{+\alpha_\tau} - e^{-\alpha_\tau}) \sum_{i=1}^{n} \mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}.\end{aligned}$$

assuming that $m_\tau(\vec{x}) \in \{-1, +1\}$

$$\mathbb{1}(\varphi) = \begin{cases} 1 \text{ if } \varphi \text{ is true,} \\ 0 \text{ otherwise.} \end{cases}$$

- The sum $\sum_{i=1}^{n} w_{i,\tau}$ is obviously independent of the model m_τ.
  Only the sum $\sum_{i=1}^{n} \mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}$ depends on the model m_τ.
- Hence the best model m_τ is the one minimizing $\sum_{i=1}^{n} \mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}$
  with data point weights $w_{i,\tau} = \exp(-y_i\, m_{\tau-1}^{\text{joint}}(\vec{x}_i))$.

Derivation of the Model Weights α_t

- The best model weight α_τ is determined by exploiting that a necessary condition for a minimum is that the derivative vanishes:

$$\frac{d}{d\alpha_\tau}\mathcal{L}(\mathbf{D}) = \frac{d}{d\alpha_\tau}\Big(\sum_{i=1}^{n}\mathbb{1}(y_i = m_\tau(\vec{x}_i))\, w_{i,\tau}\, e^{-\alpha_\tau} + \sum_{i=1}^{n}\mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}\, e^{+\alpha_\tau}\Big) \stackrel{!}{=} 0$$

- It follows

$$-e^{-\alpha_\tau}\sum_{i=1}^{n}\mathbb{1}(y_i = m_\tau(\vec{x}_i))\, w_{i,\tau} + e^{+\alpha_\tau}\sum_{i=1}^{n}\mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau} = 0$$

$$\Leftrightarrow \quad e^{-\alpha_\tau}\sum_{i=1}^{n}\mathbb{1}(y_i = m_\tau(\vec{x}_i))\, w_{i,\tau} = e^{+\alpha_\tau}\sum_{i=1}^{n}\mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}$$

$$\Leftrightarrow \quad -\alpha_\tau + \ln\Big(\sum_{i=1}^{n}\mathbb{1}(y_i = m_\tau(\vec{x}_i))\, w_{i,\tau}\Big) = \alpha_\tau + \ln\Big(\sum_{i=1}^{n}\mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}\Big)$$

$$\Leftrightarrow \quad \alpha_\tau = \frac{1}{2}\ln\left(\frac{\sum_{i=1}^{n}\mathbb{1}(y_i = m_\tau(\vec{x}_i))\, w_{i,\tau}}{\sum_{i=1}^{n}\mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}}\right)$$

Derivation of the Model Weights α_t

- The derivation on the preceding slide led to

$$\begin{aligned}\alpha_\tau &= \frac{1}{2}\ln\left(\frac{\sum_{i=1}^n \mathbb{1}(y_i = m_\tau(\vec{x}_i))\, w_{i,\tau}}{\sum_{i=1}^n \mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}}\right) \\ &= \frac{1}{2}\ln\left(\frac{\sum_{i=1}^n w_{i,\tau} - \sum_{i=1}^n \mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}}{\sum_{i=1}^n \mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}}\right).\end{aligned}$$

- By computing the weighted error rate e_τ

$$e_\tau = \frac{\sum_{i=1}^n \mathbb{1}(y_i \neq m_\tau(\vec{x}_i))\, w_{i,\tau}}{\sum_{i=1}^n w_{i,\tau}}, \qquad \text{we get} \qquad \alpha_\tau = \frac{1}{2}\ln\left(\frac{1-e_\tau}{e_\tau}\right),$$

  which is half the negative logit function of e_τ. (cf. logistic regression)

- It can be shown that $\mathcal{L}(\mathbf{D})$ is convex and hence that this α_τ yields the global minimum of the loss function.

- Remark: This derivation of the model weights requires $m_t(\vec{x}) \in \{-1, +1\}$ (hard classification).

Ensemble Methods: AdaBoost (Alternative Formulation)

- The best-known boosting approach is **AdaBoost** and works as follows:

  [Freund & Schapire 1995–1998]

- Initially, all data point weights are equal and therefore set to $w_i = 1, i = 1, \ldots n$, where $n = |\mathbf{D}|$ is the size of the data set.

- After a predictor m_t with $m_t(\vec{x}) \in \{-1, +1\}$ has been constructed in step t with current weights $w_{i,t}$, $i = 1, \ldots, n$, it is applied to the training data and

$$o_t = \frac{\sum_{i=1}^{n} w_{i,t}\, y_i\, m_t(\vec{x}_i)}{\sum_{i=1}^{n} w_{i,t}} = \frac{\sum_{i=1}^{n} w_{i,t}\, \gamma_{i,t}}{\sum_{i=1}^{n} w_{i,t}} \qquad \text{and} \qquad \alpha_t = \frac{1}{2} \ln \left(\frac{1 + o_t}{1 - o_t} \right)$$

  are computed, where $\vec{x}_i$ is the input vector, $y_i \in \{-1, 1\}$ the class of the i-th training example, and $m_t(\vec{x}_i)$ the prediction of the model m_t for the input $\vec{x}_i$.

- The data point weights are then updated according to

$$w_{i,t+1} \;=\; c_{t+1}\, w_{i,t}\, \exp(-\alpha_t\, y_i\, m_t(\vec{x}_i)) \;=\; c_{t+1}\, w_{i,t}\, \exp(-\alpha_t\, \gamma_{i,t}),$$

  where c_{t+1} is a normalization constant chosen in such a way that $\sum_{i=1}^{n} w_{i,t+1} = n$.

Ensemble Methods: AdaBoost (Original Formulation)

- The best-known boosting approach is **AdaBoost** and works as follows:

  [Freund & Schapire 1995–1998]

- Initially, all data point weights are equal and therefore set to $w_i = 1, i = 1, \ldots n$, where $n = |\mathbf{D}|$ is the size of the data set.

- After a predictor m_t with $m_t(\vec{x}) \in \{-1, +1\}$ has been constructed in step t with current weights $w_{i,t}$, $i = 1, \ldots, n$, it is applied to the training data and

$$e_t = \frac{\sum_{i=1}^{n} \mathbb{1}(y_i \neq m_t(\vec{x}_i))\, w_{i,t}}{\sum_{i=1}^{n} w_{i,t}} \qquad \text{and} \qquad \alpha_t = \frac{1}{2} \ln \left(\frac{1 - e_t}{e_t} \right)$$

  are computed, where $\vec{x}_i$ is the input vector, $y_i \in \{-1, 1\}$ the class of the i-th training example, and $m_t(\vec{x}_i)$ the prediction of the model m_t for the input $\vec{x}_i$.

- The data point weights are then updated according to

$$w_{i,t+1} = c_{t+1}\, w_{i,t}\, \exp(-\alpha_t\, y_i\, m_t(\vec{x}_i)),$$

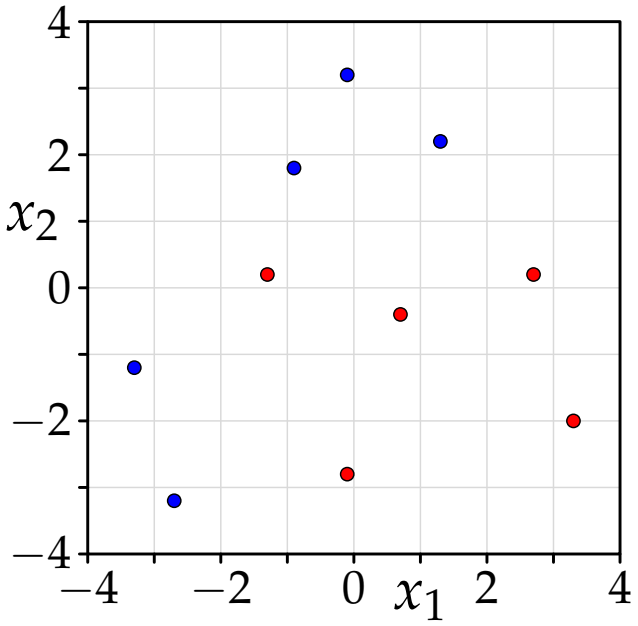
  where c_{t+1} is a normalization constant chosen in such a way that $\sum_{i=1}^{n} w_{i,t+1} = n$.

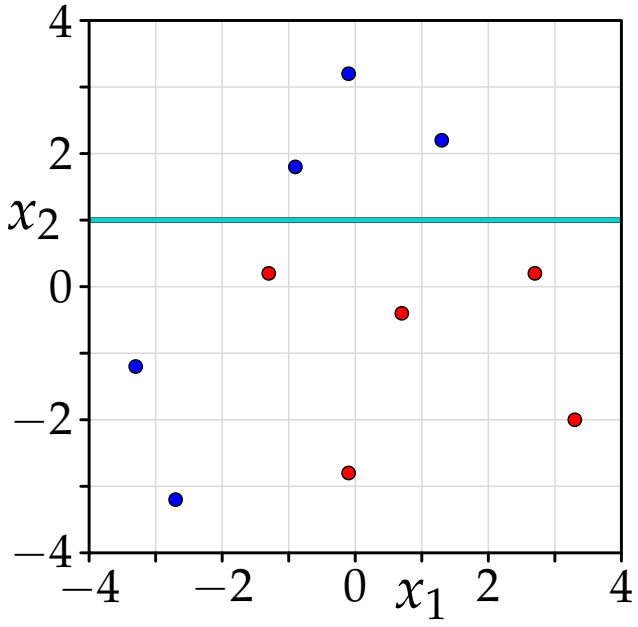
Ensemble Methods: AdaBoost

- Up to now: **Boosting by Weighting** (the data points).
- Problem: Not all model building methods allow for weighted data points.
  - Logistic Regression and Decision Tree Induction do easily.
  - Rule Induction with the A^q Algorithm does not. ($\Rightarrow$ "Advanced Data Mining 1")
    (at least it is not straightforward)
- Can boosting be used with methods that do not allow for weighted data points?
- Answer: Use **Boosting by Sampling** (from the data points).
  - Interpret the weights as a probability distribution over the data points.
    (This requires the data point weights to be normalized to sum 1.)
  - Draw data points from this probability distribution
    (and give each of the drawn data points unit weight).
  - Then the expected number of copies of a data point in the sample
    is proportional to the weight of the data point.

Ensemble Methods: AdaBoost

- **AdaBoost Example 1**
- Simple data set with two classes:
  - 5 data points belong to the **blue** class.
  - 5 data points belong to the **red** class.
- A single **decision tree** would achieve perfect classification.





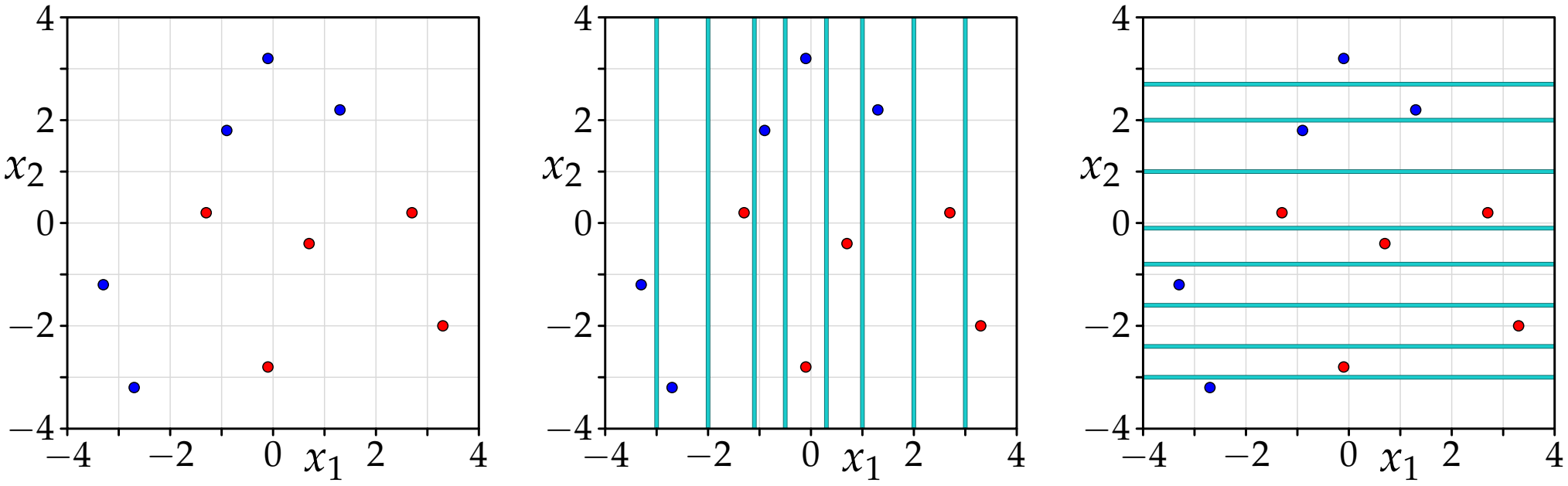
- Here, however: **decision stumps**, e.g.

  If $x_2 \leq 1$, then class **red**.
  If $x_2 > 1$, then class **blue**.

  (only one split, cyan line in left diagram)

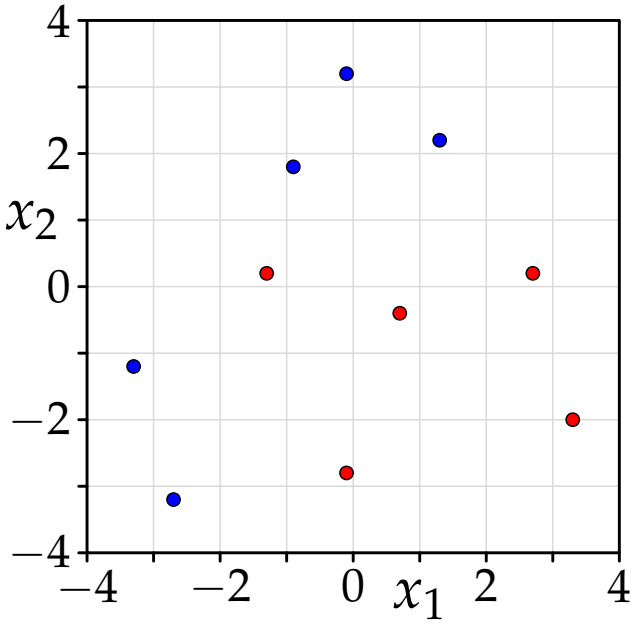
- Apply AdaBoost with decision stumps as the base learning algorithm.

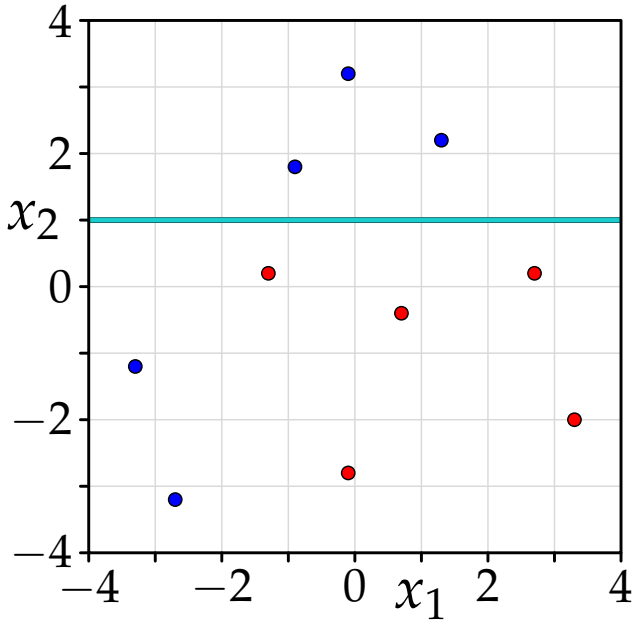
Ensemble Methods: AdaBoost

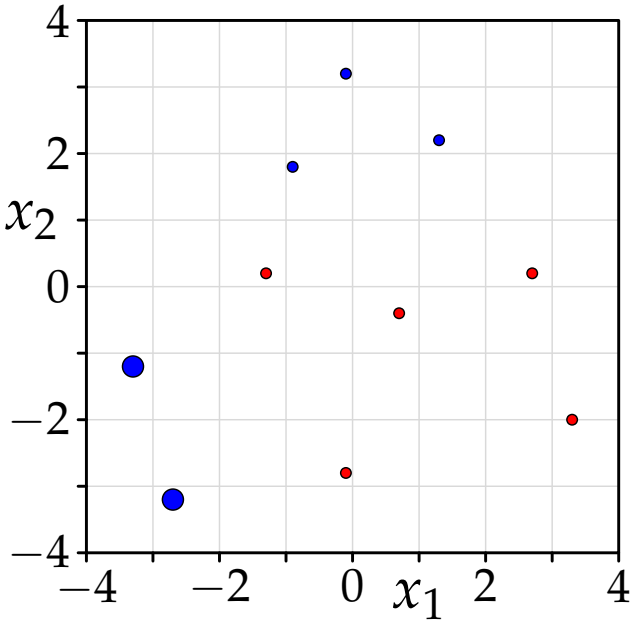


- **Procedure for decision stump selection:**
  - Collect all x-thresholds (middle between two consecutive x-coordinates).
    (cyan lines in middle diagram)
  - Collect all y-thresholds (middle between two consecutive y-coordinates).
    (cyan lines in right diagram)
  - Select threshold with minimum number of errors; here: $x_2 = 1$.

Ensemble Methods: AdaBoost







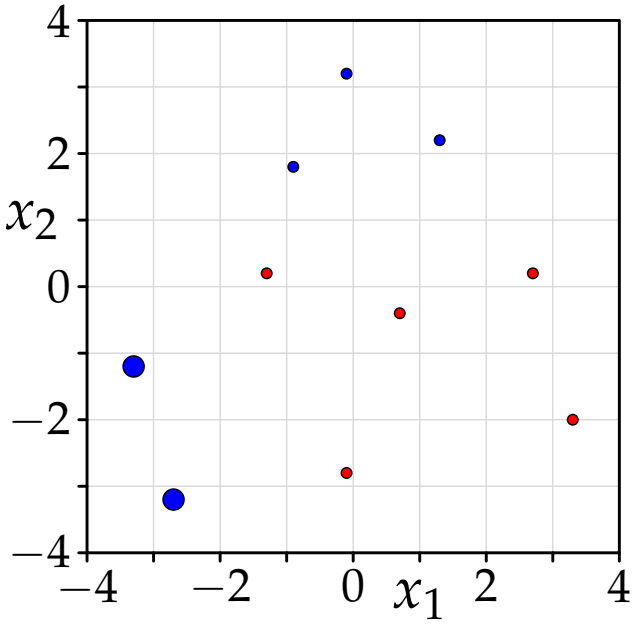
- First model is the decision stump:

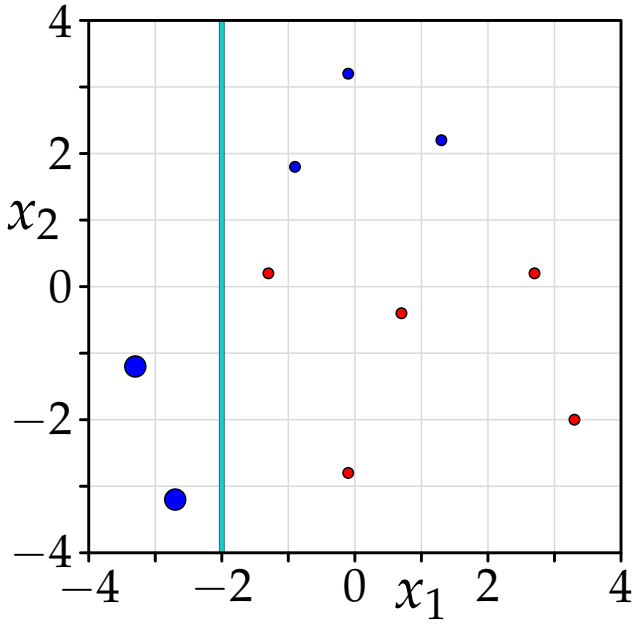
  If $x_2 \leq 1$, then class **red**.
  If $x_2 > 1$, then class **blue**.

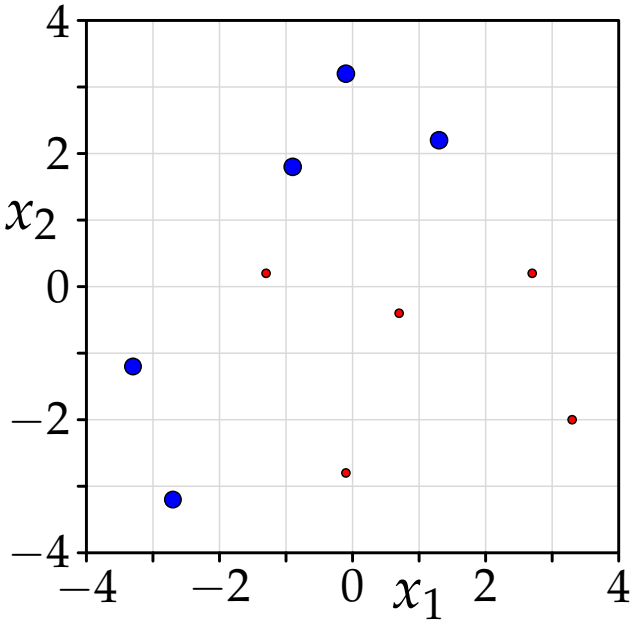
Data point weights are shown by the sizes of the dots; dot *area* is proportional to weight.

- This decision stump produces two errors (blue points on the left).
  - The weight of the two errors is increased $(1 \rightarrow 2.5)$.
  - The weight of all other data points is reduced $(1 \rightarrow 0.625)$.

Ensemble Methods: AdaBoost







- Second model is the decision stump:

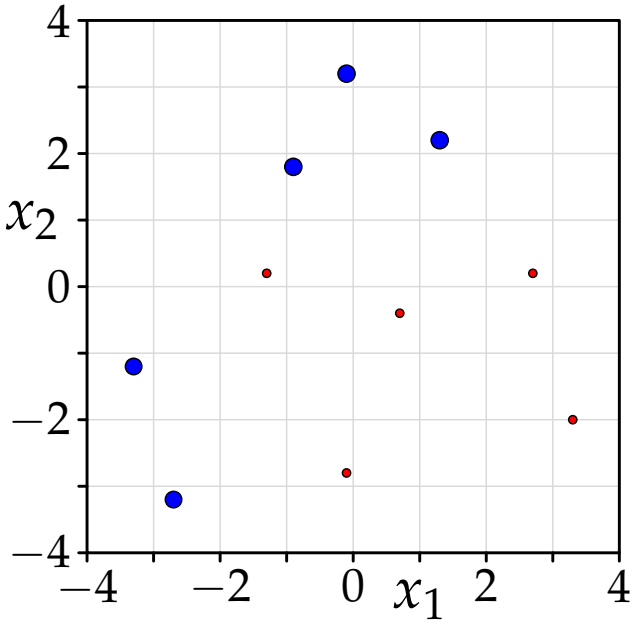
  If $x_1 \leq -2$, then class **blue**.
  If $x_1 > -2$, then class **red**.

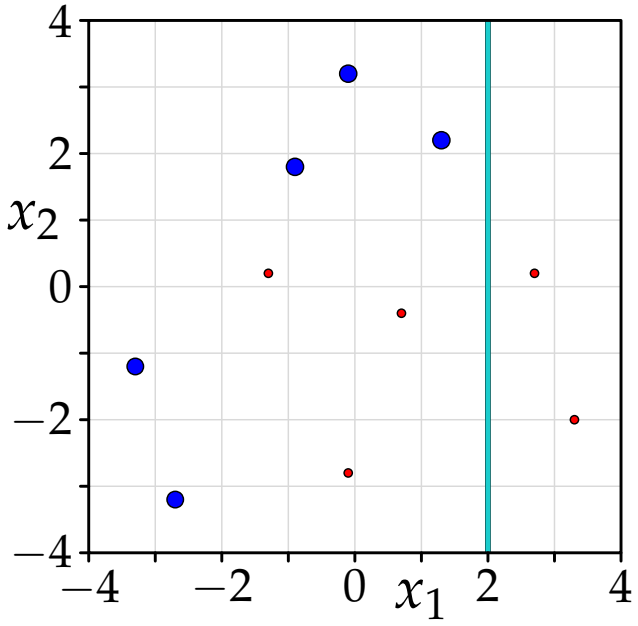
- This decision stump produces $3 \times 0.625 = 1.875$ errors.

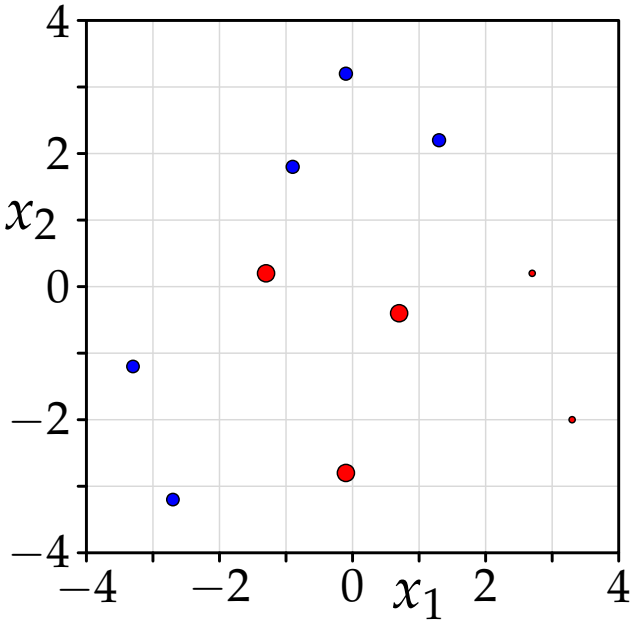
  Mind the data point weights!

- Misclassified points are weighted up.
- Correctly classified points are weighted down.
- Right diagram shows new data point weights.

Ensemble Methods: AdaBoost







- Third model is the decision stump:

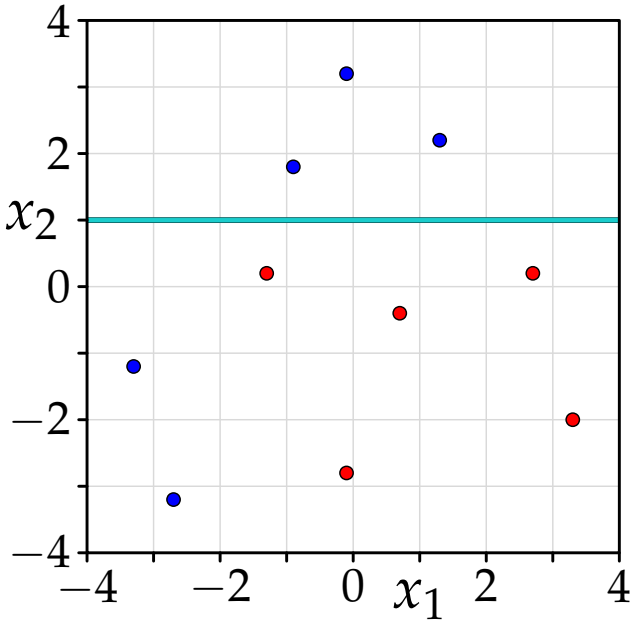
  If $x_1 \leq +2$, then class **blue**.
  If $x_1 > +2$, then class **red**.

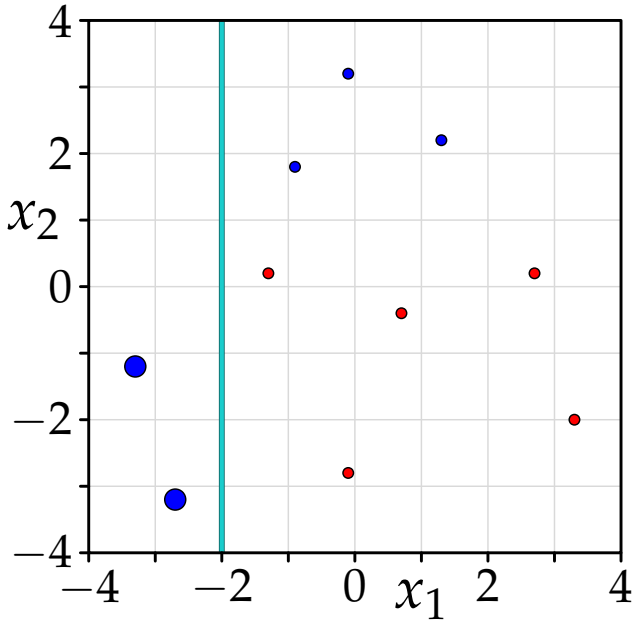
- This decision stump produces $3 \times 0.3846 = 1.1538$ errors.

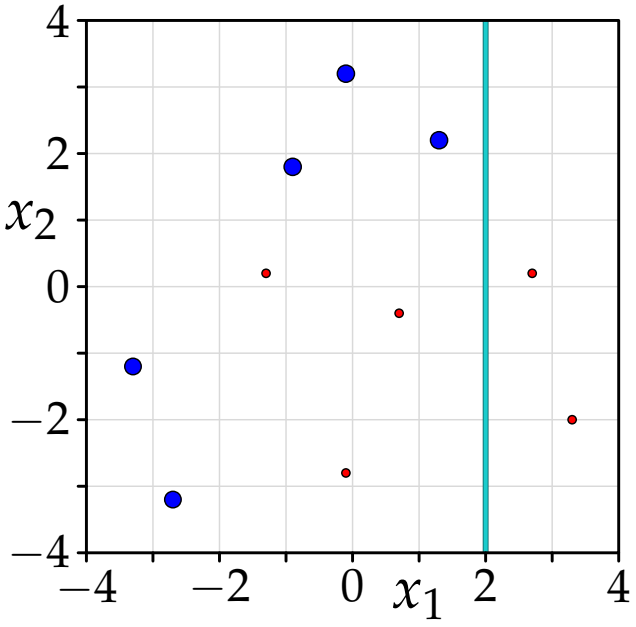
  Mind the data point weights!

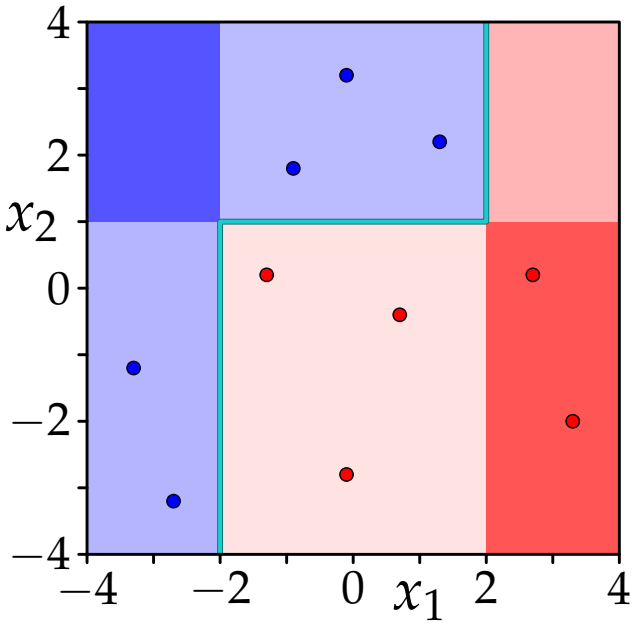
- Misclassified points are weighted up.
- Correctly classified points are weighted down.
- Right diagram shows new data point weights.

Ensemble Methods: AdaBoost





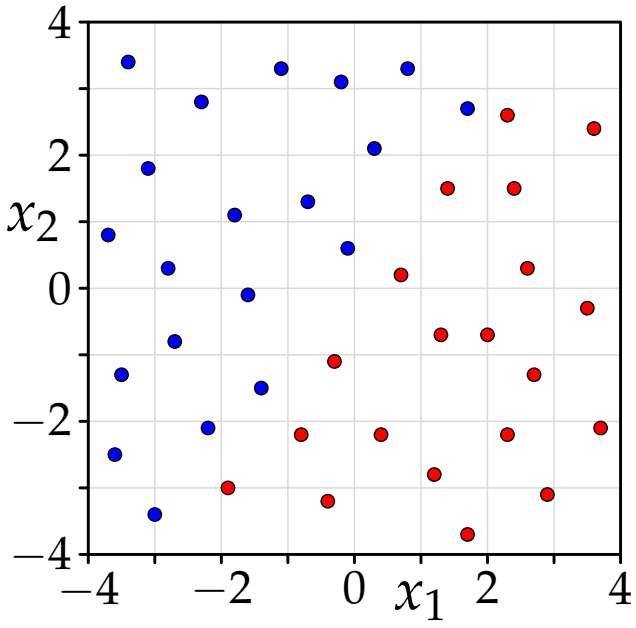


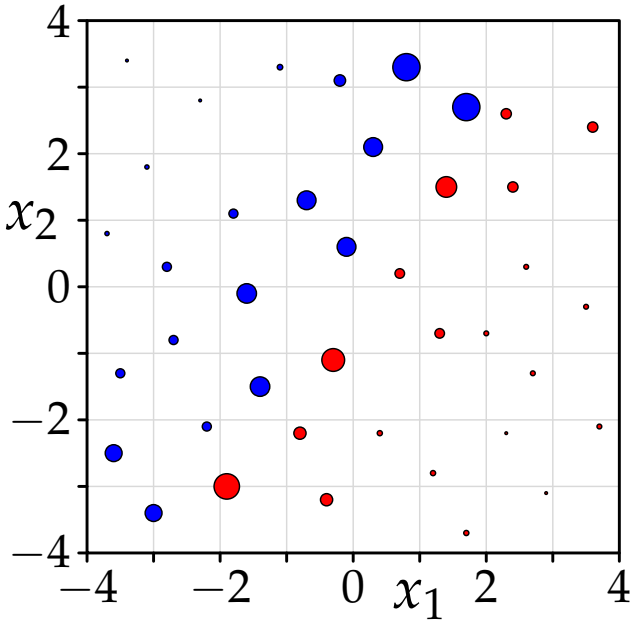


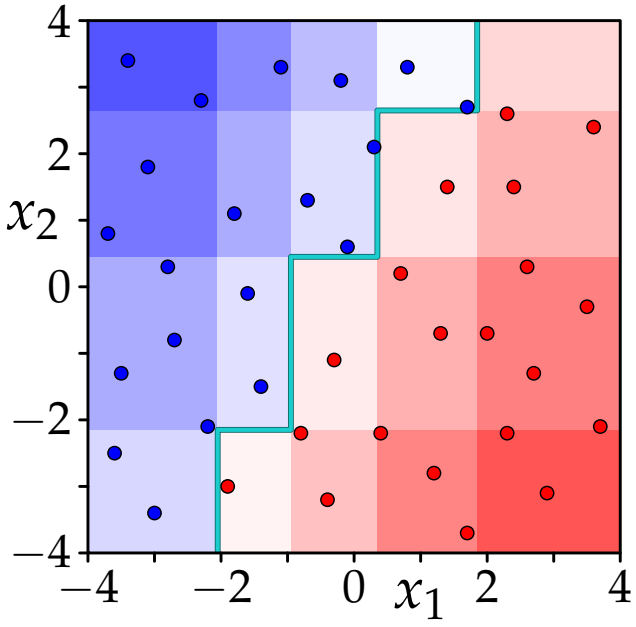
- The joint classifier (shown on the left, weighted sum of decision stumps) classifies all data points correctly.
- Decision boundary (shown in cyan) is **not** representable by a single decision stump.
- This illustrates the **representational reason** why ensemble methods work.

Ensemble Methods: AdaBoost

- **AdaBoost Example 2**
- Simple data set with two classes:
  - 20 data points belong to the **blue** class.
  - 20 data points belong to the **red** class.
- Classification with **decision stumps**.
  (just more complex data set)



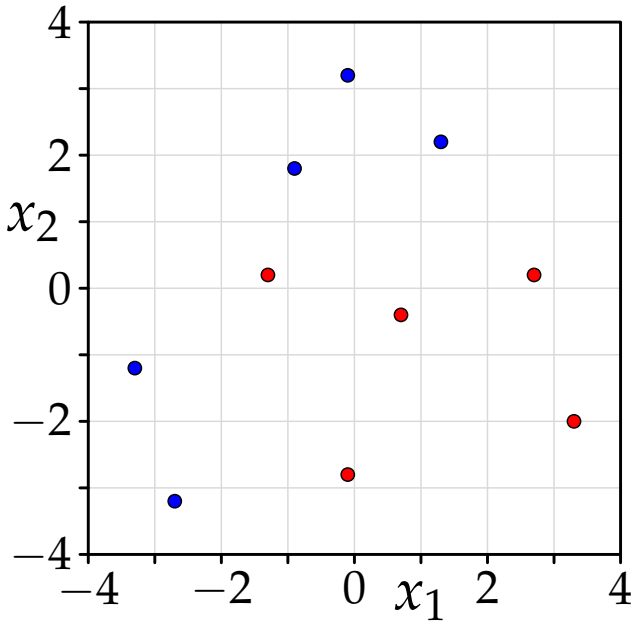


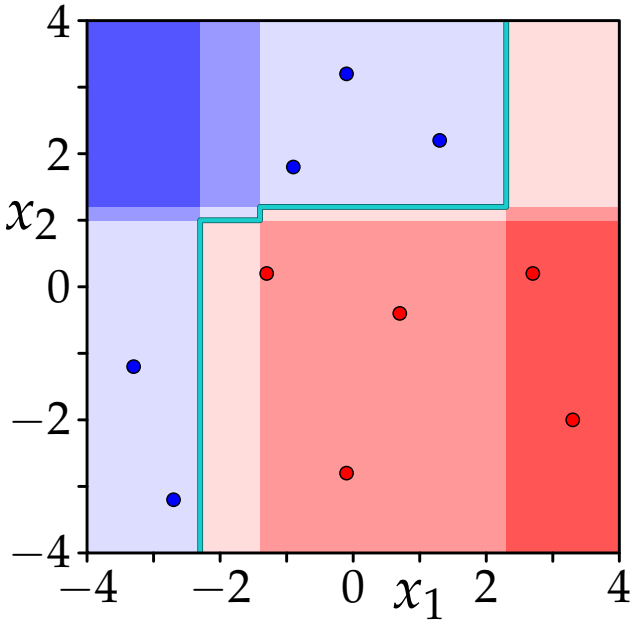


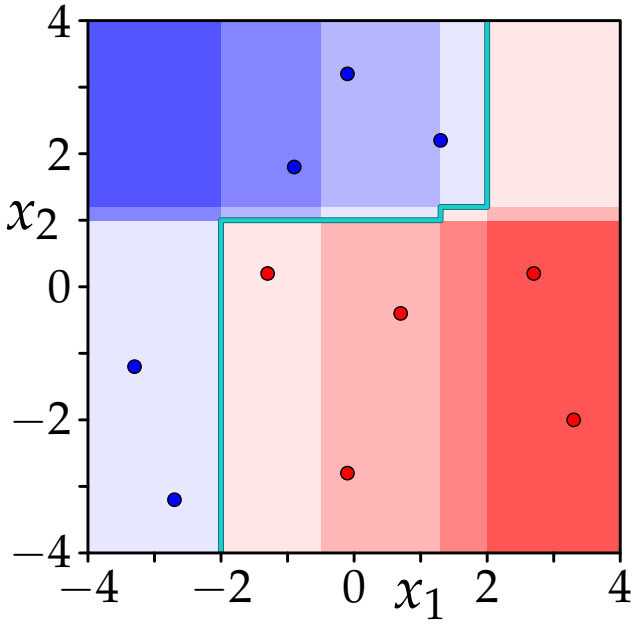
- Data point weights get increased along the separating line.
- Perfect classification is achieved with 7 decision stumps.

Ensemble Methods: Bagging

- **Bagging Example 1**
- Simple data set with two classes:
  - 5 data points belong to the **blue** class.
  - 5 data points belong to the **red** class.
- Classification with **decision stumps**.
  (but different model building than boosting)





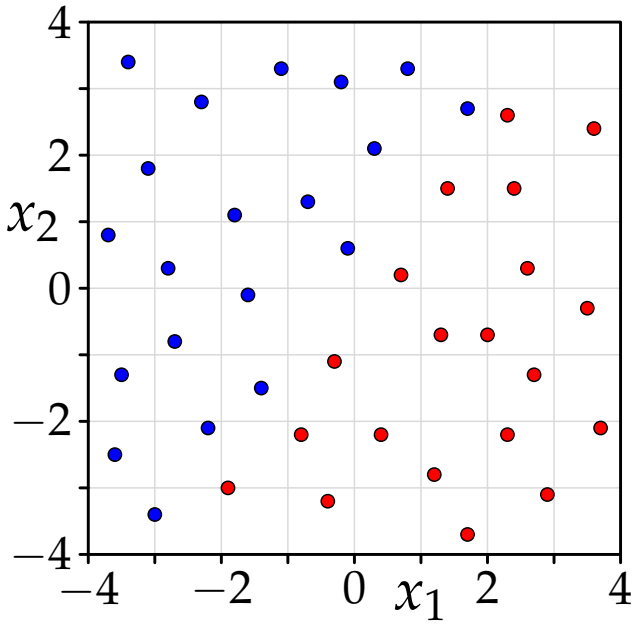


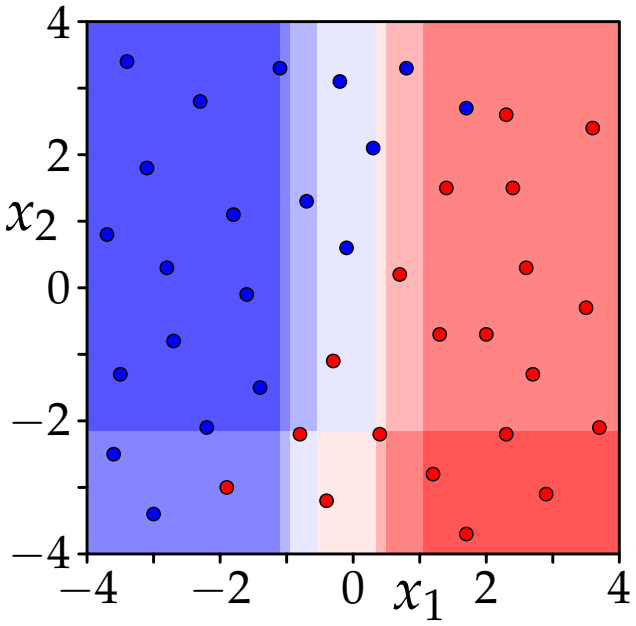
- Left: models from 5 bootstrap samples
- Right: models from 7 bootstrap samples

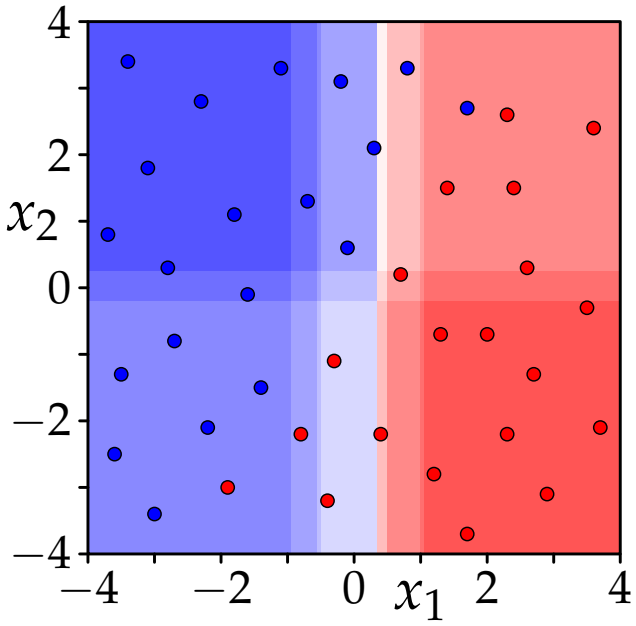
(different random seeds, hence also different bootstrap samples)

Ensemble Methods: Bagging

- **Bagging Example 2**
- Simple data set with two classes:
  - 20 data points belong to the **blue** class.
  - 20 data points belong to the **red** class.
- Classification with **decision stumps**.
  (bagging with decision stumps does not work well)





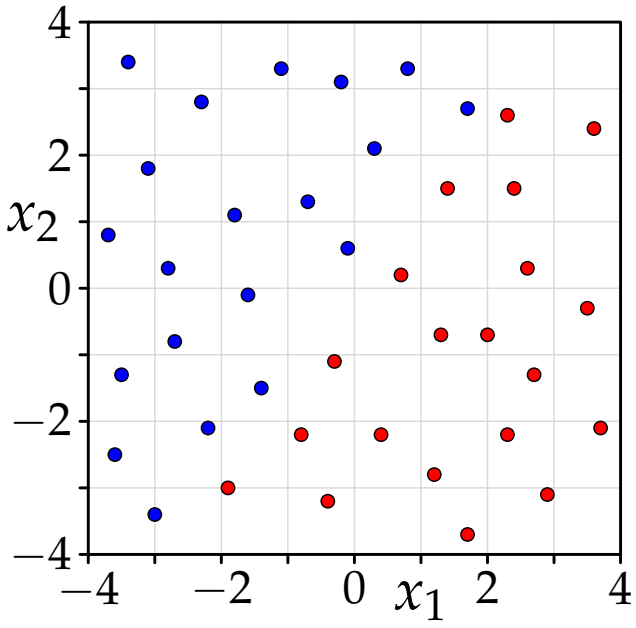


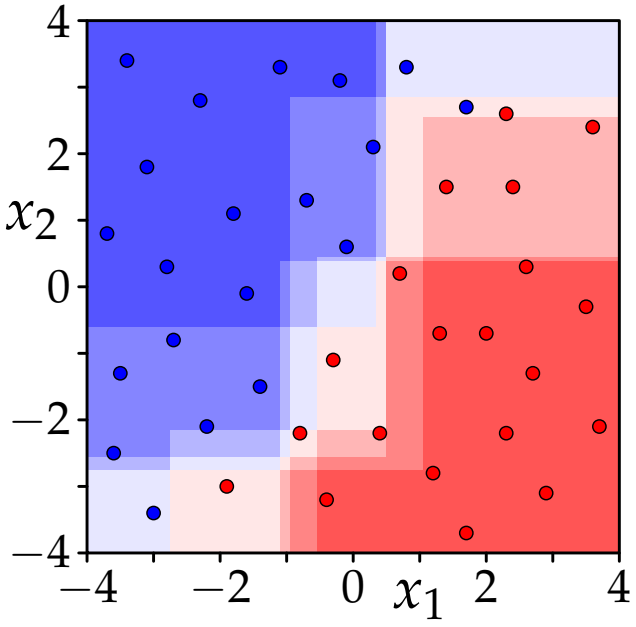
- Left: models from 7 bootstrap samples
- Right: models from 13 bootstrap samples

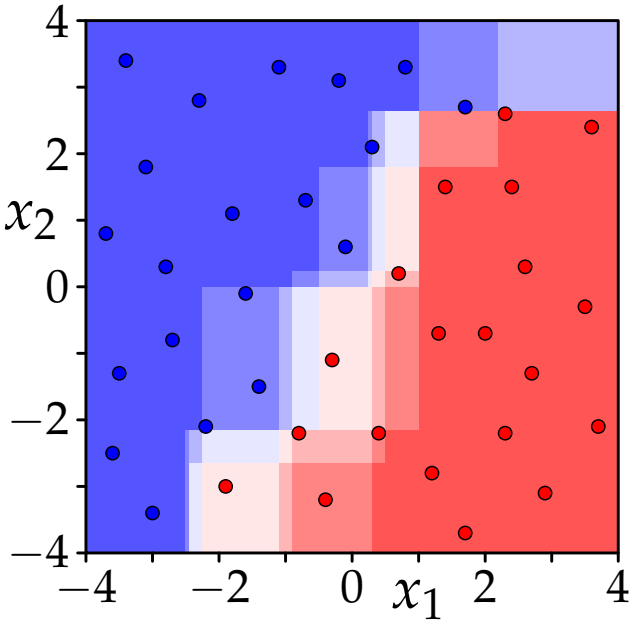
(different random seeds, hence also different bootstrap samples)

Ensemble Methods: Bagging

- **Bagging Example 2**
- Simple data set with two classes:
  - 20 data points belong to the **blue** class.
  - 20 data points belong to the **red** class.
- Classification with **decision trees**.
  (bagging with decision stumps does not work well)







- Left: tree depth 2,
  7 bootstrap samples
- Right: tree depth 3,
  7 bootstrap samples

(different random seeds,
hence also different
bootstrap samples)

Ensemble Methods: Mixture of Experts

- In the approach referred to as **mixture of experts** the individual predictors to combine are assumed as already given, for example, selected by a user.
- They may be, for instance, results of different learning algorithms (like decision tree, neural networks with different structure, support vector machine etc.), whatever a user sees as promising to solve the application task.
- Alternatively, they may be the set of models obtained from any of the ensemble methods described so far.
- The focus is then placed on finding an optimal rule to combine the predictions of the individual models.
- For classification tasks, for example, the input to this combination rule are the class probability distributions produced by the individual classifiers.
- Note that this requires more than simple (weighted) majority voting, which only asks each classifier for its best guess of the class of a new object: each classifier must assign a probability to each class.

Ensemble Methods: Mixture of Experts

- The most common rules to combine such class probabilities are
  - the so-called **sum rule**, which simply averages, for each class, the probabilities provided by the individual classifiers and
  - the so-called **product rule**, which assumes conditional independence of the classifiers given the class and therefore multiplies, for each class, the probabilities provided by the different classifiers.
- In both cases the class with the largest sum or product is chosen as the prediction of the ensemble.
- Experiments show that the sum rule is usually preferable, likely because due to the product a class that is seen as (very) unlikely by a single classifier has little chance of being predicted, even if several other classifiers assign a high probability to it.
- Both the sum rule and the product rule can be seen as special cases of a general family of combination rules that are known as f-means. Other such rules include Dempster–Shafer combination and rank based rules.

Ensemble Methods: Stacking

- Like a mixture of experts, **stacking** takes the set of predictors as already given and focuses on combining their individual predictions.
- The core idea is to view the outputs of the predictors as new features and to use a learning algorithm to find a model that combines them optimally.
- Technically, a new data table is set up with one row for each training example, the columns of which contain the predictions of the different (level-1) models for each training example.
  In addition, a final column states the true classes.
- With this new training data set a (level-2) model is learned, the output of which is the prediction of the ensemble.
- Note that the level-2 model may be of the same type as or of a different type than the level-1 models.
- For example, the output of several regression trees (e.g. a random forest) may be combined with a linear regression, or with a neural network.

Ensemble Methods: Measuring Diversity

- Reminder: A necessary and sufficient condition for an ensemble to out-perform the individuals is that the predictors are **reasonably accurate** and **diverse**.
- **Cohen's** κ (kappa) [Jacob Cohen 1960]
  Measures the agreement between two classifiers and is defined as

$$\kappa = \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e},$$

  where p_o is the observed relative agreement between the classifiers and p_e is the hypothetical relative agreement expected by chance.
- Perfect agreement between the classifiers yields $\kappa = 1$.
  Mere chance agreement between the classifiers yields $\kappa = 0$.
- Cohen's κ is computed from a **confusion matrix** of the two classifiers.
  (Note that this allows us to use the ground truth as one of the classifiers.)

  This is a contingency table with entries n_{ij} that count how many data points are assigned to class i by classifier 1 and to class j by classifier 2.

Ensemble Methods: Measuring Diversity

- Cohen's κ is computed from a confusion matrix of the two classifiers.

  (c is the number of classes.)

  (Note that one of the classifiers may be the ground truth classification, yielding a standard confusion matrix.)

		classifier 2				
		1	2	...	c	$\sum$
classifier 1	1	n_{11}	n_{12}	...	n_{1c}	$n_{1.}$
	2	n_{21}	n_{22}	...	n_{2c}	$n_{2.}$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
	c	n_{c1}	n_{c2}	...	n_{cc}	$n_{c.}$
	$\sum$	$n_{.1}$	$n_{.2}$	...	$n_{.c}$	$n_{..}$

- The two classifiers agree on the (sum of the) diagonal entries:

$$p_o = \frac{1}{n_{..}} \sum_{i=1}^{c} n_{ii}.$$

(This is the **accuracy** if one of the two classifiers is the ground truth.)

This is the **observed relative agreement** of the two classifiers.

- Note that p_o can **not** be used directly as a measure of agreement (or $1 - p_o$ as a measure of diversity), because unbalanced class distributions induce a tendency towards higher agreement (on the more frequent class(es)).

Ensemble Methods: Measuring Diversity

- In order to deal with unbalanced class distributions, we have to work out the **relative chance agreement** to obtain p_e.
- We assume that the classifiers assign classes **independently** and **randomly** (respecting the marginal probabilities as given by the marginal counts).
- That is, we assume that
  - the probability that classifier 1 assigns class i to a data point is $\hat{p}_{i.} = {}^{n_{i.}}/_{n_{..}}$
  - the probability that classifier 2 assigns class j to a data point is $\hat{p}_{.j} = {}^{n_{.j}}/_{n_{..}}$
  - the assignments of the two classifiers are stochastically independent.
- Under these assumptions, we expect the number of data points that are assigned by classifier 1 to class i and by classifier 2 to class j to be

$$\hat{n}_{ij} = n_{..}\hat{p}_{i.}\hat{p}_{.j} = n_{..} \cdot \frac{n_{i.}}{n_{..}} \cdot \frac{n_{.j}}{n_{..}} = \frac{n_{i.}n_{.j}}{n_{..}}.$$

and therefore

$$p_e = \frac{1}{n_{..}} \sum_{i=1}^{c} \hat{n}_{ii} = \frac{1}{n_{..}} \sum_{i=1}^{c} \frac{n_{i.}n_{.i}}{n_{..}} = \frac{1}{n_{..}^2} \sum_{i=1}^{c} n_{i.}n_{.i}.$$

Ensemble Methods: Measuring Diversity

- Cohen's κ is defined as
$$\kappa = \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e}.$$
- Two classifiers that are in perfect agreement have $p_o = 1$.
- Hence Cohen's κ measures how different the agreement of two classifiers is from perfect agreement if the expected chance agreement is deducted. That is, only the excess over chance agreement is considered.
- If the ground truth is used for one of the classifiers, Cohen's κ measures by how much a given classifier is better than a random classifier.
- Even though the most relevant values of Cohen's κ are $\kappa = 1$ (perfect agreement) and $\kappa = 0$ (only chance agreement), it is not limited to the interval $[0, 1]$, but can actually be negative.
- If the two classifiers disagree fully (always predict different classes), it is $p_o = 0$ and hence $\kappa = \frac{p_e}{p_e - 1}$ (for example, for $p_e = \frac{1}{2}$, it is $\kappa = -1$).

Ensemble Methods: Measuring Diversity

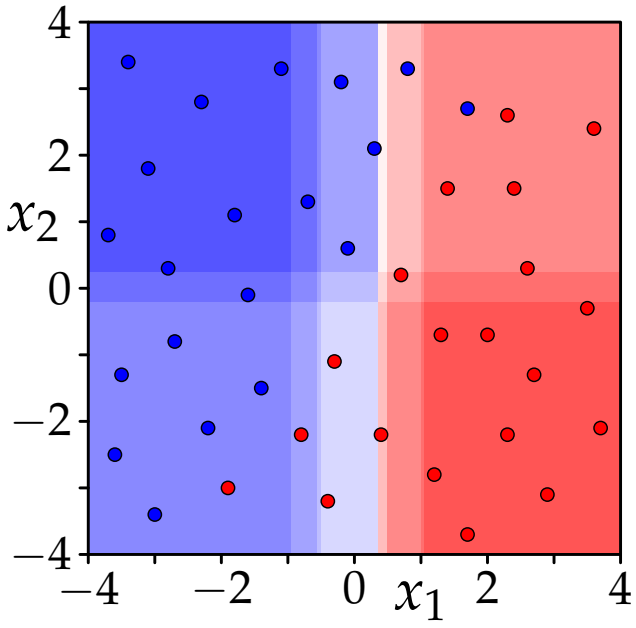
- With Cohen's κ the classifier properties of *reasonably accurate* and *diverse* can be explored with **κ-error diagrams**. [Margineantu & Dietterich 1997]
  - Scatter plot with one point for each pair of classifiers.
  - On the x-axis: Cohen's κ for the pair of classifiers.
  - On the y-axis: average of the error rates of the two classifiers.
- Note that the requirement that the classifiers should be *reasonably accurate (better than random guessing)* does not prevent Cohen's κ from being negative.
- Suppose that for a binary classification problem we have $(4k + 2)$ triplets $(\textit{prediction classifier 1}, \textit{prediction classifier 2}, \textit{true class})$ consisting of

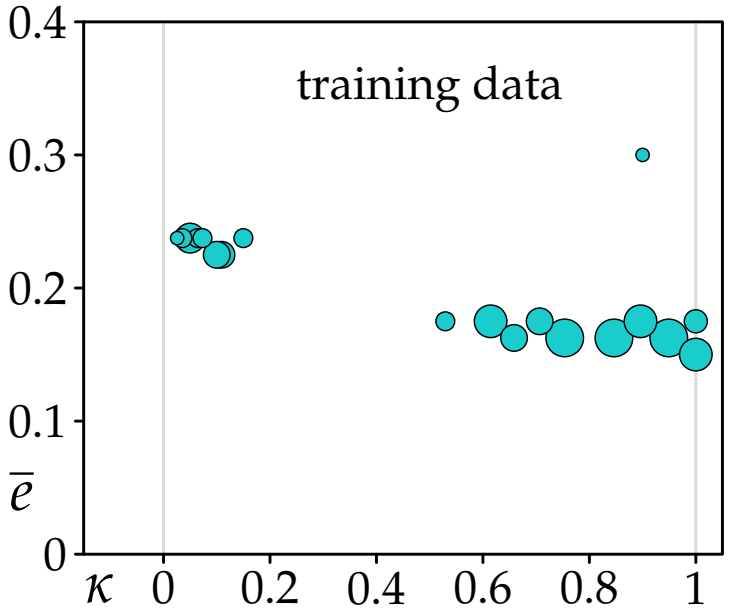
$$\begin{array}{lll} k \times (0,1,0), & k \times (1,0,0), & 1 \times (0,0,0), \\ k \times (0,1,1), & k \times (1,0,1), & 1 \times (1,1,1). \end{array}$$

- The probabilities of the classes as well as of the predictions are $p_0 = p_1 = \frac{1}{2}$.
- For $k = 24$, the classifier accuracies are $\text{acc}_1 = \text{acc}_2 \approx 0.51 > \frac{1}{2}$ and $\kappa \approx -0.96$.

Ensemble Methods: Measuring Diversity

- **Bagging Example**
- Training data set with two classes:
  - 20 data points belong to the **blue** class.
  - 20 data points belong to the **red** class.
- Classification with **decision stumps**.
  (bagging with decision stumps does not work well)



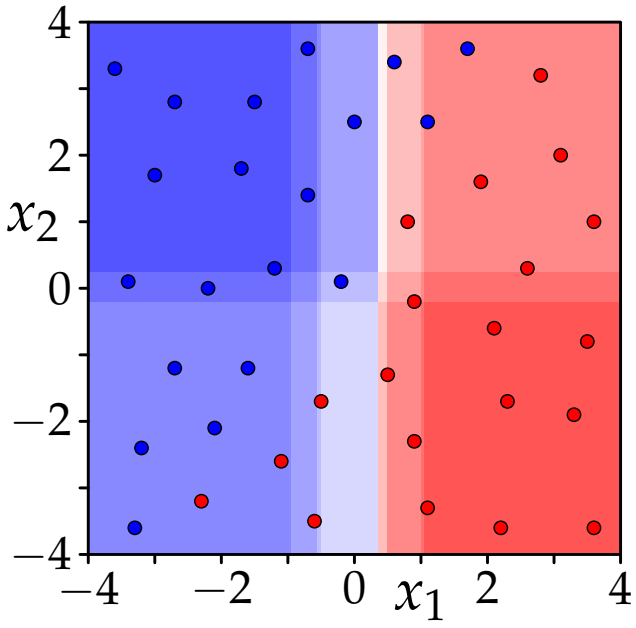


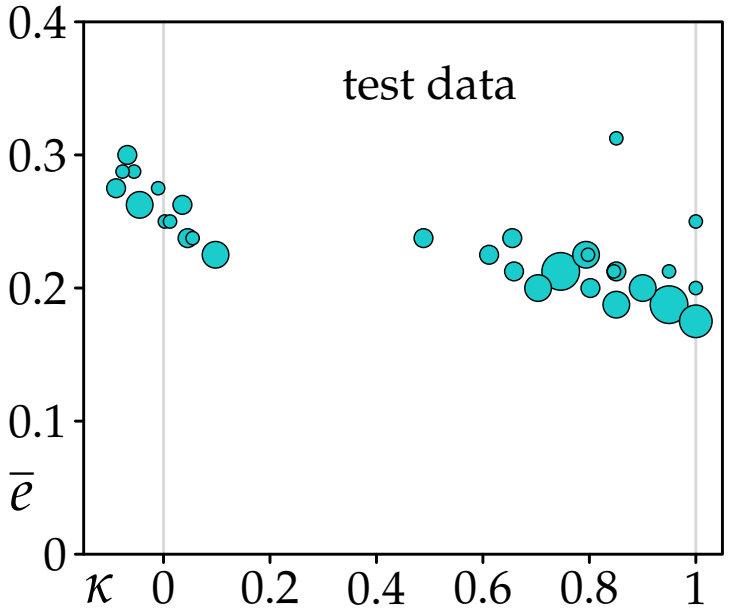
- 13 bootstrap samples
  $\Rightarrow$ 13 classifiers in principle,
  but some are identical.
- Actually 9 different classifiers,
  therefore $\binom{9}{2} = 36$ classifier pairs,
  but only 19 different κ-error pairs.

  (diagram: dot area is proportional to multiplicity)

Ensemble Methods: Measuring Diversity

- **Bagging Example**
- Test data set with two classes:
  - 20 data points belong to the **blue** class.
  - 20 data points belong to the **red** class.
- Classification with **decision stumps**.
  (bagging with decision stumps does not work well)





- 13 bootstrap samples
  $\Rightarrow$ 13 classifiers in principle,
  but some are identical.
- Actually 9 different classifiers,
  therefore $\binom{9}{2} = 36$ classifier pairs,
  but only 31 different κ-error pairs.

  (diagram: dot area is proportional to multiplicity)

Ensemble Methods: Summary

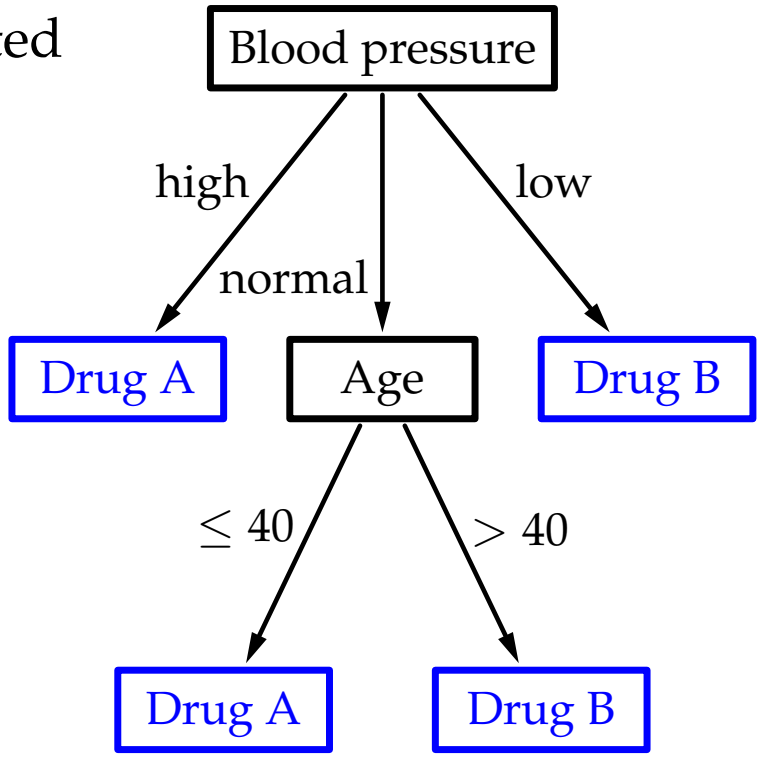
- **Fundamental Ideas**
  - Combine Multiple Classifiers/Predictors
  - Why Do Ensemble Methods Work?
- **Ingredients: Predictor Construction + Combination Rule**
- **Popular Ensemble Methods**
  - Bayesian Voting / Averaging (theoretically optimal)
  - Bagging (Bootstrap Aggregating) (predictor construction)
  - Random Subspace Selection (predictor construction)
  - Injecting Randomness (predictor construction)
  - Boosting, especially AdaBoost (predictor construction & combination rule)
  - Mixture of Experts (combination rule)
  - Stacking (combination rule)

Random Forests

(Forest of Decision Trees induced with an Injection of Randomness)

Reminder: Decision Trees

- A **classifier** is an algorithm that assigns a class from a predefined set to a case or object, based on the values of descriptive attributes.
- A **decision tree** is a classifier with tree structure, in which the classification process starts at the root node and proceeds down the branches until a leaf is reached.
  - In inner nodes, descriptive attributes are tested and the case to classify is directed down the branch corresponding to the test outcome.
  - When finally a leaf node is reached, the leaf node class is assigned to the case.
- **Decision tree induction** is characterized by:
  - Top-down approach (from root to leaves).
  - Greedy selection of test attributes / tests.
  - Divide-and-conquer / recursive descent.



Decision Trees: Advantages and Disadvantages

- **Decision Trees** are attractive, because they have various **advantages**:
  - fairly intuitive idea, interpretable model, easy to check for plausibility
  - invariant under scaling and various other transformations of feature values
  - induction is often straightforward (e.g. TDIDT) (top-down induction of decision trees)
  - once a decision tree is built, classification is very fast
- However, decision trees also have **disadvantages**:
  - They tend to overfit the training data, at least unless
    - a minimum number of cases per leaf is required or
    - induced decision trees are pruned.

    Both approaches limit the accuracy on the training data.
  - The number of attributes that can be used for a decision is limited (the path length in a tree and thus the number of tests is limited)

    $\Rightarrow$ limited "model capacity" / "model expressivity"

Random Forests: A Brief History 1

- **Random Decision Forests** [Ho 1995]
  Build multiple decision trees in **randomly selected subspaces**.
  - Idea: Decision trees induced for different feature subsets generalize in different ways (they are invariant for unselected features).
  - Choosing random subsets is a convenient way of exploring feature subsets.
  - For each (randomly) chosen feature subset a decision tree T is induced.
  - The class assigned by a random decision forest $\mathcal{T}$ to a data point $\vec{x}$ is

$$\mathcal{T}(\vec{x}) = \underset{c \in \mathrm{dom}(C)}{\operatorname{argmax}} \hat{P}_{\mathcal{T}}(c \mid \vec{x}), \qquad \text{(choose most likely class)}$$

  where C is the target (i.e. class) attribute and

$$\hat{P}_{\mathcal{T}}(c \mid \vec{x}) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} \hat{P}_T(c \mid \vec{x}) \qquad \text{(effectively: sum rule)}$$

  with $\hat{P}_T(c \mid \vec{x})$ the confidence with which tree T assigns class c to $\vec{x}$ (that is, the relative frequency of class c in the leaf node reached by $\vec{x}$ in T).

Random Forests: A Brief History 2

- **Bagging Predictors** [Breiman 1996]
  Build multiple decision trees from **bootstrap samples / bootstrap replicates**.
  - Idea: "Unstable" learning methods (like decision tree induction) produce diverse predictors from bootstrap replicates.
  - Reminder: **bootstrap samples** are random samples of the same size as the given data set that are drawn *with replacement*.
  - The class assigned by a bootstrapped decision forest $\mathcal{T}$ to a data point $\vec{x}$ is

$$\mathcal{T}(\vec{x}) = \underset{c \in \mathrm{dom}(C)}{\operatorname{argmax}} \sum_{T \in \mathcal{T}} \mathbb{I}(T(\vec{x}) = c) \qquad \text{(majority voting)}$$

  where C is the target (i.e. class) attribute and

$$T(\vec{x}) = \underset{c \in \mathrm{dom}(C)}{\operatorname{argmax}} \hat{P}_T(c \mid \vec{x}) \qquad \text{(most likely class)}$$

  with $\hat{P}_T(c \mid \vec{x})$ the confidence with which tree T assigns class c to $\vec{x}$ (that is, the relative frequency of class c in the leaf node reached by $\vec{x}$ in T).

Random Forests: A Brief History 3

- **Random Decision Forests** [Ho 1998]

  Build multiple decision trees in **randomly selected subspaces**.

  Extensions of random decision forests mentioned in [Ho 1998]:

  - Random feature subspaces may be selected globally or whenever an internal tree node is generated (**"local" subspace selection**).

    Note: The latter does not limit the features a decision tree as a whole can use.

  - An alternative to build multiple decision trees is cross-validation: Each cross-validation split yields one decision tree of a decision forest.

    $\Rightarrow$ **cross-validated committees**

  - **Bagging (bootstrap aggregating)** and **boosting** may also be used to obtain multiple decision trees.

    Note: Boosting also weights the influence of the individual decision trees (different prediction rule).

Side remarks: [Breiman 1996] found no advantage of random subspace selection over bagging when using a majority vote of the individual trees or the sum rule for result combination.
[Ho 1998] did find an advantage when using the sum rule to combine tree predictions.
[Ho 1998] found trees to be more diverse when obtained with random subspace selection, using a measure of classification region overlap of different trees.

Random Forests: A Brief History 4

- **Ensembles of Decision Trees** [Dietterich 2000]

  Build multiple decision trees using three methods:
  Bagging, **Boosting** or **Random Split Selection** (as alternatives).

  - For *random split selection* the split is chosen uniformly at random from the best k splits (e.g. $k = 20$).
  - For numeric attributes each possible threshold yields a separate split.
    (Each threshold θ gives rise to a binary pseudo-attribute with values $\leq \theta$ and $> \theta$.)
  - Alternative: Make the probability of choosing a split proportional to the information gain of that split.
    ($\Rightarrow$ better splits are chosen with higher probability)
  - Refinement: At most $s_{\max}$ splits are randomized within a tree
    (for some user-specified value of $s_{\max}$).

    Explicitly consider $s = 0, 1, \ldots, s_{\max}$ random splits (instead of a random s).

    This would ensure that the "best" tree is included in the ensemble.
    (That is, the decison tree that is induced without random split selection.)

Random Forests: A Brief History 5

- **Random Forests** [Breiman 2001]

  Build multiple decision trees from **bootstrap samples / bootstrap replicates** and/or using **randomly chosen feature subsets** for each split.

  - Definition: A **random forest (classifier)** consists of a collection $\mathcal{T} = \{T_k\}_{k=1,2,\ldots}$ of tree-structured classifiers $T_k \colon \mathcal{X} \to \mathrm{dom}(C)$, each induced via $T_k = \mathcal{F}_{\mathrm{TDIDT}}(\mathbf{D}, \Theta_k)$, where $\mathbf{D}$ is the data and the Θ_k are independent and identically distributed random vectors. Each tree T_k casts a unit vote for a class $c \in \mathrm{dom}(C)$ at an input $\vec{x} \in \mathcal{X}$.

    (C is the class attribute)

  - In *bagging* the random vectors Θ_k describe the bootstrap samples.

    (e.g., by a collection of independent random integers from $\{1, \ldots, n\}$ where n is the number of data points.)

  - In *random subspace selection* or *random split selection* the random vectors Θ_k describe the feature or split selection.

    (e.g., by a collection of independent random integers from $\{1, \ldots, m\}$ where m is the number of features.)

  - Since *bagging* may be combined with *random subspace* or *random split selection*, the random vectors Θ_k may also describe both aspects.

    (in this case Θ_k consists of two separate parts, that is, $\Theta_k = \Theta_{k,\mathrm{bag}} \cup \Theta_{k,\mathrm{rss}}$)

Random Forests: A Brief History 6

- **Extremely Randomized Trees (ExtRaTrees)** [Geurts, Ernst & Wehenkel 2006]
  Build multiple decision trees from the **whole data set** (no bagging/sampling) using **Random Split Selection** for a **subset of the attributes**.
  - For each attribute, choose a random split point uniformly at random between the minimum and the maximum attribute value.
    (This implictly assumes metric attribute values, but nominal attributes can be handled analogously: split the set of occurring nominal attribute values randomly into two non-empty subsets.)
  - Evaluate the randomly chosen candidate splits (one per attribute) and select the split/attribute with the best evaluation.
  - Difference to ensemble approach by [Dietterich 2000] (brief history 4):
    - [Dietterich 2000] evaluates *all splits* and then chooses randomly from the k best splits (e.g. $k = 20$). $\Rightarrow$ focus more on tree accuracy
    - In ExtRaTrees only *one randomly chosen split per attribute* is considered, from which the best split/attribute is selected.
    - Thus, ExtRaTrees are much more random ("*extremely random*") and hence (individually) less accurate. $\Rightarrow$ focus more on tree diversity

Random Forests: Regression Trees

- Although originally introduced for classification, random forests can just as well be used for regression:
  - Definition: A **random forest regressor** consists of a collection $\mathcal{T} = \{T_k\}_{k=1,2,\ldots}$ of tree-structured regressors $T_k : \mathcal{X} \to \mathbb{R}$, each induced via $T_k = \mathcal{F}_{\text{TDIRT}}(\mathbf{D}, \Theta_k)$, where $\mathbf{D}$ is the data and the Θ_k are independent and identically distributed random vectors. Each tree T_k produces a prediction $T_k(\vec{x}) = \hat{y} \in \mathbb{R}$ at an input $\vec{x} \in \mathcal{X}$; these predictions are simply averaged to obtain an overall prediction.
  - As for random forest classifiers, the random vectors Θ_k describe *bagging* and/or *random subspace* or *random split selection* in the induction.
- From an ensemble point of view, there is no fundamental difference between **random forest classifiers** and **random forest regressors**:
  - Both use tree-structured base models / base learners.
  - Both use the same methods to inject randomness into the induction process (*bagging*, *random subspace selection*, and *random split selection*).

Random Forests: Subspace / Split Selection

- Standard Random Forests (as implemented in many libraries) use both **bagging** and **random subspace** or **random split selection**.
- **Bagging** is conducted in the standard fashion: (not used in ExtRaTrees)
  Draw bootstrap samples and induce a separate tree for each sample.
- Typically, random forests use **(local) random subspace selection**, controlled by a parameter $m_{\text{rss}} \leq m$ for the maximum size of the subspace.
  (Here m is the total number of attributes/features.)
- Typical suggestions for the choice of m_{rss} are: (generally target-/data-dependent)

$$m_{\text{rss}} = \lfloor m/3 \rfloor \qquad \text{or} \qquad m_{\text{rss}} = \lfloor \sqrt{m} \rfloor \qquad \text{or} \qquad m_{\text{rss}} = \lfloor \log_2(m[+1]) \rfloor.$$

- In **Extremely Randomized Trees**, all available attributes may be used or they may be constrained to a subset (m_{rss} chosen as stated above).

  A subset increases the randomization even more (*"extremely randomized"*).

  Choosing $m_{\text{rss}} = 1$ yields **totally randomized trees** (w.r.t. chosen splits).

Random Forests: Bias

- In the limit $|\mathcal{T}| \to \infty$, a random forest of regression trees predicts at point $\vec{x}$:

$$\widehat{\mathcal{T}}_\infty(\vec{x}) \;=\; \lim_{|\mathcal{T}|\to\infty} \frac{1}{|\mathcal{T}|} \sum_{T\in\mathcal{T}} T(\vec{x}) \;=\; \mathbb{E}_{\mathbf{D},\Theta}(T_{\mathbf{D},\Theta}(\vec{x})),$$

  where $T_{\mathbf{D},\Theta} = \mathcal{F}_{\text{TDIRT}}(\mathbf{D},\Theta)$. ($\mathbf{D}$: data, Θ: random vector)

- The bias of a random forest of regression trees is the same as the bias of any of the individual sampled trees $T_{\mathbf{D},\Theta}(\vec{x})$: [Hastie *et al.* 2009]

$$\text{bias}(\vec{x}) \;=\; \mu_f(\vec{x}) - \widehat{\mathcal{T}}_\infty(\vec{x}) \;=\; \mu_f(\vec{x}) - \mathbb{E}_{\mathbf{D},\Theta}(T_{\mathbf{D},\Theta}(\vec{x})),$$

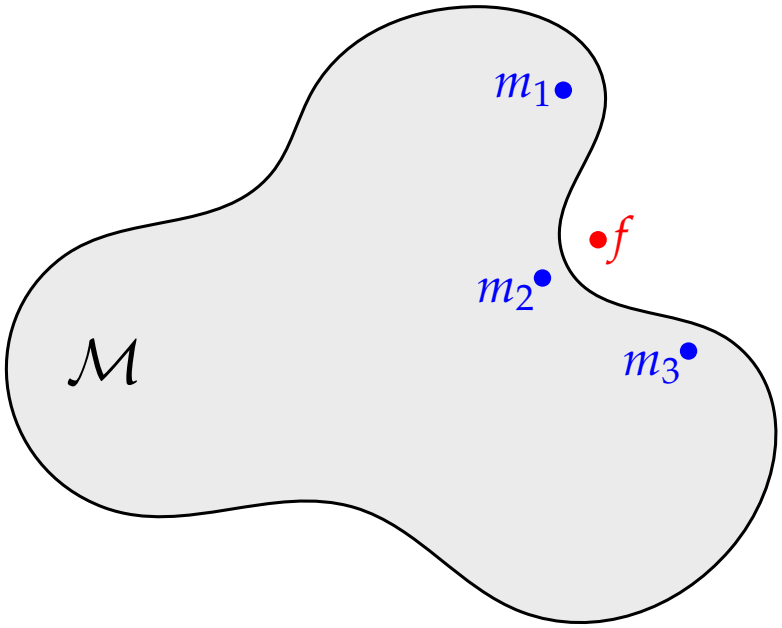
  where f describes the true underlying dependence of the target variable on $\vec{x}$; the expected value $\mu_f(\vec{x})$ takes noise into account, that is, $y = f(\vec{x}) + \varepsilon$.

- Typically, the absolute value of this **bias is greater** then the bias of an (unpruned) single regression tree induced from the data $\mathbf{D}$, because randomization introduces restrictions into the induction process.
  (restricted set of features for the splits, only a subset (bootstrap sample) of the data is considered)

- The improvements in prediction obtained by random forests are **solely a result of variance reduction**.

Reminder: Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are essentially three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.
- **Representational Reason**



- The gray area $\mathcal{M}$ is the space of all attainable models (or *hypotheses*).
  ("model class" in lecture "Elementary Data Mining")
- m_1, m_2 and m_3 are individual models, e.g. the individual models of an ensemble.
- f is the best model, the true hypothesis. However, f lies outside of $\mathcal{H}$.
- The average prediction of the models m_1, m_2 and m_3 is close(r) to the prediction of the true hypothesis f.

  $\Rightarrow$ Ensemble methods can enrich the space of representable hypotheses.

Random Forests: Bias

- The **representational reason** why ensemble methods work states that by combining several models in an ensemble, the model space can be enriched.
- An enriched model space has a **lower bias**, because additional dependences between the inputs and the target variable can be represented.
- **Why do random forests have the same bias as the individual trees?**
- The reason is that the trees in a random forest are usually
  - grown without any lower limit for the number of cases per leaf,
    (if necessary, the splits are continued down to a single case per leaf)
  - not pruned.
- Therefore the trees of a random forest usually achieve **perfect prediction** on the data they are induced on, unless this data contains contradictions.
  (contradiction: two sample cases that differ only in their target values)
- Actually, the **bias is increased** by the randomization.
  (more contradictions on average due to the subspace selection: distinction may need an unselected feature)

Excursion: Variance of Mean of (Correlated) Random Variables

- Consider n i.i.d. random variables X_i, $i = 1, \ldots, n$, with $\text{var}(X_i) = \sigma^2$.
  The variance of the mean of these variables is

$$\text{var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n}\sigma^2.$$

- If the random variables are i.d. (identically distributed, but not necessarily independent) with pairwise correlation ρ, the variance of their mean is

$$\text{var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \rho\sigma^2 + (1-\rho)\frac{1}{n}\sigma^2.$$

- This can be derived as follows: (covariance is bi-linear)

$$\begin{aligned}
\text{var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) &= \text{cov}\left(\frac{1}{n}\sum_{i=1}^{n} X_i,\ \frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\text{cov}(X_i, X_j) \\
&= \frac{1}{n^2}\left(\sum_{i=1}^{n}\text{var}(X_i) + \sum_{i=1}^{n}\sum_{j=1;j\neq i}^{n}\text{cov}(X_i, X_j)\right) \\
&= \frac{1}{n^2}(n\sigma^2 + n(n-1)\rho\sigma^2) \\
&= \frac{1}{n}\sigma^2 + \frac{n-1}{n}\rho\sigma^2 = \frac{1}{n}\sigma^2 + \rho\sigma^2 - \frac{1}{n}\rho\sigma^2 \\
&= \rho\sigma^2 + (1-\rho)\frac{1}{n}\sigma^2
\end{aligned}$$

Random Forests: Variance and De-Correlation

- In the limit $|\mathcal{T}| \to \infty$, a random forest of regression trees predicts at point $\vec{x}$:

$$\widehat{\mathcal{T}}_\infty(\vec{x}) \;=\; \lim_{|\mathcal{T}|\to\infty} \tfrac{1}{|\mathcal{T}|} \sum_{T\in\mathcal{T}} T(\vec{x}) \;=\; \mathbb{E}_{\mathbf{D},\Theta}(T_{\mathbf{D},\Theta}(\vec{x})),$$

  where $T_{\mathbf{D},\Theta} = \mathcal{F}_{\text{TDIRT}}(\mathbf{D},\Theta)$. ($\mathbf{D}$: data, Θ: random vector)

- The variance of this limit prediction is [Hastie *et al.* 2009]

$$\text{var}_{\mathbf{D},\Theta}(\widehat{\mathcal{T}}_\infty(\vec{x})) = \rho_{\mathbf{D},\Theta}(\vec{x})\, \sigma^2_{\mathbf{D},\Theta}(\vec{x}),$$

  where $\rho_{\mathbf{D},\Theta}(\vec{x})$ is the sampling correlation between pairs of trees, that is,

$$\rho_{\mathbf{D},\Theta}(\vec{x}) = \text{corr}_{\mathbf{D},\Theta}(T_{\mathbf{D},\Theta_1}(\vec{x}), T_{\mathbf{D},\Theta_2}(\vec{x})),$$

  where $T_{\mathbf{D},\Theta_i} = \mathcal{F}_{\text{TDIRT}}(\mathbf{D},\Theta_i)$, $i = 1,2$, and (Θ_i: i.i.d. random vectors)
  $\sigma^2_{\mathbf{D},\Theta}(\vec{x})$ is the sampling variance of single trees, that is,

$$\sigma^2_{\mathbf{D},\Theta}(\vec{x}) = \text{var}_{\mathbf{D},\Theta}(T_{\mathbf{D},\Theta}(\vec{x})).$$

- Note that in this description both the data $\mathbf{D}$ and Θ are random variables.

Excursion: Law of Total Expectation

- If X and Y are two random variables on the same probability space, and the expected value of Y is defined, then it can be written as

$$\mathbb{E}_X(X) = \mathbb{E}_Y(\mathbb{E}_{X|Y}(X)),$$

that is, the expected value of the conditional expected value of X given Y is the same as the expected value of X.

- In the continuous case, this law can be demonstrated as follows:

$$\begin{aligned}
\mathbb{E}_{X|Y=y}(X) &= \int_x xP(X=x|Y=y)\,dx \\
\mathbb{E}_Y(\mathbb{E}_{X|Y}(X)) &= \int_y \big(\int_x xP(X=x|Y=y)\,dx\big)P(Y=y)\,dy \\
&= \int_y\int_x xP(X=x,Y=y)\,dx\,dy \\
&= \int_y \big(\int_x xP(X=x|Y=y)\,dx\big)P(Y=y)\,dy \\
&= \int_x x\big(\int_y P(X=x,Y=y)\,dy\big)dx \\
&= \int_x xP(X=x)\,dx \;=\; \mathbb{E}_X(X)
\end{aligned}$$

Excursion: Law of Total Variance

- If X and Y are two random variables on the same probability space, and the variance of X is finite, then it can be decomposed as

$$\text{var}_X(X) = \mathbb{E}_Y(\text{var}_{X|Y}(X)) + \text{var}_Y(\mathbb{E}_{X|Y}(X)).$$

- This can be derived as follows:

$$\begin{aligned}
\text{var}_X(X) &\overset{(1)}{=} \mathbb{E}_X(X^2) - (\mathbb{E}_X(X))^2 \\
&\overset{(2)}{=} \mathbb{E}_Y(\mathbb{E}_{X|Y}(X^2)) - (\mathbb{E}_Y(\mathbb{E}_{X|Y}(X)))^2 \\
&\overset{(3)}{=} \mathbb{E}_Y(\text{var}_{X|Y}(X) + (\mathbb{E}_{X|Y}(X))^2) - (\mathbb{E}_Y(\mathbb{E}_{X|Y}(X)))^2 \\
&\overset{(4)}{=} \mathbb{E}_Y(\text{var}_{X|Y}(X)) + \mathbb{E}_Y((\mathbb{E}_{X|Y}(X))^2 - (\mathbb{E}_Y(\mathbb{E}_{X|Y}(X)))^2 \\
&\overset{(5)}{=} \mathbb{E}_Y(\text{var}_{X|Y}(X)) + \text{var}_Y(\mathbb{E}_{X|Y}(X))
\end{aligned}$$

(1) definition of variance: $\text{var}_X(X) = \mathbb{E}_X((X - E_X(X))^2) = \mathbb{E}_X(X^2) - (\mathbb{E}_X(X))^2$
(2) law of total expectation (2×: once for $\mathbb{E}_X(X^2)$ and once for $\mathbb{E}_X(X)$)
(3) definition of variance: $\text{var}_{X|Y}(X) = \mathbb{E}_{X|Y}(X^2) - (\mathbb{E}_{X|Y}(X))^2$
(4) expected value is linear (expectation of sum is sum of expectations)
(5) definition of variance: $\text{var}_Y(E_{X|Y}(X)) = \mathbb{E}_Y((E_{X|Y}(X))^2) - (\mathbb{E}_Y(E_{X|Y}(X)))^2$

Random Forests: Variance and De-Correlation

- The total variance $\mathrm{var}_{\mathbf{D},\Theta}(T_{\mathbf{D},\Theta}(\vec{x}))$ can be decomposed: [Hastie *et al.* 2009]

$$\underbrace{\mathrm{var}_{\mathbf{D},\Theta}(T_{\mathbf{D},\Theta}(\vec{x}))}_{\text{total variance}} = \underbrace{\mathrm{var}_{\mathbf{D}}(\mathbb{E}_{\Theta|\mathbf{D}}(T_{\mathbf{D},\Theta}(\vec{x}))}_{\text{data variance}} + \underbrace{\mathbb{E}_{\mathbf{D}}(\mathrm{var}_{\Theta|\mathbf{D}}(T_{\mathbf{D},\Theta}(\vec{x}))}_{\text{randomization variance}}.$$

- The randomization variance increases, for example,
  if the number m_{rss} of features considered for a split is reduced.
- However, the data variance decreases with more randomization,
  and it usually decreases more than the randomization variance increases.
- Suppose there was no randomization. (no bagging, no random subspace/split selection)
  - All trees are the same (deterministic induction procedure).
  - If the trees are unpruned and not restricted during induction,
    they vary considerably with the data (overfitting to the data).
- With randomization, averaging multiple different trees
  reduces the overfitting of the individual trees to the training data.

Random Forests: Convergence

- For fixed data $\mathbf{D}$, the conditional correlation at point $\vec{x}$, that is, [Hastie *et al.* 2009]

$$\rho_{\Theta|\mathbf{D}}(\vec{x}) = \mathrm{corr}_{\Theta_1,\Theta_2}(T_{\Theta_1|\mathbf{D}}(\vec{x}), T_{\Theta_2|\mathbf{D}}(\vec{x})),$$

  is actually zero, because bootstrap and feature sampling are i.i.d.

- For fixed data $\mathbf{D}$, the variance of the limit prediction is

$$\mathrm{var}_{\Theta|\mathbf{D}}(\widehat{\mathcal{T}}_\infty(\vec{x})) = \rho_{\Theta|\mathbf{D}}(\vec{x})\, \sigma^2_{\Theta|\mathbf{D}}(\vec{x}),$$

  which vanishes due to the vanishing correlation $\rho_{\Theta|\mathbf{D}}(\vec{x})$.

- This means that a **random forest converges** for $|\mathcal{T}| \to \infty$.

- This does **not** mean that a random forest converges to perfect predictions, but only that it **converges to a certain generalization error**.

- It also does **not** mean that the concrete trees of a concrete random forest are uncorrelated. Rather, it is a statement about the sampling distribution.

  (that is, considering all possible Θ and their probability of occurrence)

Random Forests: Convergence

- If more and more trees are added to a random forest, its performance **converges to a certain generalization error** (i.e., error on new data).
- The value of this generalization error depends on
  - the **strength** of the individual trees and
  - the **correlation** between the individual trees. (attention: different from slide 91!)

  (More details for decision forests below and in [Breiman 2001] and [Hastie *et al.* 2009].)
- This guaranteed convergence to a certain generalization error is often interpreted as "*Random forests cannot overfit.*"
- However, this is misleading, since the **convergence limit may be overfitting**.
- The reason is that the data sets from which the individual trees are built are **not** independent random samples from the underlying distribution, but bootstrap samples from a specific given data set (data sample).
- A random forest may still overfit to the pecularities of this specific data set.

Random Forests: Generalization Error

- The **margin function** of a random decision forest $\mathcal{T}$ is [Breiman 2001]

$$\gamma_{\mathcal{T}}(\vec{x}, y) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} \mathbb{1}(T(\vec{x}) = y) \; - \max_{c \in \text{dom}(C); c \neq y} \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} \mathbb{1}(T(\vec{x}) = c).$$

  This function is positive where the random forest predicts the correct class and negative where it predicts an incorrect class.

- The larger the margin, the greater the confidence in the classification.

- The **generalization error** of a random decision forest $\mathcal{T}$ is

$$E_{\mathcal{T}} \; = \int_{\mathcal{X}} \sum_{y \in \text{dom}(C)} \mathbb{1}(\gamma_{\mathcal{T}}(\vec{x}, y) < 0) \, p(\vec{x}, y) \, d\vec{x} \;\; = \;\; P_{\vec{X}, Y}(\gamma_{\mathcal{T}}(\vec{X}, Y) < 0),$$

  that is, the total probability mass of data points that are misclassified.

- If more and more trees are added, the **margin function converges** to

$$\begin{aligned} \gamma_{\mathbf{D}}(\vec{x}, y) \;\; &= \;\; \mathbb{E}_{\Theta}(\mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{x}) = y)) - \max_{c \in \text{dom}(C); c \neq y} \mathbb{E}_{\Theta}(\mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{x}) = c)) \\ &= \;\; P_{\Theta}(\quad T_{\Theta|\mathbf{D}}(\vec{x}) = y \;\;) - \max_{c \in \text{dom}(C); c \neq y} P_{\Theta}(\quad T_{\Theta|\mathbf{D}}(\vec{x}) = c \;\;). \end{aligned}$$

Random Forests: Generalization Error

- If more and more trees are added, the **margin function converges** to

$$\gamma_{\mathbf{D}}(\vec{x}, y) = P_\Theta(T_{\Theta|\mathbf{D}}(\vec{x}) = y) - \max_{c \in \mathrm{dom}(C); c \neq y} P_\Theta(T_{\Theta|\mathbf{D}}(\vec{x}) = c).$$

- As a consequence, the **generalization error converges** to

$$E_{\mathbf{D}} = \int_{\mathcal{X}} \sum_{y \in \mathrm{dom}(C)} \mathbb{1}(\gamma_{\mathbf{D}}(\vec{x}, y) < 0)\, p(\vec{x}, y)\, d\vec{x} = P_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X}, Y) < 0),$$

  that is, the total probability mass of data points that are misclassified.

- The **strength** of a random decision forest (in the limit) is [Breiman 2001]

$$s_{\mathbf{D}} = \mathbb{E}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X}, Y)).$$

- Assuming $s_{\mathbf{D}} \geq 0$, Chebychev's inequality yields for the **generalization error**

$$E_{\mathbf{D}} \leq \mathrm{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X}, Y)) \,/\, s_{\mathbf{D}}^2.$$

  Chebychev's inequality states that for finite mean and finite non-vanishing variance no more than a certain fraction of values can be further than a certain distance from the mean.

Random Forests: Generalization Error

- **Chebychev's inequality** generally states that for a random variable Z with finite mean μ_Z and finite non-vanishing variance σ_Z^2 and for any $k > 0$:

$$P(|Z - \mu_Z| \geq k\sigma_Z) \leq 1/k^2.$$

- If one chooses $k = \mu_Z/\sigma_Z$, one obtains

$$P(|Z - \mu_Z| \geq \mu_Z) \leq \sigma_Z^2/\mu_Z^2.$$

- Now choose $Z = \gamma_{\mathbf{D}}(\vec{X}, Y)$ as the random variable.
  Then $\mu_Z = \mathbb{E}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X}, Y)) = s_{\mathbf{D}}$ and $\sigma_Z^2 = \text{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X}, Y))$.

- Drawing on the definition of the generalization error, with $s_{\mathbf{D}} > 0$ this leads to

$$\begin{aligned} E_{\mathbf{D}} &= P_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X}, Y) < 0) \\ &= P_{\vec{X},Y}(s_{\mathbf{D}} - \gamma_{\mathbf{D}}(\vec{X}, Y) > s_{\mathbf{D}}) \\ &\leq P_{\vec{X},Y}(|s_{\mathbf{D}} - \gamma_{\mathbf{D}}(\vec{X}, Y)| \geq s_{\mathbf{D}}) \\ &\leq \text{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X}, Y)) \;/\; s_{\mathbf{D}}^2. \end{aligned}$$

Random Forests: Strength and Correlation

- We now derive a more expressive characterization of the generalization error.
- Define the **raw margin function** as [Breiman 2001]

$$\gamma_{\mathbf{D},\Theta}(\vec{x},y) = \mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{x}) = y) - \mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{x}) = \zeta_{\mathbf{D}}(\vec{x},y)), \quad \text{where}$$

$$\zeta_{\mathbf{D}}(\vec{x},y) = \underset{c \in \mathrm{dom}(C); c \neq y}{\operatorname{argmax}} \mathbb{E}_{\Phi}(\mathbb{1}(T_{\Phi|\mathbf{D}}(\vec{x}) = c)).$$

Note: $\zeta_{\mathbf{D}}(\vec{x},y)$ is an *expectation over all possible trees*, but Θ refers to *one tree*.

- The **margin function** is the expectation of the raw margin function: (cf. slide 97)

$$\gamma_{\mathbf{D}}(\vec{x},y) = \mathbb{E}_{\Theta}(\gamma_{\mathbf{D},\Theta}(\vec{x},y)).$$

- We will also need that for any function $f(\Theta)$ it is

$$\left(\mathbb{E}_{\Theta}(f(\Theta))\right)^2 = \mathbb{E}_{\Theta_1,\Theta_2}(f(\Theta_1)\, f(\Theta_2)),$$

provided Θ_1 and Θ_2 are independent and have the same distribution (as Θ).

- This allows us to find a better expression for the variance of the margin function.

Random Forests: Strength and Correlation

- We can now find a better expression for the **variance of the margin function**:

$$\begin{aligned}
\mathrm{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X},Y)) &= \mathrm{cov}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X},Y),\gamma_{\mathbf{D}}(\vec{X},Y)) \\
&= \mathrm{cov}_{\vec{X},Y}(\mathbb{E}_{\Theta_1}(\gamma_{\mathbf{D},\Theta_1}(\vec{X},Y)),\mathbb{E}_{\Theta_2}(\gamma_{\mathbf{D},\Theta_2}(\vec{X},Y))) \\
&= \mathbb{E}_{\Theta_1,\Theta_2}(\mathrm{cov}_{\vec{X},Y}(\gamma_{\mathbf{D},\Theta_1}(\vec{X},Y),\gamma_{\mathbf{D},\Theta_2}(\vec{X},Y))) \\
&= \mathbb{E}_{\Theta_1,\Theta_2}(\rho_{\vec{X},Y}(\Theta_1,\Theta_2,\mathbf{D})\,\sigma_{\vec{X},Y}(\Theta_1,\mathbf{D})\,\sigma_{\vec{X},Y}(\Theta_2,\mathbf{D})),
\end{aligned}$$

  where $\rho_{\vec{X},Y}(\Theta_1,\Theta_2,\mathbf{D}) = \mathrm{corr}_{\vec{X},Y}(\gamma_{\mathbf{D},\Theta_1}(\vec{X},Y),\gamma_{\mathbf{D},\Theta_2}(\vec{X},Y))$ for fixed Θ_1, Θ_2, and $\sigma_{\vec{X},Y}(\Theta,\mathbf{D}) = \mathrm{sd}_{\vec{X},Y}(\gamma_{\mathbf{D},\Theta}(\vec{X},Y))$ for fixed Θ (standard deviation).

- With this we have (using also $E_Z(Z)^2 \leq E_Z(Z^2)$) [Breiman 2001]

$$\mathrm{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X},Y)) = \bar{\rho}_{\mathbf{D}}(\mathbb{E}_{\Theta}(\sigma_{\vec{X},Y}(\Theta,\mathbf{D})))^2 \leq \bar{\rho}_{\mathbf{D}}\,\mathbb{E}_{\Theta}(\sigma^2_{\vec{X},Y}(\Theta,\mathbf{D})),$$

  where $\bar{\rho}_{\mathbf{D}}$ is the mean value of the **correlation**, that is, (cf. slide 100)

$$\bar{\rho}_{\mathbf{D}} = \frac{\mathbb{E}_{\Theta_1,\Theta_2}(\rho_{\vec{X},Y}(\Theta_1,\Theta_2,\mathbf{D})\,\sigma_{\vec{X},Y}(\Theta_1,\mathbf{D})\,\sigma_{\vec{X},Y}(\Theta_2,\mathbf{D}))}{\mathbb{E}_{\Theta_1,\Theta_2}(\sigma_{\vec{X},Y}(\Theta_1,\mathbf{D})\,\sigma_{\vec{X},Y}(\Theta_2,\mathbf{D}))}.$$

Random Forests: Strength and Correlation

- The considered correlation $\bar{\rho}_{\mathbf{D}}$ is the mean value of the (tree) **correlations**

$$\rho_{\vec{X},Y}(\Theta_1, \Theta_2, \mathbf{D}) = \mathrm{corr}_{\vec{X},Y}(\gamma_{\mathbf{D},\Theta_1}(\vec{X}, Y), \gamma_{\mathbf{D},\Theta_2}(\vec{X}, Y)).$$

- Note that this correlation is different from the correlation (cf. slide 91)

$$\rho_{\mathbf{D},\Theta}(\vec{x}) = \mathrm{corr}_{\mathbf{D},\Theta}(T_{\mathbf{D},\Theta_1}(\vec{x}), T_{\mathbf{D},\Theta_2}(\vec{x})),$$

  which is the sampling correlation between pairs of trees, or

$$\rho_{\Theta|\mathbf{D}}(\vec{x}) = \mathrm{corr}_{\mathbf{D},\Theta}(T_{\Theta_1|\mathbf{D}}(\vec{x}), T_{\Theta_2|\mathbf{D}}(\vec{x})),$$

  which is the same, but refers to a fixed (training) data set $\mathbf{D}$.

- The former refers to the margin functions of two trees over the data space.
  - The pair of trees is fixed (the specific trees induced with Θ_1 and Θ_2).
  - The (query) point $\vec{x}$ is varied over the data space ($\vec{X}$ is a random variable).

- The latter refers to the predictions at a specific (query) point $\vec{x}$.
  - The (query) point $\vec{x}$ is fixed (it is *one specific point* in the data space).
  - The pair of trees is varied (Θ_1 and Θ_2 are random variables).

Random Forests: Strength and Correlation

- The second factor in the expression for $\text{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X},Y))$ can be bounded as

$$\begin{aligned}
\mathbb{E}_\Theta(\text{var}_{\vec{X},Y}(\Theta)) &= \mathbb{E}_\Theta(\text{var}_{\vec{X},Y}(\gamma_{\mathbf{D},\Theta}(\vec{X},Y)) \\
&= \mathbb{E}_\Theta\Big(\mathbb{E}_{\vec{X},Y}\Big(\big(\gamma_{\mathbf{D},\Theta}(\vec{X},Y) - \mathbb{E}_{\vec{X},Y}(\gamma_{\mathbf{D},\Theta}(\vec{X},Y))\big)^2\Big)\Big) \\
&= \mathbb{E}_\Theta\Big(\mathbb{E}_{\vec{X},Y}\big(\gamma_{\mathbf{D},\Theta}(\vec{X},Y)^2\big)\Big) - \mathbb{E}_\Theta\Big(\mathbb{E}_{\vec{X},Y}\big(\gamma_{\mathbf{D},\Theta}(\vec{X},Y)\big)^2\Big) \\
&\leq \mathbb{E}_\Theta\Big(\mathbb{E}_{\vec{X},Y}\big(\gamma_{\mathbf{D},\Theta}(\vec{X},Y)^2\big)\Big) - \mathbb{E}_{\vec{X},Y}\Big(\underbrace{\mathbb{E}_\Theta\big(\gamma_{\mathbf{D},\Theta}(\vec{X},Y)\big)}_{=\gamma_{\mathbf{D}}(\vec{X},Y)}\Big)^2 \\
&= \mathbb{E}_\Theta\Big(\mathbb{E}_{\vec{X},Y}\big(\gamma_{\mathbf{D},\Theta}(\vec{X},Y)^2\big)\Big) - s_{\mathbf{D}}^2 \\
&\leq 1 - s_{\mathbf{D}}^2. \qquad \text{(inequalities: } E_Z(Z^2) \geq E_Z(Z)^2 \text{ and } \gamma_{\mathbf{D},\Theta}(\vec{X},Y) \in [-1,1])
\end{aligned}$$

- Therefore an **upper bound for the generalization error** is [Breiman 2001]

$$E_{\mathbf{D}}^{\infty} \leq \bar{\rho}_{\mathbf{D}}\,(1 - s_{\mathbf{D}}^2)\;/\;s_{\mathbf{D}}^2.$$

- Clearly, this bound is the lower,
  - the greater the strength $s_{\mathbf{D}}$ and
  - the smaller the mean correlation $\bar{\rho}_{\mathbf{D}}$.

Random Forests: Strength and Correlation

- Reminder: The margin function of a random forest converges to

$$\begin{aligned}\gamma_{\mathbf{D}}(\vec{x}, y) &= \mathbb{E}_\Theta(\mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{x}) = y)) - \max_{c \in \text{dom}(C); c \neq y} \mathbb{E}_\Theta(\mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{x}) = c)) \\ &= P_\Theta(\quad T_{\Theta|\mathbf{D}}(\vec{x}) = y \quad) - \max_{c \in \text{dom}(C); c \neq y} P_\Theta(\quad T_{\Theta|\mathbf{D}}(\vec{x}) = c \quad).\end{aligned}$$

- If there are only two classes, this limit margin function simplifies to

$$\begin{aligned}\gamma_{\mathbf{D}}(\vec{x}, y) &= P_\Theta(T_{\Theta|\mathbf{D}}(\vec{x}) = y) - \big(1 - P_\Theta(T_{\Theta|\mathbf{D}}(\vec{x}) = y)\big). \\ &= 2P_\Theta(T_{\Theta|\mathbf{D}}(\vec{x}) = y) - 1.\end{aligned}$$

- A requirement of positive strength is thus similar to the weak learning condition

$$\mathbb{E}_{\vec{X},Y}(P_\Theta(T_{\Theta|\mathbf{D}}(X) = Y)) > \tfrac{1}{2},$$

  that is, an individual classifier should be better than random guessing.

- The raw margin function is $2\,\mathbb{1}(T_{\Theta|\mathbf{D}}(X) = Y) - 1$ and the correlation $\bar{\rho}$ is between $\mathbb{1}(T_{\Theta_1|\mathbf{D}}(X) = Y)$ and $\mathbb{1}(T_{\Theta_2|\mathbf{D}}(X) = Y)$.

Random Forests: Out-of-Bag Error

- Reminder **out-of-bag error**: For each data point $(\vec{x}, y)$, construct a predictor by averaging / letting vote only those individual predictors corresponding to bootstrap samples in which the data point $(\vec{x}, y)$ does not appear.
- In random forests, the out-of-bag error can be determined in parallel with the induction of decision trees.

  (Note that this does not mean that the trees have to be induced sequentially. Induction parallelization is still possible: The trees are induced in batches, the size of which is determined by the number of processor cores. The out-of-bag error is (re-)computed after each batch.)
- An out-of-bag error estimate is usually very close to the error estimate obtained by k-fold cross-validation (but unbiased).
- This can be used to determine the number of trees (individuals of the ensemble): **Once the out-of-bag error stabilizes, the training can be terminated.**
- Alternatively, a good number of trees may be found by cross-validation or a user may specify the number of trees beforehand.

Random Forests: Out-of-Bag Strength

- **Strength** can also be estimated using out-of-bag methods.
- The *strength* of a random decision forest (in the limit) is (cf. slide 98)

$$s_{\mathbf{D}} = \mathbb{E}_{\vec{X},Y}\Big(P_\Theta(T_{\Theta|\mathbf{D}}(\vec{x}) = y) - \max_{c \in \mathrm{dom}(C); c \neq y} P_\Theta(T_{\Theta|\mathbf{D}}(\vec{x}) = c)\Big).$$

- Define the relative frequencies $\forall y \in \mathrm{dom}(C)$: [Breiman 2001]

$$Q_{\mathbf{D},\mathcal{T}}(\vec{x}, y) = \frac{\sum_{T \in \mathcal{T}} \mathbb{1}((\vec{x}, y) \in \mathbf{O}_T \wedge T(\vec{x}) = y)}{\sum_{T \in \mathcal{T}} \mathbb{1}((\vec{x}, y) \in \mathbf{O}_T)} \qquad \text{with } \mathbf{O}_T = \mathbf{D} - \mathbf{D}_T,$$

  where $\mathbf{D}_T$ is the in-the-bag data set from which T was induced.

- $Q_{\mathbf{D},\mathcal{T}}(\vec{x}, y)$ is the fraction of votes cast at the point $\vec{x}$ for class y and hence $Q_{\mathbf{D},\mathcal{T}}(\vec{x}, y)$ is an estimate for $P_\Theta(T_{\Theta|\mathbf{D}}(\vec{x}) = y)$.
- As a consequence we get the **strength estimate** [Breiman 2001]

$$\hat{s}_{\mathbf{D},\mathcal{T}} = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x},y) \in \mathbf{D}} \Big(Q_{\mathbf{D},\mathcal{T}}(\vec{x}, y) - \max_{c \in \mathrm{dom}(C); c \neq y} Q_{\mathbf{D},\mathcal{T}}(\vec{x}, c)\Big).$$

Random Forests: Out-of-Bag Correlation

- **Correlation** can also be estimated using out-of-bag methods.
- The mean value of the correlation is (cf. slide 101)

$$\bar{\rho}_{\mathbf{D}} = \frac{\mathrm{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X},Y))}{\mathbb{E}_{\Theta}(\sigma_{\vec{X},Y}(\Theta,\mathbf{D}))^2}.$$

- The numerator of this quotient is (cf. slide 97)

$$\begin{aligned}
&\mathrm{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{x},y)) \\
&= \mathrm{var}_{\vec{X},Y}(P_{\Theta}(T_{\Theta|\mathbf{D}}(\vec{x})=y) - \max_{c\in\mathrm{dom}(C);c\neq y} P_{\Theta}(T_{\Theta|\mathbf{D}}(\vec{x})=c)) \\
&= \mathbb{E}_{\vec{X},Y}\big((P_{\Theta}(T_{\Theta|\mathbf{D}}(\vec{x})=y) - \max_{c\in\mathrm{dom}(C);c\neq y} P_{\Theta}(T_{\Theta|\mathbf{D}}(\vec{x})=c))^2\big) - s_{\mathbf{D}}^2,
\end{aligned}$$

  the first term of which can be estimated as

$$\hat{\delta}^2_{\mathbf{D},\mathcal{T}} = \frac{1}{|\mathbf{D}|}\sum_{(\vec{x},y)\in D}\Big(Q_{\mathbf{D},\mathcal{T}}(\vec{x},y) - \max_{c\in\mathrm{dom}(C);c\neq y} Q_{\mathbf{D},\mathcal{T}}(\vec{x},c)\Big)^2,$$

  while for the strength $s_{\mathbf{D}}$ we can use the out-of-bag estimate $\hat{s}_{\mathbf{D}}$. (cf. slide 106)

Random Forests: Out-of-Bag Correlation

- The denominator is $\mathbb{E}_\Theta(\sigma_{\vec{X},Y}(\Theta,\mathbf{D}))^2 = \mathbb{E}_\Theta(\mathrm{sd}_{\vec{X},Y}(\gamma_{\mathbf{D},\Theta}(\vec{X},Y)))^2$, where $\gamma_{\mathbf{D},\Theta}(\vec{x},y)$ is the raw margin function (cf. slide 100)

$$\begin{aligned}\gamma_{\mathbf{D},\Theta}(\vec{x},y) &= \mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{x}) = y) - \mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{x}) = \zeta_{\mathbf{D}}(\vec{x},y)), \quad \text{where}\\ \zeta_{\mathbf{D}}(\vec{x},y) &= \operatorname*{argmax}_{c\in\mathrm{dom}(C);c\neq y} E_\Phi(\mathbb{1}(T_{\Phi|\mathbf{D}}(\vec{x}) = c)).\end{aligned}$$

(Note that $\zeta_{\mathbf{D}}(\vec{x},y) \neq y$ by definition and that $T_{\Theta|\mathbf{D}}(\vec{x})$ yields one specific class for each $\vec{x}$.)

- Therefore the standard deviation $\sigma_{\vec{X},Y}(\Theta,\mathbf{D})$ is given by [Breiman 2001]

$$\begin{aligned}\sigma_{\vec{X},Y}(\Theta,\mathbf{D}) &= \sqrt{p_1(\Theta,\mathbf{D}) + p_2(\Theta,\mathbf{D}) + (p_1(\Theta,\mathbf{D}) - p_2(\Theta,\mathbf{D}))^2}, \quad \text{where}\\ p_1(\Theta,\mathbf{D}) &= \mathbb{E}_{\vec{X},Y}(\mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{X}) = Y)) \quad \text{and}\\ p_2(\Theta,\mathbf{D}) &= \mathbb{E}_{\vec{X},Y}(\mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{X}) = \zeta_{\mathbf{D}}(\vec{X},Y))).\end{aligned}$$

- Hint: Look at $\mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{X}) = Y)$ versus $\mathbb{1}(T_{\Theta|\mathbf{D}}(\vec{X}) = \zeta_{\mathbf{D}}(\vec{X},Y))$.
  - What are the possible values of their difference?
  - What are the probabilities of these values? (details ⇒ Exercise Sheet 2)

Random Forests: Out-of-Bag Correlation

- For estimation compute for each tree $T \in \mathcal{T}$ and for each data point $(\vec{x}, y) \in \mathbf{D}$ the relative frequencies $Q_{\mathbf{D},\mathcal{T}}(\vec{x}, y)$, $y \in \mathrm{dom}(C)$, and from them estimates

$$\hat{\zeta}_{\mathbf{D},\mathcal{T}}(\vec{x}, y) = \max_{c \in \mathrm{dom}(C); c \neq y} Q_{\mathbf{D},\mathcal{T}}(\vec{x}, y).$$

- Then estimate for each tree $T \in \mathcal{T}$ with $\mathbf{O}_T = \mathbf{D} - \mathbf{D}_T$:

($\mathbf{D}_T$: in-the-bag data for T,
$\mathbf{O}_T$: out-of-bag data for T)

$$\hat{p}_1(T, \mathbf{D}) = \frac{1}{|\mathbf{O}_T|} \sum_{(\vec{x},y) \in \mathbf{O}_T} \mathbb{1}(T(\vec{x}) = y) \qquad \text{and}$$

$$\hat{p}_2(T, \mathbf{D}) = \frac{1}{|\mathbf{O}_T|} \sum_{(\vec{x},y) \in \mathbf{O}_T} \mathbb{1}(T(\vec{x}) = \hat{\zeta}_{\mathbf{D}}(\vec{x}, y)).$$

(Same principle as also used before:
Replace expected values over the whole data space with averages over the given data.)

- With $\hat{p}_1(T, \mathbf{D})$ and $\hat{p}_2(T, \mathbf{D})$ we can estimate

$$\hat{\sigma}_{\mathbf{D}}(T, \mathbf{D}) = \sqrt{\hat{p}_1(T, \mathbf{D}) + \hat{p}_2(T, \mathbf{D}) + (\hat{p}_1(T, \mathbf{D}) - \hat{p}_2(T, \mathbf{D}))^2}.$$

Random Forests: Out-of-Bag Correlation

- With $\hat{p}_1(T, \mathbf{D})$ and $\hat{q}_1(T, \mathbf{D})$ we can estimate

$$\hat{\sigma}_{\mathbf{D}}(T, \mathbf{D}) = \sqrt{p_1(T, \mathbf{D}) + p_2(T, \mathbf{D}) + (p_1(T, \mathbf{D}) - p_2(T, \mathbf{D}))^2}$$

  and thus get as an estimate of the expected value of the standard deviation

$$\bar{\hat{\sigma}}_{\mathbf{D},\mathcal{T}} = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} \hat{\sigma}_{\mathbf{D}}(T, \mathbf{D}).$$

  (Still same principle as also used before:
  Replace expected value over Θ with average over the trees in the random forest.)

- We finally arrive at the **correlation estimate** (cf. slides 106, 107 and above)

$$\hat{\bar{\rho}}_{\mathbf{D},\mathcal{T}} = \frac{\hat{\delta}^2_{\mathbf{D},\mathcal{T}} - \hat{s}^2_{\mathbf{D},\mathcal{T}}}{\bar{\hat{\sigma}}^2_{\mathbf{D},\mathcal{T}}}$$

  of the random forest $\mathcal{T}$, which is an estimate of the (limit) mean correlation

$$\bar{\rho}_{\mathbf{D}} = \frac{\mathrm{var}_{\vec{X},Y}(\gamma_{\mathbf{D}}(\vec{X}, Y))}{\mathbb{E}_{\Theta}(\sigma_{\vec{X},Y}(\Theta, \mathbf{D}))^2}.$$

Random Forests: Prediction Uncertainty

- If the task is regression, the prediction uncertainty may be estimated as the standard deviation of the individual predictions:

$$\sigma(\vec{x}) = \sqrt{\frac{1}{|\mathcal{T}|-1} \sum_{T \in \mathcal{T}} (T(\vec{x}) - \widehat{T}(\vec{x}))^2}, \qquad \text{where} \qquad \widehat{T}(\vec{x}) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} T(\vec{x})$$

  is the prediction of the random forest as a whole.

- If the task is classification, the prediction uncertainty may be characterized by the Shannon entropy of the class vote distribution:

$$\forall c \in \mathrm{dom}(C): \quad p_{\mathcal{T}}(c|\vec{x}) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} \mathbb{1}(T(\vec{x}) = c)),$$

$$\mathrm{U}(\vec{x}) = H(p_{\mathcal{T}}(C|\vec{x})) = -\sum_{c \in \mathrm{dom}(C)} p_{\mathcal{T}}(c|\vec{x}) \log_2 p_{\mathcal{T}}(c|\vec{x}).$$

- A very simple alternative is

$$\mathrm{U}(\vec{x}) = 1 - p_{\mathcal{T}}(c_*|\vec{x}) \quad \text{with} \quad c_* = \operatorname*{argmax}_{c \in \mathrm{dom}(C)} p_{\mathcal{T}}(c|\vec{x}).$$

Random Forests: Variable Importance

- Single decision and regression trees have the advantage that they are **interpretable**: they can easily be checked for plausibility.
- This no longer holds for random forests, due to the multitude of trees.
  (Usually dozens to hundreds of trees are induced; each tree treats a case differently.)
- As a substitute for direct interpretability of the model,
  one may consider **variable (or feature) importance scores**.
- In applications the descriptive variables are seldom equally relevant.
- Often only a few of them have a substantial influence on the prediction;
  the vast majority are irrelevant and could just as well have been excluded.
- It is often useful to learn the relative importance or contribution
  of each descriptive variable in predicting the response.
- This can be a reasonable substitute for the loss of interpretability,
  though one has to pay attention to correlated informative variables.

Reminder: Hypothesis Testing

- A **hypothesis test** is a statistical procedure with which a decision is made between two contrary hypotheses about the process that generated the data.
- The two hypotheses may refer to, for example,
  - the value of a parameter **(Parameter Test)**,
  - a distribution assumption **(Goodness-of-Fit Test)**,
  - the dependence of two attributes **(Dependence Test)**.
- One of the two hypothesis is preferred, that is, in case of doubt the decision is made in its favor. (One says that it gets the "*benefit of the doubt*".)
- The preferred hypothesis is called the **Null Hypothesis** H_0,

  the other hypothesis is called the **Alternative Hypothesis** H_1.
- Intuitively: The null hypothesis H_0 is put on trial. It is accused to be false.

  Only if the evidence is strong enough, it is convicted (that is, rejected).

  If there is (sufficient) doubt, however, it is acquitted (that is, accepted).

Reminder: Hypothesis Testing

- The test decision is based on a **test statistic**,
  that is, a function of the sample values.
- The null hypothesis is rejected if the value of the test statistic
  lies inside the so-called **critical region** C.
  (Outside the critical region C the evidence against the null hypothesis H_0 is not strong enough.)
- Developing a hypothesis test consists in finding the critical region
  for a given test statistic and **significance level** (see below).
- The test decision may be wrong. There are two possible types of errors:

  Type 1 (or α): The null hypothesis H_0 is rejected, even though it is correct.
  Type 2 (or β): The null hypothesis H_0 is accepted, even though it is false.

- Type 1 errors are considered as more severe,
  since the null hypothesis gets the "*benefit of the doubt*".
- Hence it is tried to limit the probability of a type 1 error to a certain maximum α.
  This maximum value α is called **significance level**.

Reminder: Parameter Test

- In a parameter test the contrary hypotheses refer to the value of a parameter, for example (one-sided test):

$$H_0: \quad \theta \geq \theta_0, \qquad\qquad H_1: \quad \theta < \theta_0.$$

- For such a test usually a point estimator T is chosen as the test statistic.
  (A point estimator yields a single "best" value as an estimate for the parameter.)

- The null hypothesis H_0 is rejected if the value t of the point estimator does not exceed a certain value c, the so-called **critical value** (that is, $C = (-\infty, c]$).

- Formally the critical value c is determined as follows: We consider

$$P_{\text{reject}}(\theta) = P_\theta(H_0 \text{ is rejected}) = P_\theta(T \in C),$$

  the probability of rejecting H_0 depending on the parameter θ.

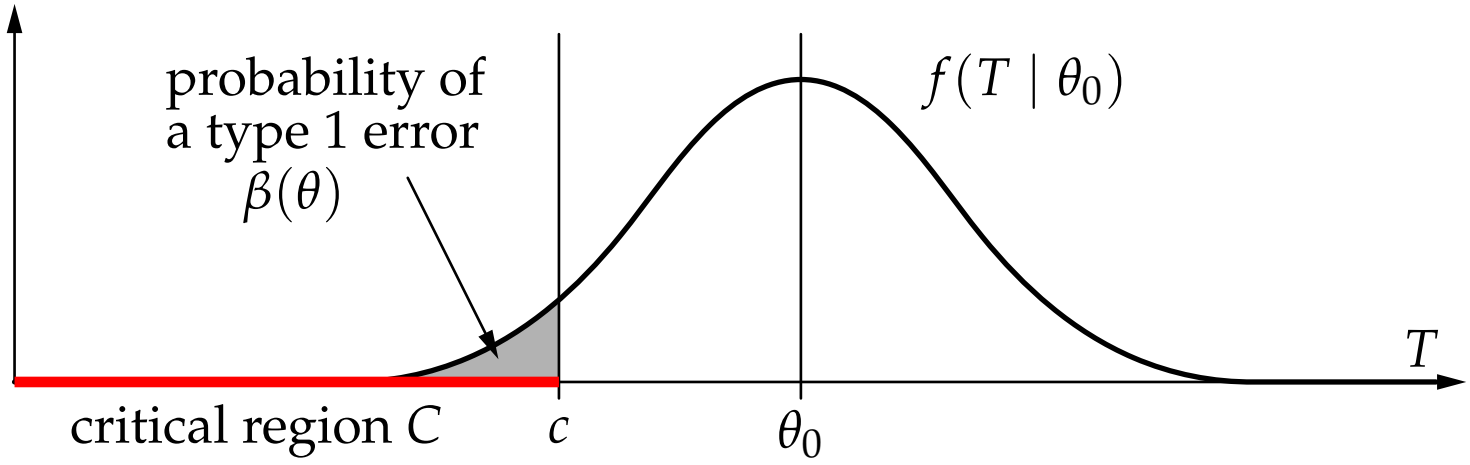
- This value must not exceed the significance level α for values θ satisfying H_0:

$$\max_{\theta\text{: } \theta \text{ satisfies } H_0} P_{\text{reject}}(\theta) \leq \alpha. \qquad \text{(here:} \qquad P_{\text{reject}}(\theta_0) \leq \alpha)$$

Reminder: Parameter Test

- The probability of a type 1 error is the area under the estimator's probability density function $f(T \mid \theta_0)$ to the left of the critical value c.

  (Note: This example illustrates $H_0 : \theta \geq \theta_0$ and $H_1 : \theta < \theta_0$.)

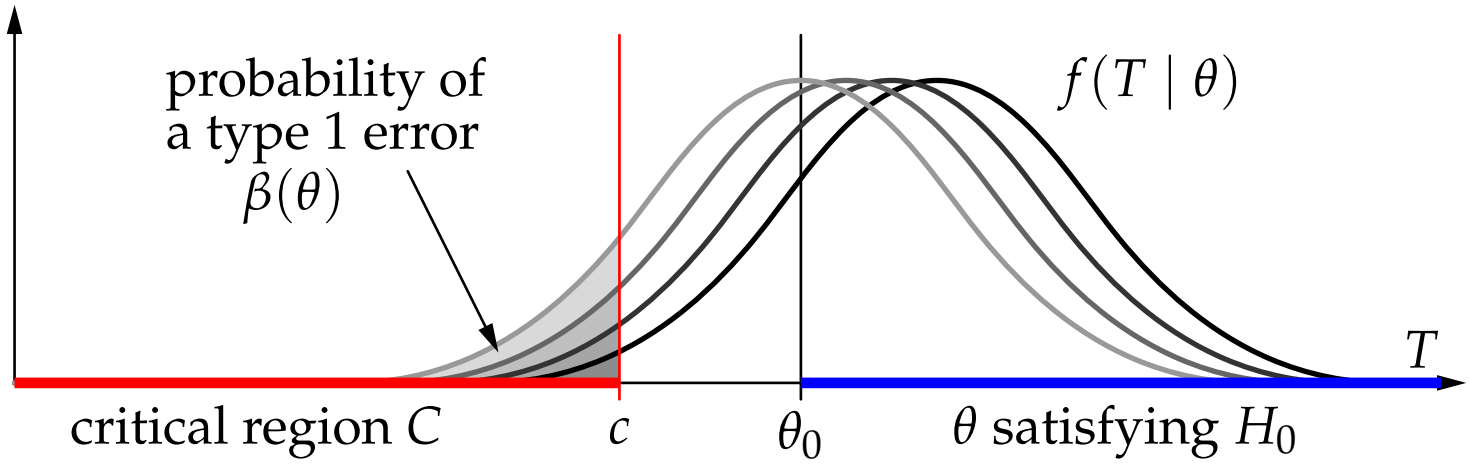


- Obviously the probability of a type 1 error depends on the location of the critical value c: higher values mean a higher error probability.
- Idea: Choose the location of the cricital value so that the maximal probability of a type 1 error equals α, the chosen significance level.

  (That is, limit the probability of a type 1 error to the significance level α.)

Reminder: Parameter Test

- What is so special about θ_0 that we use $f(T \mid \theta_0)$?



- In principle, all θ satisfying H_0 have to be considered, that is, all density functions $f(T \mid \theta)$ with $\theta \geq \theta_0$.
- Among these values θ, the one with the highest probability of a type 1 error (that is, the one with the highest power $\beta(\theta)$) determines the critical value.

  Intuitively: we consider the **worst possible case**.
- General principle: The test result should hold even in the worst possible case.

Reminder: Parameter Test

- Consider a one-sided test of the expected value μ of a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with known variance σ^2, that is, consider the hypotheses

$$H_0: \quad \mu \geq \mu_0, \qquad\qquad H_1: \quad \mu < \mu_0.$$

- As a test statistic we use the standard point estimator for the expected value

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

This point estimator has the probability density

$$f_{\bar{X}}(x) = \mathcal{N}\left(x; \mu, \frac{\sigma^2}{n}\right).$$

- Therefore it is (with the $\mathcal{N}(0,1)$-distributed random variable Z)

$$\alpha = \beta(\mu_0) = P_{\mu_0}(\bar{X} \leq c) = P\left(\frac{\bar{X} - \mu_0}{\sigma/\sqrt{n}} \leq \frac{c - \mu_0}{\sigma/\sqrt{n}}\right) = P\left(Z \leq \frac{c - \mu_0}{\sigma/\sqrt{n}}\right).$$

(Here α is the chosen significance level of the hypothesis test.)

Reminder: Parameter Test

- We have as a result that

$$\alpha = \Phi\left(\frac{c - \mu_0}{\sigma/\sqrt{n}}\right),$$

  where Φ is the distribution function of the standard normal distribution.

- The distribution function Φ is tabulated or approximated by rational functions, because it cannot be represented in closed form. From such a table or approximation we retrieve the value z_α satisfying $\alpha = \Phi(z_\alpha)$ (α-quantile of Φ).

- Then the critical value is

$$c = \mu_0 + z_\alpha \frac{\sigma}{\sqrt{n}}.$$

  (Note that the value of z_α is negative due to the usually small value of α. Typical values are $\alpha = 0.1$, $\alpha = 0.05$ or $\alpha = 0.01$.)

- H_0 is rejected if the value $\bar{x}$ of the point estimator $\bar{X}$ does not exceed c, otherwise it is accepted.

Reminder: Parameter Test

- Example: Let $\sigma = 5.4$, $n = 25$ and $\bar{x} = 128$. We choose $\mu_0 = 130$ and $\alpha = 0.05$.
- From a standard normal distribution table we retrieve $z_{0.05} \approx -1.645$ and get

$$c_{0.05} \approx 130 - 1.645\frac{5.4}{\sqrt{25}} \approx 128.22.$$

  Since $\bar{x} = 128 < 128.22 = c$, we reject the null hypothesis H_0.

- If, however, we had chosen $\alpha = 0.01$, it would have been (with $z_{0.01} \approx -2.326$):

$$c_{0.01} \approx 130 - 2.326\frac{5.4}{\sqrt{25}} \approx 127.49$$

  Since $\bar{x} = 128 > 127.49 = c$, we would have accepted the null hypothesis H_0.

- Instead of fixing a significance level α one may state the so-called **p-value**

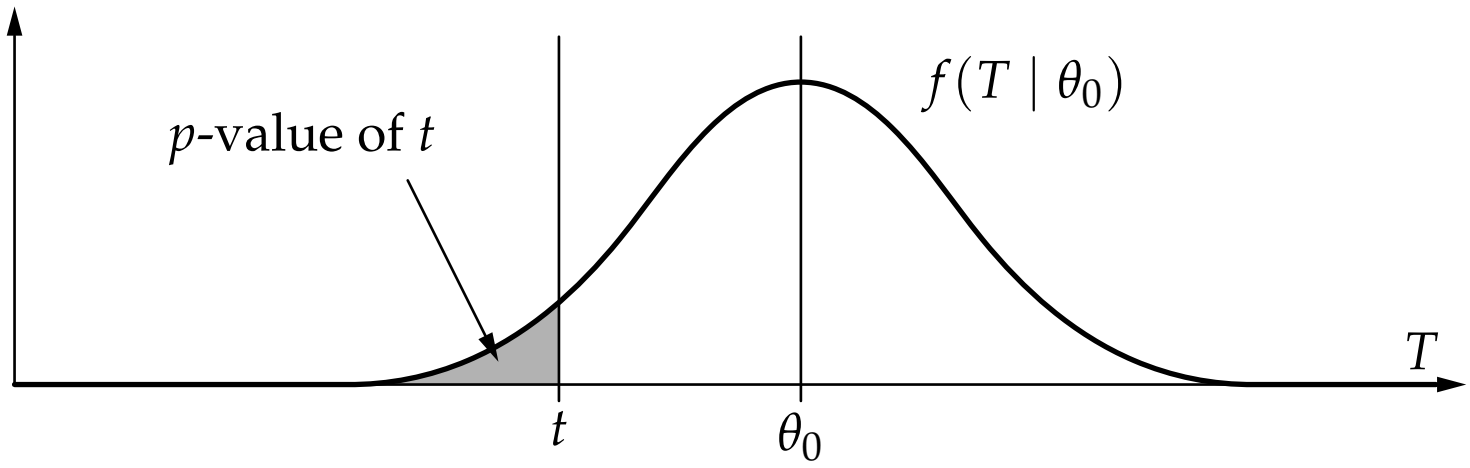
$$p = \Phi\left(\frac{128 - 130}{5.4/\sqrt{25}}\right) \approx 0.032.$$

  For $\alpha \geq p = 0.032$ the null hypothesis is rejected, for $\alpha < p = 0.032$ accepted.

Reminder: p-value

- Let t be the value of the test statistic T that has been computed from a given data set.

  (Note: This example illustrates $H_0 : \theta \geq \theta_0$ and $H_1 : \theta < \theta_0$.)



- The **p-value** is the probability that a value of t or less can be observed for the chosen test statistic T.

- The p-value is a **lower limit for the significance level** α that may have been chosen if we wanted to reject the null hypothesis H_0.

  (...but α has to be chosen *before* knowing p, see below!)

Reminder: p-value

Attention: p-values are often misused or misinterpreted!

- A low p-value does **not** mean that the result is very reliable!

  All that matters for the test result is whether the computed p-value is **below the chosen significance level or not**.

  (A low p-value could just be a chance event, an accident!)

- The significance level may **not** be chosen **after** computing the p-value, since we tend to choose lower significance levels if we know that they are met.

  Doing so would undermine the reliability of the procedure!

- Stating p-values is just a convenient way of avoiding a fixed significance level (since significance levels are a matter of choice and thus user-dependent; they express a user's attitude toward the risk of a wrong test result).

  However: A significance level must still be chosen **before** a reported p-value is looked at.

Excursion: Surrogate Data Tests

- Often, as in the above example, the null hypothesis is represented analytically.
- In this case the distribution of the test statistic under the null hypothesis can (often) be determined analytically.
- Then a p-value can be obtained analytically from this distribution.
  (As demonstrated in the example discussed above.)
- If this is not possible, because the distribution of the test statistic under the null hypothesis is unknown or cannot be obtained in a feasible manner, one may resort to an **implicit representation of the null hypothesis**.
- Core idea: Represent the null hypothesis by **surrogate data sets**.
  - Each surrogate data set is generated in such a way that it can be seen as resulting under the null hypothesis.
  - For a surrogate data set, the test statistic can be computed.
  - By generating sufficiently many surrogate data sets, we can obtain (an approximation of) the distribution of the test statistic.

Excursion: Permutation Tests

- A **permutation test** (also known as **shuffle test** or **re-randomization test**) is a non-parametric hypothesis test that is based on **resampling**.
- Core idea: **Surrogate data sets are obtained by permuting the sample values.**
  - Example: Test for dependence of two variables.
  - By permuting the values of one of the variables over the sample, any dependence between the two variables is destroyed.
  - Therefore the resulting data set can be seen as resulting under the null hypothesis of independence (H_0).
  - By collecting the values of the test statistic for many different permutations, (an approximation of) the test statistic distribution (under H_0) is obtained.
  - From this (approximate) distribution a p-value can be derived by checking where the test statistic for the original sample falls in the sorted list of test statistic values for the permutations.
- Permutation tests exist in many situations where parametric tests do not exist.

Excursion: Permutation Tests

- Example: Do the distributions underlying two samples have the same mean?

  Sample 0: $x_1, x_2, \cdots, x_n$. Sample 1: $y_1, y_2, \cdots, y_m$.

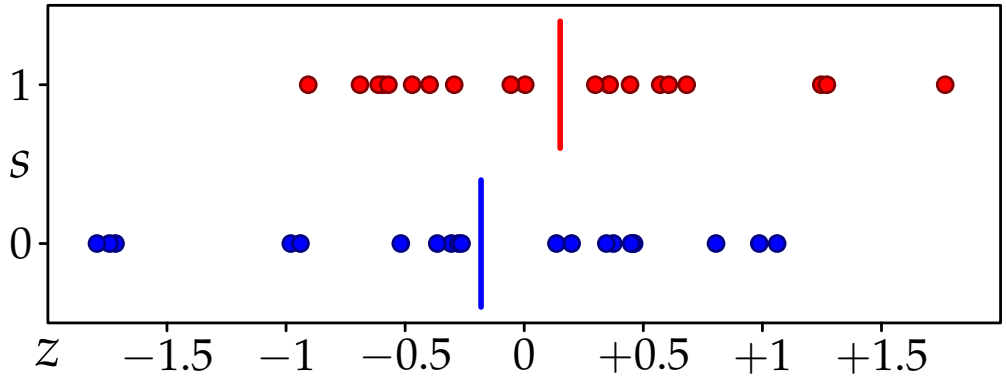
  Hypothesis H_0: $\mu_X = \mu_Y$, Hypothesis H_1: $\mu_X < \mu_Y$.

  Test statistic: mean difference, that is, $d = \left(\frac{1}{m}\sum_{i=1}^{m} y_i\right) - \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)$.

- Concrete example: $n = m = 20$, $X \sim \mathcal{N}(-\frac{1}{20}, 1)$, $Y \sim \mathcal{N}(+\frac{1}{20}, 1)$.

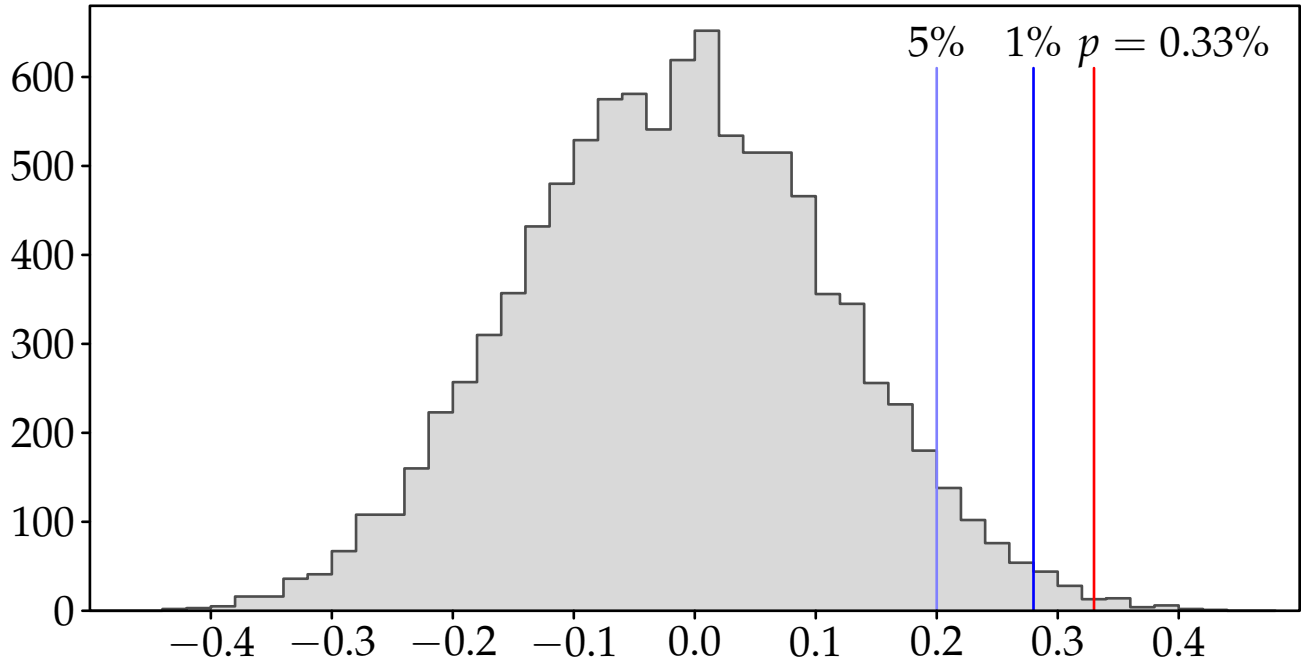
  - Represent data as a table with variables z (value) and s (sample index):
  - Permute values of s. (Randomly reassign values to samples.)
  - Collect test statistics for many permutations.

z	x_1	x_3	$\cdots$	x_{20}	y_1	y_3	$\cdots$	y_{20}
s	0	0	$\cdots$	0	1	1	$\cdots$	1



Excursion: Permutation Tests

- Example: 10000 permutations of the variable s and collection of the mean differences between z for $s = 1$ and z for $s = 0$.
- Distribution of the test statistic values (histogram, bin width 0.02):



light blue: critical value for $\alpha = 5\%$,
dark blue: critical value for $\alpha = 1\%$,
red: p-value for given sample.

Since it is $p = 0.33\% < 1\%$,
H_0 can be rejected at $\alpha = 1\%$,
i.e., (likely) $H_1 \colon \ \mu_X < \mu_Y$ holds.

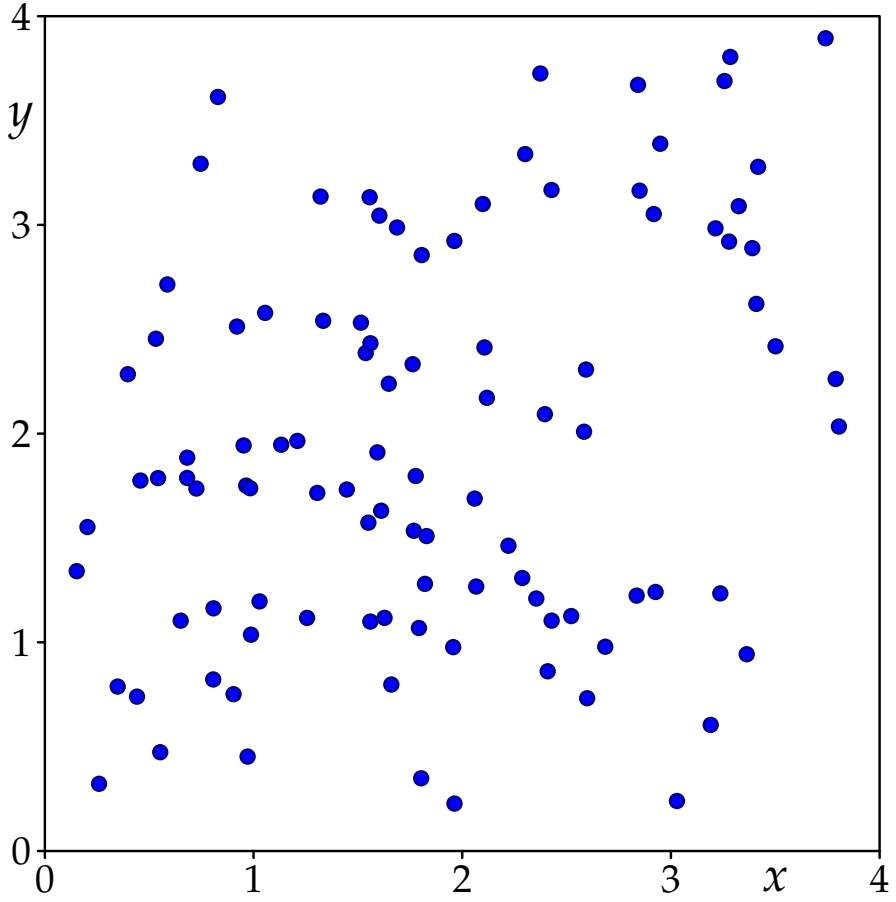
Excursion: Permutation Tests

- Example: Is a (positive) correlation coefficient significant?
  (Consider both Pearson and Spearman correlation.)

- Concrete example: 100 data points (x_i, y_i), $i = 1, \ldots, 100$, are sampled with $x_i = \frac{i}{n} + \varepsilon$ and $y_i = \frac{i}{n} + \varepsilon$, where $\varepsilon \sim \mathcal{U}(0, 3)$ (uniform distribution).

- The values of the y_i are permuted over the sample.
  (One may just as well permute the x_i.)
  This should destroy any correlation between the two variables (other than chance correlation).

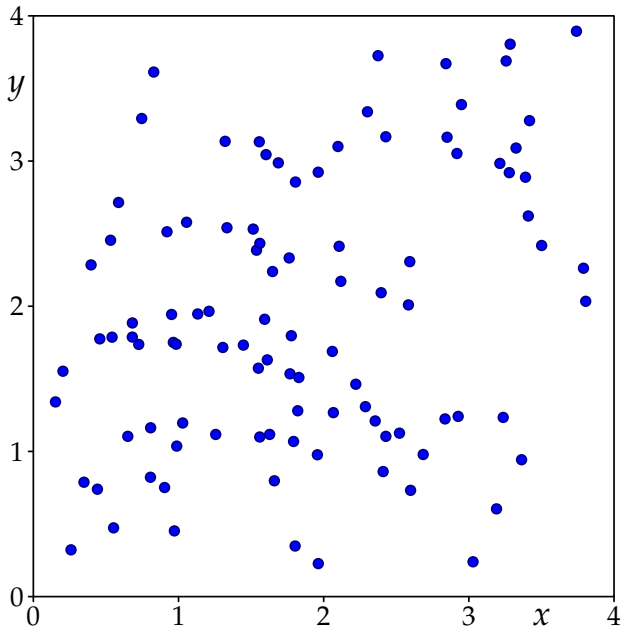
- Collect correlation coefficients for many permutations and thus derive an (approximate) distribution.



$\rho_{\text{Pearson}} \approx 0.292$ $\rho_{\text{Spearman}} \approx 0.251$

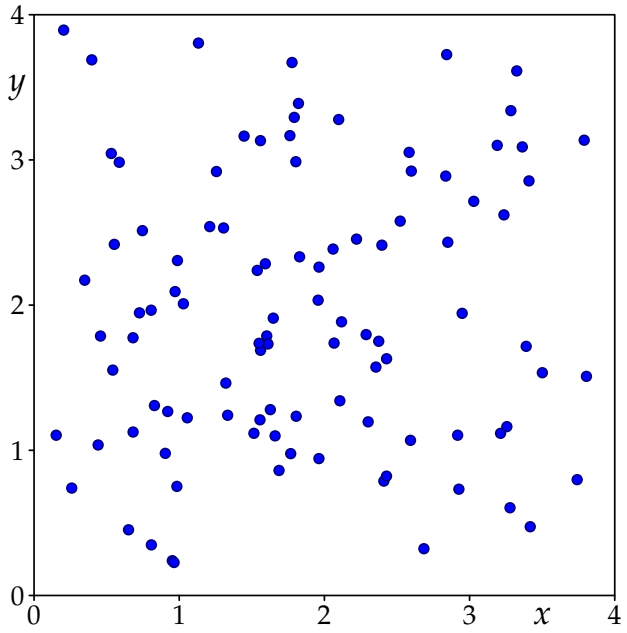
Excursion: Permutation Tests

original sample



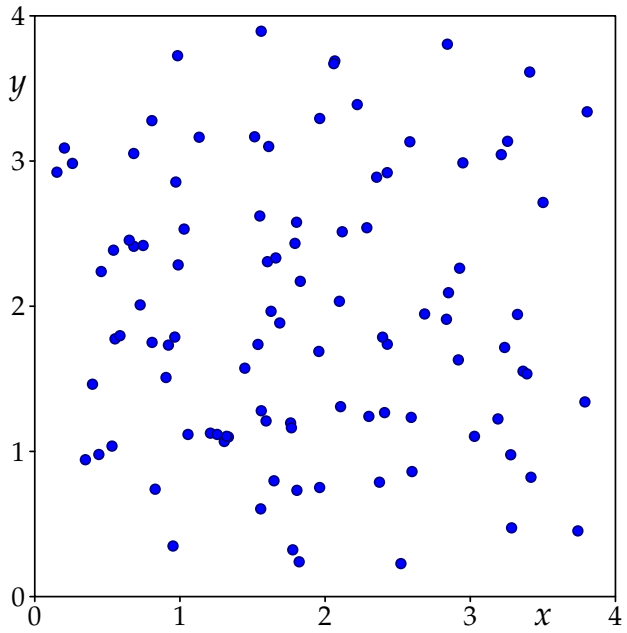
$\rho_{\text{Pearson}} \approx 0.292$ $\rho_{\text{Spearman}} \approx 0.251$

one permutation



$\rho_{\text{Pearson}} \approx 0.069$ $\rho_{\text{Spearman}} \approx 0.065$

another permutation

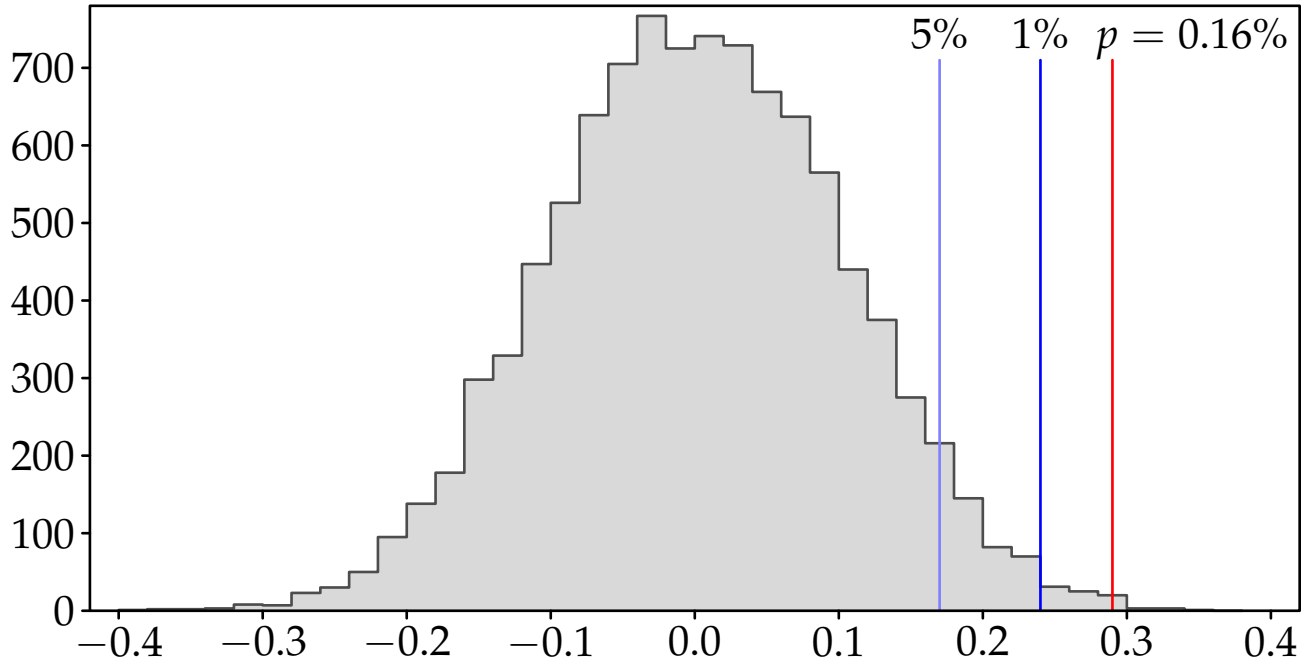


$\rho_{\text{Pearson}} \approx -0.036$ $\rho_{\text{Spearman}} \approx -0.034$

- Note that the correlation coefficients of the permutations are not exactly zero.
- However, they tend to be very small and show mere chance correlation.
- The original sample exhibits significant correlation
  if the observed correlation is unlikely to occur by chance.

Excursion: Permutation Tests

- Example: 10000 permutations of the variable y and collection of the correlation coefficients (both Pearson and Spearman).
- Distribution of Pearson correlation coefficients (histogram, bin width 0.02):

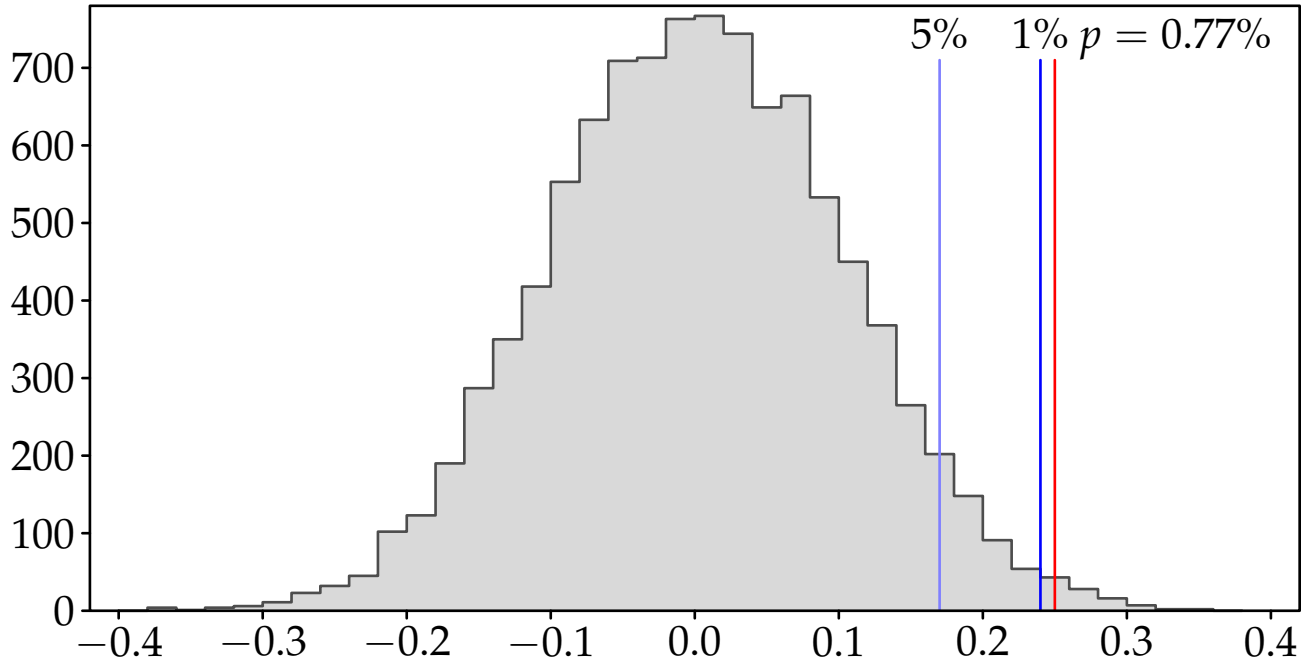


light blue: critical value for $\alpha = 5\%$,
dark blue: critical value for $\alpha = 1\%$,
red: p-value for given sample.

Since it is $p = 0.16\% < 1\%$,
H_0 can be rejected at $\alpha = 1\%$,
i.e., x, y are (likely) correlated.

Excursion: Permutation Tests

- Example: 10000 permutations of the variable y and collection of the correlation coefficients (both Pearson and Spearman).
- Distribution of Spearman correlation coefficients (histogram, bin width 0.02):



light blue: critical value for $\alpha = 5\%$,
dark blue: critical value for $\alpha = 1\%$,
red: p-value for given sample.

Since it is $p = 0.77\% < 1\%$,
H_0 can be rejected at $\alpha = 1\%$,
i.e., x, y are (likely) correlated.

Random Forests: Variable Importance via Permutations

- For each variable X do the following: [Breiman 2001]
  - For each tree T in a random forest $\mathcal{T}$, consider its out-of-bag data set $\mathbf{O}_T$.
  - Permute the values of the variable X over $\mathbf{O}_T$ to produce $\mathbf{O}_{T,\pi(X)}$.
  - Classify all $\vec{x} \in \mathbf{O}_{T,\pi(X)}$ with tree T; record the result for the original of $\vec{x}$.
  - Collect the class votes for all data points in $\mathbf{D}_{\text{oob}} = \bigcup_{T \in \mathcal{T}} \mathbf{O}_T$.
  - Compare the accuracy to the accuracy on $\mathbf{D}_{\text{oob}}$ without permutation.
  - **The accuracy difference is a measure of the importance of variable X.**
- In principle, multiple permutations would have to be considered in order to obtain an average performance under permutation.
- In practice, often only one permutation is considered for efficiency.
- Multiple permutations would allow to conduct a significance test on the variables ($\Rightarrow$ feature selection, but this is costly with permutations).

Random Forests: Variable Importance via Mean Impurity Reduction

- Core idea: In the decision tree induction process, a split is chosen based on how much it **reduces the (expected) impurity of the class distribution**.
- A (descriptive) variable may be used for different splits in a tree, that is, it may be used in multiple inner nodes of the tree, each time producing a different impurity reduction.
- Each such split refers to a different region of the data space, which has a different probability of occurrence.
- **Averaging over all impurity reductions** (respecting occurrence probabilities) yields a (relative) importance measure for the (descriptive) variable.

  In a random forest, these importance values are **averaged over all trees**.
- Since such an importance measure is only relative, it is common to rescale the individual importance values so that the maximum is 100.
- In principle, any impurity measure works, but **Gini impurity** is most common (as used in CART). [Breiman, Friedman, Olshen & Stone 1984]

Reminder: Information Gain

Information Gain (used in ID3) [Kullback and Leibler 1951, Quinlan 1986]

Based on Shannon Entropy $H = -\sum_{i=1}^{n} p_i \log_2 p_i$ (Shannon 1948)

$$I_{\text{gain}}(C, A) = \overbrace{H(C)}^{} - \overbrace{H(C|A)}^{}$$

$$= \overbrace{-\sum_{i=1}^{n_C} p_{i.} \log_2 p_{i.}}^{} - \overbrace{\sum_{j=1}^{n_A} p_{.j} \left(-\sum_{i=1}^{n_C} p_{i|j} \log_2 p_{i|j} \right)}^{}$$

$H(C)$	Entropy of the class distribution (C: class attribute) (intuitively: measures the impurity of the class distribution or the lack of information about the obtaining class)
$H(C\|A)$	*Expected entropy* of the class distribution if the value of the attribute A becomes known
$H(C) - H(C\|A)$	Expected entropy reduction or *information gain*

Reminder: Variants of Information Gain

Normalized Information Gain

- Information gain is biased towards many-valued attributes.
- Normalization removes / reduces this bias.

Information Gain Ratio (used in C4.5) [Quinlan 1986 / 1993]

$$I_{\text{gr}}(C, A) = \frac{I_{\text{gain}}(C, A)}{H(A)} = \frac{I_{\text{gain}}(C, A)}{-\sum_{j=1}^{n_A} p_{\cdot j} \log_2 p_{\cdot j}}$$

Symmetric Information Gain Ratio [López de Mántaras 1991]

$$I^{(1)}_{\text{sgr}}(C, A) = \frac{I_{\text{gain}}(C, A)}{H(A, C)} \qquad \text{or} \qquad I^{(2)}_{\text{sgr}}(C, A) = \frac{I_{\text{gain}}(C, A)}{H(A) + H(C)}$$

"Symmetric" means that $I^{(1)}_{\text{sgr}}(C, A) = I^{(1)}_{\text{sgr}}(A, C)$ and $I^{(2)}_{\text{sgr}}(C, A) = I^{(2)}_{\text{sgr}}(A, C)$.

Reminder: Generalized Entropy and Gini Impurity

- Shannon entropy can be generalized, for example, as: [Daróczy 1970]

$$H_\beta^{(\text{general})} = \frac{2^{\beta-1}}{2^{\beta-1}-1}\sum_{i=1}^{n} p_i(1-p_i^{\beta-1}) = \frac{2^{\beta-1}}{2^{\beta-1}-1}\left(1-\sum_{i=1}^{n} p_i^{\beta}\right)$$

- **Shannon entropy** results as a special case in the limit for $\beta \to 1$:

$$H^{(\text{Shannon})} = \lim_{\beta\to 1} H_\beta^{(\text{general})} = -\sum_{i=1}^{n} p_i \log_2 p_i$$

- Another commonly used variant is **quadratic entropy**, which results for $\beta = 2$:

$$H^{(\text{quad})} = H_2^{(\text{general})} = 2\sum_{i=1}^{n} p_i(1-p_i) = 2 - 2\sum_{i=1}^{n} p_i^2$$

- Intuitively: choosing each alternative s_i with its probability p_i leads to:

$$P(\text{wrong guess}) = \sum_{i=1}^{n} p_i(1-p_i) = 1-\sum_{i=1}^{n} p_i^2 = \tfrac{1}{2}H^{(\text{quad})} = \text{I}_{\text{Gini}}$$

This is also known as **Gini impurity** (= half of quadratic entropy).

Reminder: Evaluation Measures based on Quadratic Entropy

- **Shannon information gain** may be written in two ways:

$$\begin{aligned} I_{\text{gain}}^{(\text{Shannon})}(C,A) &= H^{(\text{Shannon})}(C) - H^{(\text{Shannon})}(C|A) \\ &= -\sum_{i=1}^{n_C} p_{i.} \log_2 p_{i.} - \sum_{j=1}^{n_A} p_{.j} \left(-\sum_{i=1}^{n_C} p_{i|j} \log_2 p_{i|j} \right) \\ &= -\sum_{i=1}^{n_C} p_{i.} \log_2 p_{i.} - \sum_{j=1}^{n_A} p_{.j} \log_2 p_{.j} + \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} p_{ij} \log_2 p_{ij} \\ &= H^{(\text{Shannon})}(C) + H^{(\text{Shannon})}(A) - H^{(\text{Shannon})}(CA) \end{aligned}$$

- For quadratic entropy the two versions differ: [Breiman, Friedman, Olshen & Stone 1984]
  Using the former variant leads to the **Gini index**: (used in CART)

$$\text{Gini}(C,A) = \frac{1}{2}\left(H^{(\text{quad})}(C) - H^{(\text{quad})}(C|A) \right) = \sum_{j=1}^{n_A} p_{.j} \sum_{i=1}^{n_C} p_{i|j}^2 - \sum_{i=1}^{n_C} p_{i.}^2$$

- Using the latter leads to what may be called **quadratic information gain**:

$$I_{\text{gain}}^{(\text{quad})}(C,A) = H^{(\text{quad})}(C) + H^{(\text{quad})}(A) - H^{(\text{quad})}(CA)$$

Random Forests: Variable Importance via Mean Impurity Reduction

- Let $T \in \mathcal{T}$ be a decision tree in a random forest $\mathcal{T}$.
  Let V_T be the set of vertices (or nodes) of the decision tree T.
  Let π_T describe the partitioning of the data space induced by T,
  that is, $\pi_T(v)$ for $v \in V_T$ is the region of the data space covered by v.
  Let $S(v)$ be the split attribute used in v (or the class attribute C for leaves).

- For an inner vertex v of T the **(weighted) reduction in Gini impurity** is

$$\Delta\text{Gini}(v) = P(\pi_T(v)) \cdot \text{Gini}(C, S(v)),$$

  where $P(\pi_T(v))$ is the probability of the data space region $\pi_T(v)$,
  that is, the probability of observing a data point $\vec{x} \in \pi_T(v)$.

- In practice, the probability $P(\pi_T(v))$ of the data space region $\pi_T(v)$
  is estimated from a given data set $\mathbf{D}$ as

$$\hat{P}(\pi_T(v)) = \frac{|\{\vec{x} \in \mathbf{D} \mid \vec{x} \in \pi_T(v)\}|}{|\mathbf{D}|},$$

  that is, as the fraction of data points falling into the region $\pi_T(v)$.

Random Forests: Variable Importance for Regression Trees

- The **importance of variable** X resulting for a single decision tree T is thus

$$I_T(X) = \sum_{v \in V_T} \mathbb{1}(S(v) = X) \cdot P(\pi_T(v)) \cdot \text{Gini}(C, S(v)).$$

- For a random forest $\mathcal{T}$ these importance values are averaged over the trees:

$$I_{\mathcal{T}}(X) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} I_T(X).$$

- Note that these importance values are **relative measures**.
  In variable rankings they are rescaled so that the maximum value is 100.

- Variable importance based on impurity reduction of the class distribution is most commonly computed with the **Gini index**, because this approach was first suggested in connection to CART. [Breiman, Friedman, Olshen & Stone 1984]

- It is convenient with CART, because then the Gini values are directly available.

- However, in principle other impurity (reduction) measures may be used as well.

Random Forests: Variable Importance for Regression Trees

- The prediction quality of a random forest of regression trees is measured by the **sum (or mean) of squared errors** (instead of accuracy)

$$[\text{SSE}|\text{MSE}](Y) \;=\; \left[\frac{1}{|\mathbf{D}|}\right] \sum_{(\vec{x},y)\in\mathbf{D}} (y - \mathcal{T}(\vec{x}))^2,$$

  where $\mathbf{D}$ is a (test/validation) data set, Y is the target variable to predict, and $\mathcal{T}(\vec{x}) = \frac{1}{\mathcal{T}} \sum_{T\in\mathcal{T}} T(\vec{x})$ is the prediction of the random forest $\mathcal{T}$.

- Variable importance may be obtained as the increase of SSE (or MSE) resulting from the permutation of the values of a variable of interest.

- Instead of impurity reduction, regression trees use, for example, (expected) **variance reduction** as it results from a split based on (discretized) variable X:

$$\Delta\sigma^2(Y|X) \;=\; \sigma^2(Y) - \sum_{x\in\text{dom}(X)} \sigma^2(Y \mid X = x) \cdot P(X = x).$$

- (Relative) Variable importance may be obtained as the (weighted) average of the variance reductions that are obtained from a variable of interest.

Gradient Boosting

Gradient Boosted Trees

AdaBoost Revisited

- Reminder: A usual goal is to minimize the **expected loss** (also known as **risk**)

$$\mathbb{E}_{\vec{X},Y}(\mathcal{L}(m(\vec{X}),Y)) = \int_{\mathcal{X}\times\mathcal{Y}} \mathcal{L}\big(m(\vec{x}),y\big)\, p(\vec{x},y)\, d\vec{x}\, dy,$$

  where $p(\vec{x},y)$ is the probability density function of the data generating process, which may also be written as $p(y \mid \vec{x})p(\vec{x})$.

- Loss functions written in terms of the **margin** $\gamma = y\, m(\vec{x})$ at a point $(\vec{x},y)$,

$$\mathcal{L}\big(m(\vec{x}),y\big) = \mathcal{L}^{\circ}\big(y\, m(\vec{x})\big) = \mathcal{L}^{\circ}(\gamma)$$

  with $\mathcal{L}^{\circ}\colon \mathbb{R} \to \mathbb{R}$, are called **margin-based loss functions**.

- The **exponential loss function** is a margin-based loss function that uses

$$\mathcal{L}\big(m(\vec{x}),y\big) = \mathcal{L}^{\circ}\big(y\, m(\vec{x})\big) = \mathcal{L}^{\circ}(\gamma) = e^{-\gamma} = e^{-y\, m(\vec{x})}.$$

- AdaBoost: minimize the **expected exponential loss** with an additive model

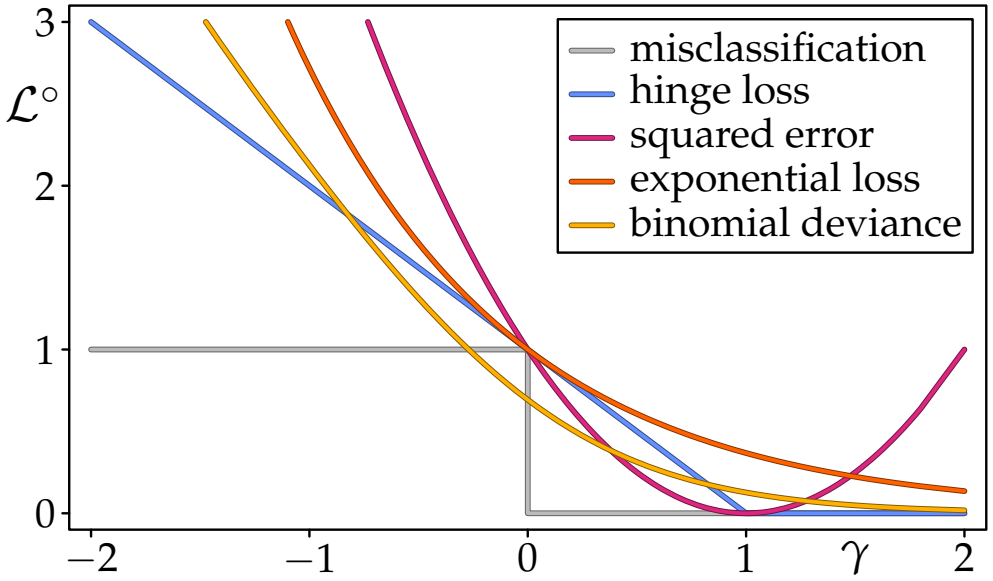
$$m_{\tau}^{\text{joint}}(\vec{x}_i) = \sum_{t=1}^{\tau} \alpha_t m_t(\vec{x}_i), \qquad \text{with the goal} \qquad m_{\tau}^{\text{joint}} = \operatorname*{argmin}_{m} \mathbb{E}_{\vec{X},Y}(e^{-Ym(\vec{X})}).$$

Excursion: Margin-Based Loss Functions

- In (binary) classification, many loss functions are written in terms of the **margin** $\gamma = y\, m(\vec{x})$ at a point $(\vec{x}, y)$, that is,

  $$\mathcal{L}\big(m(\vec{x}), y\big) = \mathcal{L}^\circ\big(y\, m(\vec{x})\big) = \mathcal{L}^\circ(\gamma)$$

  with $\mathcal{L}^\circ : \mathbb{R} \to \mathbb{R}$. These are called **margin-based loss functions**.



- misclassification: $\mathcal{L}^\circ(\gamma) = \mathbb{1}(\gamma < 0)$ (constant almost everywhere, not continuous)
- hinge loss: $\mathcal{L}^\circ(\gamma) = \max(0, 1 - x)$ (used in support vector machines)
- squared error: $\mathcal{L}^\circ(\gamma) = (1 - \gamma)^2$ (not monotone decreasing)
- exponential loss: $\mathcal{L}^\circ(\gamma) = e^{-\gamma}$ (used in AdaBoost)
- binomial deviance: $\mathcal{L}^\circ(\gamma) = \ln(1 + e^{-2\gamma})$ (log-likelihood, cross-entropy)

  (factor 2 to match the result to that of exponential loss, see below)

Minimizing Exponential Loss

- Consider the general task of **minimizing the expected exponential loss**:

$$m_{\text{exp}} \;=\; \operatorname*{argmin}_{m \in \mathcal{M}} \mathbb{E}_{\vec{X},Y}(e^{-Ym(\vec{X})}) \qquad = \operatorname*{argmin}_{m \in \mathcal{M}} \mathbb{E}_{\vec{X}}(\mathbb{E}_{Y|\vec{X}=\vec{x}}(e^{-Ym(\vec{X})})).$$

- Clearly, it suffices to minimize the expectation conditional on $\vec{x}$:

$$\mathbb{E}_{Y|\vec{X}=\vec{x}}(e^{-Ym(\vec{x})}) \;=\; P(Y=+1 \mid \vec{X}=\vec{x})\, e^{-m(\vec{x})} + P(Y=-1 \mid \vec{X}=\vec{x})\, e^{+m(\vec{x})},$$

  because the prediction $m(\vec{x})$ depends only on individual input points $\vec{x}$.

- A necessary condition for the (conditional) expectation to be minimal is that the derivative w.r.t. the model (prediction) vanishes:

$$\frac{\partial}{\partial m(\vec{x})} \mathbb{E}_{Y|\vec{X}=\vec{x}}(e^{-Ym(\vec{x})}) \;\stackrel{!}{=}\; 0.$$

  It follows

$$-P(Y=+1 \mid \vec{X}=\vec{x})\, e^{-m_{\text{exp}}(\vec{x})} \;+\; P(Y=-1 \mid \vec{X}=\vec{x})\, e^{+m_{\text{exp}}(\vec{x})} \;=\; 0$$

$$\Leftrightarrow\quad \frac{P(Y=+1|\vec{X}=\vec{x})}{P(Y=-1|\vec{X}=\vec{x})} \;=\; e^{+2m_{\text{exp}}(\vec{x})} \qquad \Leftrightarrow \qquad m_{\text{exp}}(\vec{x}) \;=\; \tfrac{1}{2} \ln \left(\frac{P(Y=+1|\vec{X}=\vec{x})}{P(Y=-1|\vec{X}=\vec{x})} \right).$$

Minimizing Exponential Loss

- Therefore, the **model minimizing the expected exponential loss** is

$$m_{\text{el}}(\vec{x}) \;=\; \frac{1}{2}\ln\left(\frac{P(Y=+1 \mid \vec{X}=\vec{x})}{P(Y=-1 \mid \vec{X}=\vec{x})}\right).$$

- This implies $\quad p_+(\vec{x}) = P(Y=+1 \mid \vec{X}=\vec{x}) = \dfrac{1}{1+e^{-2m_{\text{exp}}(\vec{x})}}$

  and $\quad p_-(\vec{x}) = P(Y=-1 \mid \vec{X}=\vec{x}) = \dfrac{1}{1+e^{+2m_{\text{exp}}(\vec{x})}}.$

- This is closely related to **logistic classification**, which assumes a linear model:

$$\ln\left(\frac{P(Y=+1 \mid \vec{X}=\vec{x})}{P(Y=-1 \mid \vec{X}=\vec{x})}\right) \;=\; a_0 + \sum_{i=1}^{m} a_i x_i \;=\; \vec{a}^{\circ\top}\vec{x}^{\circ} \;=\; m_{\text{lin}}(\vec{x}).$$

  where $\quad \vec{a}^{\circ} = (a_0, a_1, \ldots, a_m)^\top \quad$ and $\quad \vec{x}^{\circ} = (1, x_1, \ldots, x_m)^\top.$

- Logistic classification tries to maximize the **expected (log-)likelihood** (or: minimize the cross-entropy, or: minimize the binomial deviance).

Logistic Classification

- **Logistic classification** leads to $p_+(\vec{x}) = P(Y=+1 \mid \vec{X}=\vec{x}) = \frac{1}{1+e^{-\vec{a}^{\circ\top}\vec{x}^{\circ}}}$

  and $p_-(\vec{x}) = P(Y=-1 \mid \vec{X}=\vec{x}) = \frac{1}{1+e^{+\vec{a}^{\circ\top}\vec{x}^{\circ}}}$,

  where $\vec{a}^{\circ} = (a_0, a_1, \ldots, a_m)^\top$ and $\vec{x}^{\circ} = (1, x_1, \ldots, x_m)^\top$.

- That is, it is assumed that the probability $p_+(\vec{x}) = P(Y=+1 \mid \vec{X}=\vec{x})$ can be described by a **logistic function** $f(\vec{a}^{\circ\top}\vec{x}^{\circ}) = (1+e^{-\vec{a}^{\circ\top}\vec{x}^{\circ}})^{-1}$.

- Generalizing from a linear to a (more) general model yields

$$p_+(\vec{x}) = \frac{1}{1+e^{-m(\vec{x})}} \quad \text{and} \quad p_-(\vec{x}) = \frac{1}{1+e^{+m(\vec{x})}}.$$

- The **expected likelihood** given some model m is then

$$\begin{aligned}
&\mathbb{E}_{\vec{X}}\big(\mathbb{E}_{Y|\vec{X}=\vec{x}}\big(p_+(\vec{x})^{\mathbb{1}(Y=+1)}\, p_-(\vec{x})^{\mathbb{1}(Y=-1)}\big)\big) \\
&= \mathbb{E}_{\vec{X}}\big(\mathbb{E}_{Y|\vec{X}=\vec{x}}\big((1+e^{-m(\vec{x})})^{-\mathbb{1}(Y=+1)} \cdot (1+e^{+m(\vec{x})})^{-\mathbb{1}(Y=-1)}\big)\big)
\end{aligned}$$

Logistic Classification

- Considering the (conditional) **expected log-likelihood** yields

$$\begin{aligned}
&\mathbb{E}_{Y|\vec{X}=\vec{x}}\left(-\mathbb{1}(Y=+1)\cdot\ln(1+e^{-m(\vec{x})})-\mathbb{1}(Y=-1)\cdot\ln(1+e^{+m(\vec{x})})\right)\\
&= \mathbb{E}_{Y|\vec{X}=\vec{x}}\left(-\mathbb{1}(Y=+1)\cdot\ln(1+e^{-Ym(\vec{x})})-\mathbb{1}(Y=-1)\cdot\ln(1+e^{-Ym(\vec{x})})\right)\\
&= \mathbb{E}_{Y|\vec{X}=\vec{x}}\left(-\ln(1+e^{-Ym(\vec{x})})\right)
\end{aligned}$$

- The **deviance** is generally defined as -2 times the log-likelihood and may be used as a loss function (here: **binomial deviance**):

$$\mathcal{L}(m(\vec{x}),y)=[2]\ln(1+e^{-y\,m(\vec{x})}).$$

(The constant factor 2 is sometimes neglected, because it does not influence minimization.)

- The best model for the binomial deviance loss is (see above)

$$m_{\text{bdl}}(\vec{x})=\ln\left(\frac{P(Y=+1\mid\vec{X}=\vec{x})}{P(Y=-1\mid\vec{X}=\vec{x})}\right).$$

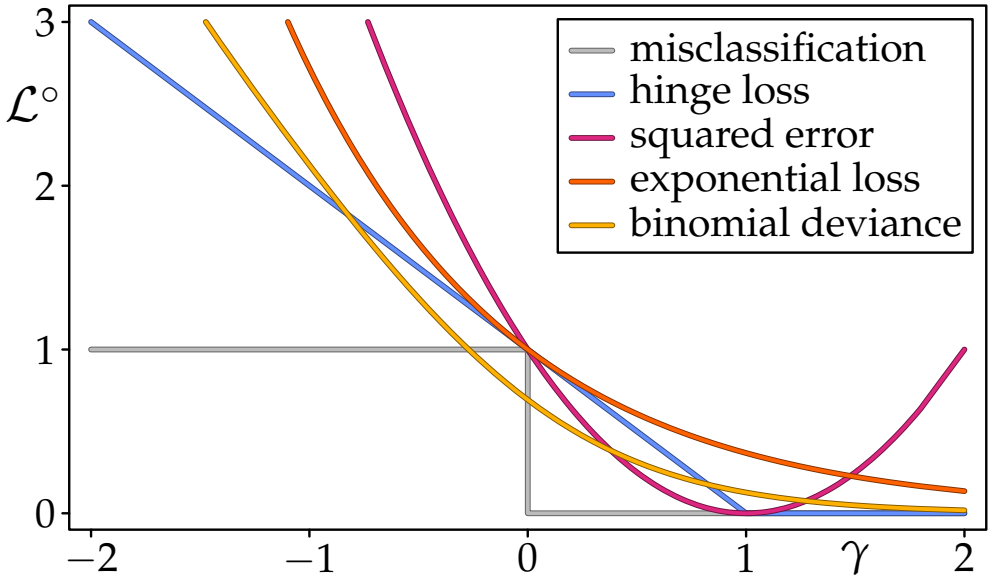
It differs from the **model minimizing exponential loss** only by a factor of 2.

Reminder: Margin-Based Loss Functions

- In (binary) classification, many loss functions are written in terms of the **margin** $\gamma = y\, m(\vec{x})$ at a point $(\vec{x}, y)$, that is,

  $$\mathcal{L}\big(m(\vec{x}), y\big) = \mathcal{L}^\circ\big(y\, m(\vec{x})\big) = \mathcal{L}^\circ(\gamma)$$

  with $\mathcal{L}^\circ : \mathbb{R} \to \mathbb{R}$. These are called **margin-based loss functions**.



- misclassification: $\mathcal{L}^\circ(\gamma) = \mathbb{1}(\gamma < 0)$ (constant almost everywhere, not continuous)
- hinge loss: $\mathcal{L}^\circ(\gamma) = \max(0, 1 - x)$ (used in support vector machines)
- squared error: $\mathcal{L}^\circ(\gamma) = (1 - \gamma)^2$ (not monotone decreasing)
- exponential loss: $\mathcal{L}^\circ(\gamma) = e^{-\gamma}$ (used in AdaBoost)
- binomial deviance: $\mathcal{L}^\circ(\gamma) = \ln(1 + e^{-2\gamma})$ (log-likelihood, cross-entropy)

(factor 2 to match the result to that of exponential loss)

Additive Modeling

- Instead of completely general models, we consider **additive models**, that is,

$$m_\tau^{\text{joint}}(\vec{x}) = m_0(\vec{x}) + \sum_{t=1}^{\tau} \alpha_t m_t(\vec{x}).$$

- Such a model is to be **built incrementally** (or: **sequentially**), that is,

$$\begin{aligned} m_0^{\text{joint}}(\vec{x}) &= m_0(\vec{x}) = c, \\ m_t^{\text{joint}}(\vec{x}) &= m_{t-1}^{\text{joint}}(\vec{x}) + \alpha_t m_t(\vec{x}), \end{aligned}$$

  where c is a constant, e.g. $c = \mathbb{E}_{\vec{X},Y}(Y)$ (or data set mean) for regression or $c = \text{sign}(\mathbb{E}_{\vec{X},Y}(Y))$ (or majority class) for (binary) classification.

- The model m_t and the weight α_t are to be chosen in such a way that some loss function (e.g. exponential loss) is maximally reduced.

- We now re-derive the original AdaBoost algorithm (with $m_t(\vec{x}) \in \{-1, +1\}$) and a continuous alternative (with $m_t(\vec{x}) \in \mathbb{R}$, also known as "Real AdaBoost").

  The latter often yields somewhat better results and converges faster.

Additive Modeling for Exponential Loss

- Consider the task of **minimizing the expected exponential loss**:

$$m_{\text{el}} \;=\; \underset{m \in \mathcal{M}}{\operatorname{argmin}}\, \mathbb{E}_{\vec{X},Y}\left(e^{-Ym(\vec{X})}\right).$$

- Suppose we have a model $m_{t-1}^{\text{joint}}(\vec{x})$ and seek an improved model

$$m_t^{\text{joint}}(\vec{x}) \;=\; m_{t-1}^{\text{joint}}(\vec{x}) + \alpha_t m_t(\vec{x}).$$

- Inserting into the expected exponential loss and computing the second-order Taylor expansion at/around $m_t(\vec{x}) = 0$ for fixed α_t (and fixed $\vec{x}$) yields

$$\begin{aligned}
\mathbb{E}_{\vec{X},Y}\left(e^{-Ym_t^{\text{joint}}(\vec{X})}\right) &= \mathbb{E}_{\vec{X},Y}\left(e^{-Y\left(m_{t-1}^{\text{joint}}(\vec{X})+\alpha_t m_t(\vec{X})\right)}\right) \\
&\approx \mathbb{E}_{\vec{X},Y}\left(e^{-Ym_{t-1}^{\text{joint}}(\vec{X})}\left(1 - Y\alpha_t m_t(\vec{X}) + \tfrac{1}{2}Y^2\alpha_t^2 m_t(\vec{X})^2\right)\right) \\
&= \mathbb{E}_{\vec{X},Y}\left(e^{-Ym_{t-1}^{\text{joint}}(\vec{X})}\left(1 - Y\alpha_t m_t(\vec{X}) + \tfrac{1}{2}\alpha_t^2\right)\right),
\end{aligned}$$

  where the last step follows from $Y \in \{-1,+1\}$ and $m_t(\vec{x}) \in \{-1,+1\}$.

Reminder: Taylor Series / Taylor Expansion

- Let $f(u)$ be a real-valued function, that is, $f : \mathbb{R} \to \mathbb{R}$, that is infinitely differentiable at a real number a.
- Then the **Taylor series** of f at/around a is the power series [Brook Taylor 1715]

$$f(a) + \tfrac{1}{1!} f'(a)(u-a) + \tfrac{1}{2!} f''(a)(u-a)^2 + \tfrac{1}{3!} f'''(a)(u-a)^3 + \cdots$$
$$= \sum_{k=0}^{\infty} \tfrac{1}{i!} f^{(i)}(a)(u-a)^n.$$

- The **Maclaurin series** of f is its Taylor series at $a = 0$. [Colin Maclaurin 17??]
- The second-order Taylor expansion is the above series up to the term with $f''(a)$.
- The Taylor expansion on the preceding slide uses

$$f(u) = f\big(\underbrace{m_{t-1}^{\text{joint}}(\vec{x})}_{u}\big) = e^{-y\left(m_{t-1}^{\text{joint}}(\vec{x}) + \alpha_t m_t(\vec{x})\right)} = e^{-y m_{t-1}^{\text{joint}}(\vec{x}) - y\alpha_t m_t(\vec{x})}$$

and expands at/around $a = m_t(\vec{x}) = 0$ (and hence it is a Maclaurin expansion).

Additive Modeling for Exponential Loss

- Let $w(\vec{x}, y) = e^{-y m_{t-1}^{\text{joint}}(\vec{x})}$ be a weighting function and define

$$\mathbb{E}^{(w)}_{Y|\vec{X}=x}(g(\vec{x}, Y)) \;=\; \frac{\mathbb{E}_{Y|\vec{X}=\vec{x}}(w(\vec{x}, Y) g(\vec{x}, Y))}{\mathbb{E}_{Y|\vec{X}=\vec{x}}(w(\vec{x}, Y))}$$

  as the **weighted conditional expectation** of the function g.

- With this, a pointwise minimization w.r.t. $m_t(\vec{x}) \in \{-1, +1\}$ can be written as

$$m_t^{(\text{opt})} \;=\; \underset{m_t \in \mathcal{M}}{\operatorname{argmin}}\, \mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}\big(1 - Y \alpha_t m_t(\vec{x}) + \tfrac{1}{2}\alpha_t^2\big)$$

- For $\alpha_t > 0$, minimizing the above expression is equivalent to maximizing

$$m_t^{(\text{opt})} \;=\; \underset{m_t \in \mathcal{M}}{\operatorname{argmax}}\, \mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}\big(Y m_t(\vec{x})\big).$$

- The solution is (predict the (weighted) more likely class)

$$m_t^{(\text{opt})}(\vec{x}) = \begin{cases} +1, & \text{if } \mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}(Y) = P_w(Y=+1|\vec{X}=\vec{x}) - P_w(Y=-1|\vec{X}=\vec{x}) > 0, \\ -1, & \text{otherwise.} \end{cases}$$

Additive Modeling for Exponential Loss

- By rewriting the objective as (using again $Y \in \{-1, +1\}$ and $m_t(\vec{x}) \in \{-1, +1\}$)

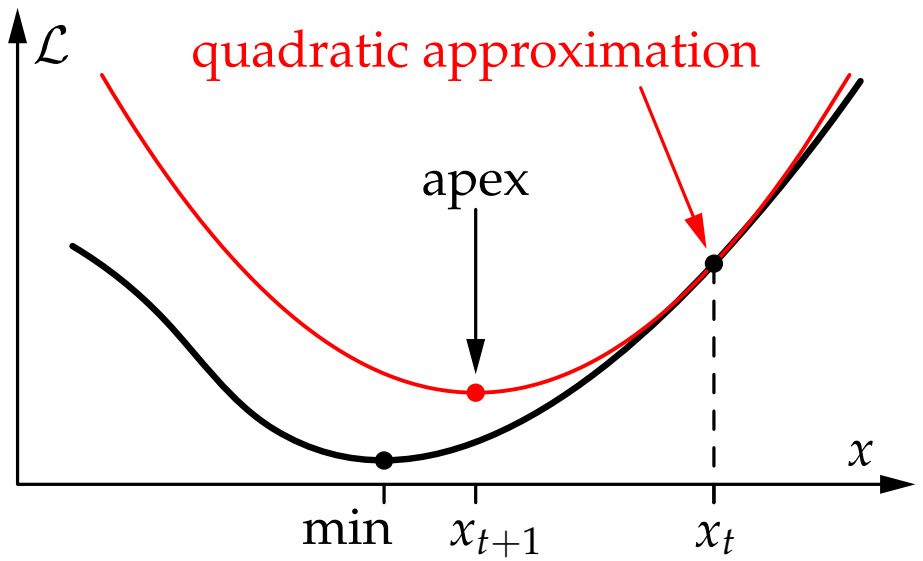
$$\begin{aligned} \mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}\big((Y - m_t(\vec{x}))^2\big)/2 - 1 &= \mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}\big(Y^2 - 2Ym_t(\vec{x}) + m_t(\vec{x})^2\big)/2 - 1 \\ &= -\mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}(Ym_t(\vec{x})), \end{aligned}$$

  we can interpret the optimization as **minimizing a quadratic approximation**.

  (Reminder: Minimizing a quadratic function means finding the apex of a parabola.)

- Minimizing the quadratic approximation leads to a **weighted least-squares choice** of $m_t(\vec{x}) \in \{-1, +1\}$.

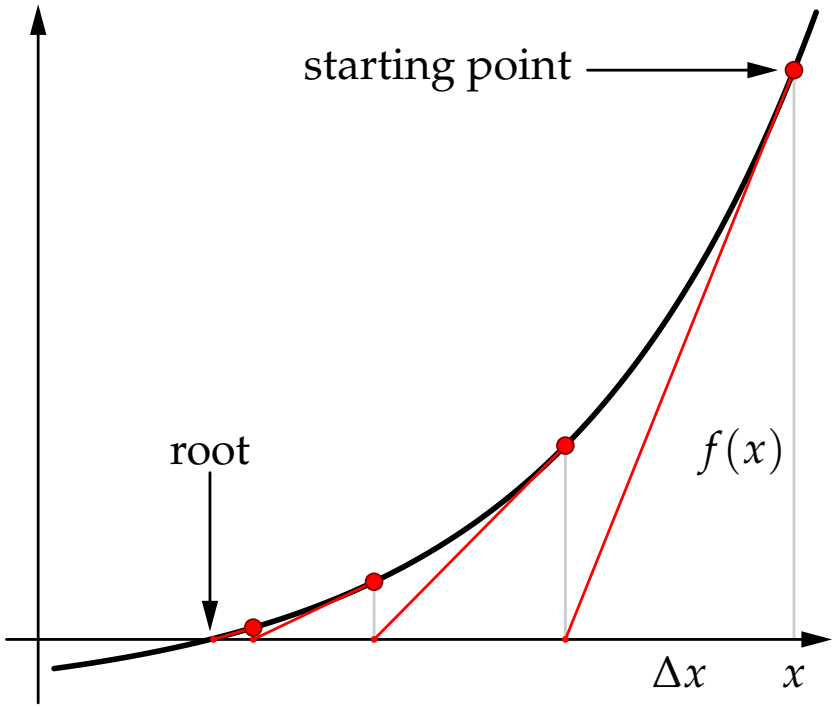
- Making one Newton–Raphson step on a quadratic approximation jumps directly to the minimum of the quadratic approximation.



Reminder: Newton–Raphson Method

- The Newton–Raphson method is an iterative numeric algorithm to approximate a root of a function.
- Idea: use slope/direction of a tangent to the function at a current point to find the next approximation.
- Formally:

$$\vec{x}_{t+1} = \vec{x}_t + \Delta\vec{x}_t; \quad \Delta\vec{x}_t = -\frac{f(\vec{x}_t)}{\nabla_{\vec{x}} f(\vec{x}_t)}$$



- The gradient describes the direction of steepest ascent of tangent (hyper-)planes to the function (in one dimension: the slope of tangents to the function).
- In one dimension (see diagram):

  Solve $(x_{t+1}, 0) = (x_t, f(x_t))^\top + k \cdot (1, \frac{df}{dx}(x_t))^\top$, $k \in \mathbb{R}$, for x_{t+1}.

  Since $0 = f(x_t) + k \cdot \frac{df}{dx}(x_t)$, it is $k = -f(x_t) / \frac{df}{dx}(x_t)$.

Reminder: Newton–Raphson Method for Finding Optima

- The standard Newton-Raphson method finds roots of functions.
- By **applying the Newton-Raphson method to the gradient** of a function, it may be used to **find optima** (minima, maxima, or saddle points), because a vanishing gradient is a necessary condition for an optimum.
- In this case the update formula is

$$\vec{x}_{t+1} = \vec{x}_t + \Delta\vec{x}_t, \qquad \Delta\vec{x}_t = -\left(\nabla^2_{\vec{x}} f(\vec{x}_t)\right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t),$$

  where $\nabla^2_{\vec{x}} f(\vec{x}_t)$ is the so-called **Hessian matrix**, that is, the matrix of second-order partial derivatives of the scalar-valued function f.
- In one dimension:

$$x_{t+1} = x_t + \Delta x_t, \qquad \Delta x_t = -\frac{\frac{df}{dx}(x_t)}{\frac{d^2f}{dx^2}(x_t)}.$$

- For a quadratic function, this method finds the optimum in one step (. . . and this is what was referred to before).

Reminder: Newton–Raphson Method for Finding Optima

- **Derivation of the Newton–Raphson Method for Finding Optima**
  in one dimension using a second-order Taylor expansion of the function.
- **Objective:** Solve the optimization (minimization/maximization) problem

$$x_{t+1} = x_t + \Delta x_t, \qquad \Delta x_t = \operatorname{argopt}_{\Delta x \in \mathbb{R}} f(x_t + \Delta x).$$

- **Approach:** Approximate f in the vicinity of the current estimate x_t
  by a second-order Taylor expansion (quadratic approximation):

$$f(x_t + \Delta x) \approx f(x_t) + \frac{df}{dx}(x_t)\Delta x + \frac{1}{2}\frac{d^2 f}{dx^2}(x_t)\Delta x^2.$$

- A necessary condition for an optimum is that the derivative w.r.t. Δx vanishes:

$$\frac{d}{d\Delta x}\left(f(x_t) + \frac{df}{dx}(x_t)\Delta x + \frac{1}{2}\frac{d^2 f}{dx^2}(x_t)\Delta x^2\right) = \frac{df}{dx}(x_t) + \frac{d^2 f}{dx^2}(x_t)\Delta x \stackrel{!}{=} 0.$$

- It follows

$$\Delta x_t = -\frac{\frac{df}{dx}(x_t)}{\frac{d^2 f}{dx^2}(x_t)}, \qquad \text{which “jumps” to the apex of the parabola.}$$

Additive Modeling for Exponential Loss

- Given $m_t(\vec{x}) \in \{-1, +1\}$, α_t can be found as ($\mathbb{E}^{(w)}_{\vec{X},Y}$: weighted expectation)

$$\begin{aligned}
\alpha_t &= \operatorname*{argmin}_{\alpha \in \mathbb{R}} \mathbb{E}_{\vec{X},Y}\left(e^{-Y\left(m^{\text{joint}}_{t-1}(\vec{X}) + \alpha m_t(\vec{X})\right)}\right) \\
&= \operatorname*{argmin}_{\alpha \in \mathbb{R}} \mathbb{E}_{\vec{X},Y}\left(\underbrace{e^{-Y m^{\text{joint}}_{t-1}(\vec{X})}}_{w(\vec{X},Y)} e^{-\alpha Y m_t(\vec{X})}\right) = \operatorname*{argmin}_{\alpha \in \mathbb{R}} \mathbb{E}^{(w)}_{\vec{X},Y}\left(e^{-\alpha Y m_t(\vec{X})}\right).
\end{aligned}$$

- A necessary condition for a minimum is that the derivative w.r.t. α vanishes

$$\begin{aligned}
\frac{d}{d\alpha}\mathbb{E}^{(w)}_{\vec{X},Y}\left(e^{-\alpha Y m_t(\vec{X})}\right) &= \frac{d}{d\alpha_t}\mathbb{E}^{(w)}_{\vec{X},Y}\left(e^{-\alpha \mathbb{1}(Y = m_t(\vec{X}))} \cdot e^{+\alpha \mathbb{1}(Y \neq m_t(\vec{X}))}\right) \\
&= -e^{-\alpha} P_w\left(Y = m_t(\vec{X})\right) + e^{+\alpha} P_w\left(Y \neq m_t(\vec{X})\right) \stackrel{!}{=} 0.
\end{aligned}$$

- It follows

$$\begin{aligned}
& e^{-\alpha_t} P_w\left(Y = m_t(\vec{X})\right) = e^{+\alpha_t} P_w\left(Y \neq m_t(\vec{X})\right) \\
\Leftrightarrow\ & -\alpha_t + \ln P_w(Y = m_t(\vec{X})) = +\alpha_t + \ln P_w(Y \neq m_t(\vec{X})) \\
\Leftrightarrow\ & \alpha_t = \frac{1}{2}\ln\left(\frac{1 - P_w(Y \neq m_t(\vec{X}))}{P_w(Y \neq m_t(\vec{X}))}\right) = \frac{1}{2}\ln\left(\frac{1 - e_t}{e_t}\right).
\end{aligned}$$

AdaBoost as Additive Modeling

- Adding $m_t(\vec{x})$ thus updates the weighting function as

$$w(\vec{x}, y) \leftarrow w(\vec{x}, y) e^{-\alpha_t Y m_t(\vec{x})},$$

  which due to $-y m_t(\vec{x}) = 2 \cdot \mathbb{1}(Y \neq m_t(\vec{X})) - 1$ can be written as

$$w(\vec{x}, y) \leftarrow w(\vec{x}, y) \exp\left(\ln\left(\tfrac{1-e_t}{e_t}\right) \mathbb{1}(y \neq m_t(\vec{x}))\right).$$

- This is the **distribution-based version** of the data set version on slide 51.
- Applied to decision trees as the model class for the models m_t,
  - the weighted conditional expectation
    is approximated by the leaf node weighted averages,
  - **the weighted least-squares criterion may be used to grow the tree**,
  - the constant α_t can be computed from the weighted training error of m_t.
- Up to now we assumed that the model m_t to be added is weighted with α_t.
  Now we explore the case in which there is no weight α_t (or $\alpha_t = 1$).

AdaBoost as Additive Modeling

- Suppose we have a model $m_{t-1}^{\text{joint}}(\vec{x})$ and seek an improved model

$$m_t^{\text{joint}}(\vec{x}) \;=\; m_{t-1}^{\text{joint}}(\vec{x}) + m_t(\vec{x}).$$

- Consider again the task of **minimizing the expected exponential loss**

$$\begin{aligned}
\mathbb{E}_{Y|\vec{X}=\vec{x}}\left(e^{-Y m_t^{\text{joint}}(\vec{x})}\right) &= \mathbb{E}_{Y|\vec{X}=\vec{x}}\left(e^{-Y\left(m_{t-1}^{\text{joint}}(\vec{x})+m_t(\vec{x})\right)}\right) \\
&= \mathbb{E}_{Y|\vec{X}=\vec{x}}\left(e^{-Y m_{t-1}^{\text{joint}}(\vec{x})} e^{-Y m_t(\vec{x})}\right) \;=\; \mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}\left(e^{-Y m_t(\vec{x})}\right) \\
&= e^{-m_t(\vec{x})}\mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}\big(\mathbb{1}(Y=+1)\big) + e^{+m_t(\vec{x})}\mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}\big(\mathbb{1}(Y=-1)\big).
\end{aligned}$$

- A necessary condition for a minimum is that the derivative w.r.t. $m_t(\vec{x})$ vanishes. Using the usual transformations, this leads to

$$m_t(\vec{x}) \;=\; \frac{1}{2}\ln\left(\frac{P_w(Y=+1 \mid \vec{X}=\vec{x})}{P_w(Y=-1 \mid \vec{X}=\vec{x})}\right).$$

- The weight update is

$$w(\vec{x},y) \;\leftarrow\; w(\vec{x},y)e^{-Y m_t(\vec{x})}.$$

AdaBoost as Additive Modeling

- Without a weight α_t, the best model to add can be found without any assumption about the values that $m_t(\vec{x})$ produces, in particular, we no longer need the assumption $m_t(\vec{x}) \in \{-1, +1\}$.

  It may be $m_t(\vec{x}) \in \mathbb{R}$, that is **continuous AdaBoost** (also: **real AdaBoost**).

- Also, the distribution-based version we considered here would terminate after a single step, since the optimal model would be found immediately.

- However, in practice we do not have access to the underlying distribution. Rather, the weighted conditional distribution is estimated from a finite data set. As a consequence, multiple iterations are needed to obtain a final model.

- It can be shown that at the optimal m^{joint} it is [Friedman *et al.* 2000]

$$\mathbb{E}_{Y|\vec{X}=\vec{x}}(e^{-Ym^{\text{joint}}(\vec{x})} \cdot Y) \;=\; \mathbb{E}_{Y|\vec{X}=\vec{x}}(w(\vec{x}, Y) \cdot Y) \;=\; \mathbb{E}^{(w)}_{Y|\vec{X}=\vec{x}}(Y) \;=\; 0.$$

- This supports the view that the weights are analogous to residuals in regression. As long as the mean of the residuals does not vanish, m^{joint} can be improved.

Gradient Boosting: General Idea

- For given training data $\mathbf{D}$, the loss incurred by using model m to predict y is

$$\mathcal{L}(m, \mathbf{D}) \;=\; \sum_{(\vec{x},y)\in\mathbf{D}} \mathcal{L}(m(\vec{x}), y).$$

- Minimizing this loss can be viewed as analogous to numerical optimization of the vector $\vec{m} = (m(\vec{x}))^{\top}_{(\vec{x},y)\in\mathbf{D}} \in \mathbb{R}^{|\mathbf{D}|}$ of predictions for the data points as

$$\hat{\vec{m}} \;=\; \operatorname*{argmin}_{\vec{m}\in\mathbb{R}^n} \mathcal{L}(\vec{m}, \vec{y}) \;=\; \operatorname*{argmin}_{\vec{m}\in\mathbb{R}^n} \sum_{(\vec{x},y)\in\mathbf{D}} \mathcal{L}(m(\vec{x}), y).$$

- In numerical optimization $\hat{\vec{m}}$ would be found as a sum of component vectors

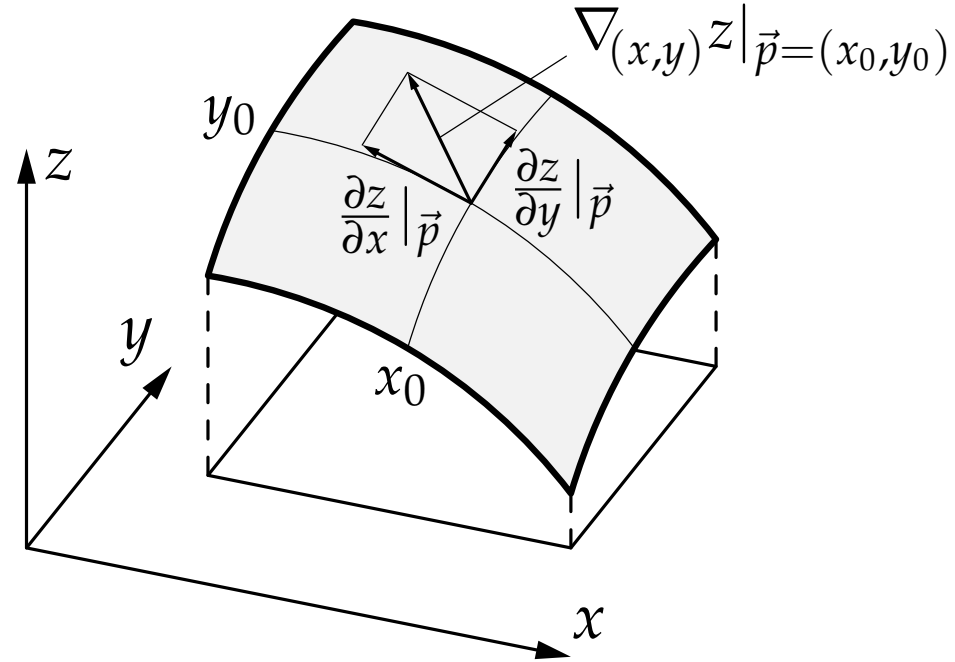
$$\vec{m}^{\text{joint}}_{\tau} = \vec{m}_0 + \sum_{t=1}^{\tau} \vec{m}_t \qquad \text{with} \qquad \vec{m}_t \in \mathbb{R}^{|\mathbf{D}|},$$

  where $\vec{m}_0$ is an initial guess and each increment $\vec{m}_t$, $t \in \{1, \ldots, \tau\}$, is induced based on the sum of the preceding $t-1$ components.

- Different optimization methods differ in how the next $\vec{m}_t$ is computed.

Reminder: Gradient Methods

- **Presupposition:** $\Omega \subseteq \mathbb{R}^n$, $f : \Omega \to \mathbb{R}$ is differentiable
- **Gradient:** differential operator that produces a vector field.



- At each point in the domain of a (scalar) function the gradient operator yields a vector in the direction of of the steepest ascent of that function.
- The length of this vector is a measure of the steepness (slope) at that point.

- Illustration of the gradient of a function $z = f(x, y)$ at a point (x_0, y_0).
  It is $\nabla_{(x,y)} z|_{(x_0,y_0)} = \left(\frac{\partial z}{\partial x}(x_0, y_0), \frac{\partial z}{\partial y}(x_0, y_0)\right)$.
  (∇ is pronounced "nabla" or "del".)

Gradient Methods: Cookbook Recipe

Idea: Starting from a randomly chosen point in the search space, make small steps in the search space, always in the direction of the steepest ascent (or descent) of the function to optimize, until a (local) maximum (or minimum) is reached.

1. Choose a (random) starting point $\vec{x}^{(0)} = \left(x_1^{(0)}, \ldots, x_m^{(0)}\right)$
2. Compute the gradient at the current point $\vec{x}^{(t-1)}$:

$$\nabla_{\vec{x}} f(\vec{x}^{(t-1)}) = \left(\frac{\partial f}{\partial x_1}(\vec{x}^{(t-1)}), \ldots, \frac{\partial f}{\partial x_m}(\vec{x}^{(t-1)})\right)$$

3. Make a small step in the direction (or against the direction) of the gradient:

$$\vec{x}^{(t)} = \vec{x}^{(t-1)} \pm \eta \, \nabla_{\vec{x}} f(\vec{x}^{(t-1)}).$$

$+$: gradient ascent
$-$: gradient descent

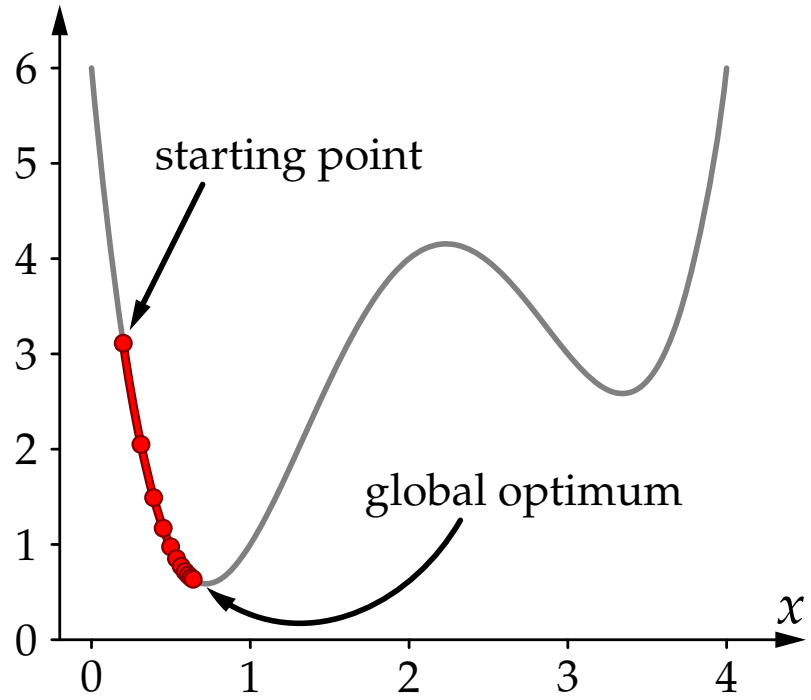
η is a step width parameter ("learning rate" in artificial neural networks).

4. Repeat steps 2 and 3, until some termination criterion is satisfied (e.g., a certain number of steps has been executed, current gradient is small).

Gradient Descent: Simple Example

Example function: $$f(u) = \frac{5}{6}u^4 - 7u^3 + \frac{115}{6}u^2 - 18u + 6$$

i	u_i	$f(u_i)$	$f'(u_i)$	Δu_i
0	0.200	3.112	−11.147	0.111
1	0.311	2.050	−7.999	0.080
2	0.391	1.491	−6.015	0.060
3	0.451	1.171	−4.667	0.047
4	0.498	0.976	−3.704	0.037
5	0.535	0.852	−2.990	0.030
6	0.565	0.771	−2.444	0.024
7	0.589	0.716	−2.019	0.020
8	0.610	0.679	−1.681	0.017
9	0.626	0.653	−1.409	0.014
10	0.640	0.635		



Gradient descent with initial value $u_0 = 0.2$, step width/learning rate $\eta = 0.01$.

Due to a proper step width/learning rate, the minimum is approached quickly.

Gradient Methods with Line Search

Idea: Choose the step width with a line search in the gradient direction.

1. Choose a (random) starting point $\vec{x}^{(0)} = \left(x_1^{(0)}, \ldots, x_m^{(0)}\right)$
2. Compute the gradient at the current point $\vec{x}^{(t-1)}$:

$$\nabla_{\vec{x}} f(\vec{x}^{(t-1)}) = \left(\frac{\partial f}{\partial x_1}(\vec{x}^{(t-1)}), \ldots, \frac{\partial f}{\partial x_m}(\vec{x}^{(t-1)})\right)$$

3. Find the optimum in the direction of the gradient (line search):

$$\beta^{(t)} = \underset{\beta \in \mathbb{R}}{\text{argopt}} \; \vec{x}^{(t-1)} + \beta \, \nabla_{\vec{x}} f(\vec{x}^{(t-1)})$$

argmax : gradient ascent
argmin : gradient descent

4. Move to the directional minimum found in the line search:

$$\vec{x}^{(t)} = \vec{x}^{(t-1)} + \beta^{(t)} \, \nabla_{\vec{x}} f(\vec{x}^{(t-1)}).$$

5. Repeat steps 2 to 4, until some termination criterion is satisfied
(e.g., a certain number of steps has been executed, current gradient is small).

Gradient Boosting: General Idea

- Applying an analog of **gradient descent** (also known as **steepest descent**) to model optimization (as described above) means choosing

$$\vec{m}_t \;=\; -\beta_t \vec{g}_t,$$

where $\vec{g}_t$ is the gradient of the loss function evaluated at $\vec{m}_{t-1}^{\text{joint}}$, that is,

$$\vec{g}_t \;=\; \nabla_{\vec{m}}\, \mathcal{L}(\vec{m}_{t-1}^{\text{joint}}, \vec{y}),$$

and β_t is a scalar step width computed as the solution of

$$\beta_t \;=\; \underset{\beta \in \mathbb{R}}{\operatorname{argmin}}\, \mathcal{L}(\vec{m}_{t-1}^{\text{joint}} - \beta \vec{g}_t).$$

- This is analogous to gradient descent with line search.
- Gradient (or steepest) descent can be seen as a greedy strategy, because $-\vec{g}_t$ is the direction in which $\mathcal{L}(\vec{m}, \vec{y})$ is most rapidly decreasing at the current point $\mathcal{L}(\vec{m}_{t-1}^{\text{joint}}, \mathbf{D}) \in \mathbb{R}^{|\mathbf{D}|}$.
- For actual model optimization $-\beta_t \vec{g}_t$ has to be represented by a model m_t.

Gradient Boosting / Gradient Boosted Trees

- Since the "gradients" play a role analogous to the weights in boosting, methods based on this general approach are called **gradient boosting**.
- Reconsider that we have a model $m_{t-1}^{\text{joint}}(\vec{x})$ and seek an improved model

$$m_t^{\text{joint}}(\vec{x}) \;=\; m_{t-1}^{\text{joint}}(\vec{x}) + [\beta_t] m_t(\vec{x}).$$

  The models m_t are supposed to model the "gradient" of the loss function w.r.t. the (joint) model predictions at the training data points.

- The individual models may be tree-structured, actually **regression trees**. (Even if we are trying to solve a **classification problem**!)
  This leads to the method of **gradient boosted trees**.
- We consider two variants:
  - standard gradient boosting / gradient boosted trees (**GBM**)
    (analogous to gradient descent with line search)
  - extreme gradient boosting / gradient boosted trees (**XGBoost**)
    (analogous to the Newton–Raphson method)

Gradient Boosting: Cookbook Recipe

1. Initialize $m_0(\vec{x}) = \operatorname{argmin}_{\beta \in \mathbb{R}} \sum_{(\vec{x},y) \in \mathbf{D}} \mathcal{L}(\beta, y)$. (e.g. majority class or zero)

2. Compute the "**gradient**" of the current model m_{t-1}^{joint}, that is,

$$\vec{g}_t = \nabla_m \mathcal{L}(m_{t-1}^{\text{joint}}, \mathbf{D}) = \left(\frac{\partial \mathcal{L}(m(\vec{x}), y)}{\partial m(\vec{x})} \right)^\top_{(\vec{x},y) \in \mathbf{D}} \Bigg|_{m = m_{t-1}^{\text{joint}}}.$$

3. Fit a (regression) model m_t to the "**data points**" $\mathbf{D}_t = \{(\vec{x}, -g_{t,(\vec{x},y)})\}_{(\vec{x},y) \in \mathbf{D}}$.

4. Find a model weight/multiplier β_t by solving

$$\beta_t = \operatorname{argmin}_{\beta \in \mathbb{R}} \sum_{(\vec{x},y) \in \mathbf{D}} \mathcal{L}(m_{t-1}^{\text{joint}} + \beta m_t(\vec{x}), y).$$

5. Set $m_t^{\text{joint}}(\vec{x}) = m_{t-1}^{\text{joint}}(\vec{x}) + \beta_t m_t(\vec{x})$. (update the additive model)

6. Repeat steps 2 to 5 until some termination criterion is satisfied (e.g., certain number of steps executed, or loss sufficiently small).

Gradient Boosted Trees

- Suppose the regression models m_t are regression trees $T_{\Theta_t|\mathbf{D}_t}$.
  (This would be the case for both regression and classification, since for both $-g_{t,(\vec{x},y)} \in \mathbb{R}$.)

- Here Θ_t represents hyper-parameters used for the regression tree induction, for example, a maximum tree height or a maximum number of leaves.
  (While for standard random forests the trees are grown without restriction or pruning, gradient boosted trees are usually constrained in height or number of leaves.)

- Each regression tree m_t is induced with a squared error criterion: (cf. slide 152)

$$\text{minimize} \quad \sum_{(\vec{x},z)\in\mathbf{D}_t} (m_t(\vec{x}) - z)^2 \qquad \text{w.r.t. } m_t.$$

- Let V_{m_t} be the set of leaves of the regression tree $m_t = T_{\Theta_t|\mathbf{D}_t}$. (Note the difference to V_T on slide 137.)

  Let π_{m_t} describe the partitioning of the data space induced by $m_t = T_{\Theta_t|\mathbf{D}_t}$, that is, $\pi_{m_t}(v)$ for $v \in V_{m_t}$ is the region of the data space covered by leaf v.

$$\forall v \in V_{m_t}: \qquad \beta_v = \operatorname*{argmin}_{\beta\in\mathbb{R}} \sum_{(\vec{x},y)\in\mathbf{D},\, \vec{x}\in\pi_{m_t}(v)} \mathcal{L}(m_{t-1}^{\text{joint}}(\vec{x}) + \beta, y).$$

Gradient Boosted Trees

- Note that instead of a line search for a best β, here we have a "region search" for the best β_v in each region covered by a leaf node v of the regression tree.

  As a consequence, there is not a single "step width" parameter β, but multiple, region-specific step width parameters β_v.

- With regression trees, the model update is

$$\begin{aligned} m_t^{\text{joint}}(\vec{x}) &= m_{t-1}^{\text{joint}}(\vec{x}) + \overbrace{m_t(\vec{x})}^{} \\ &= m_{t-1}^{\text{joint}}(\vec{x}) + \overbrace{\sum_{v \in V_{m_t}} \beta_v \, \mathbb{1}(\vec{x} \in \pi_{m_t}(v))}^{}. \end{aligned}$$

- Note that the expression $m_t(\vec{x}) = \sum_{v \in V_{m_t}} \beta_v \, \mathbb{1}(\vec{x} \in \pi_{m_t}(v))$ describes the prediction of the regression tree $m_t = T_{\Theta_t | \mathbf{D}_t}$ at the point $\vec{x}$.

  For each $\vec{x}$ only one of the terms of the sum does not vanish, namely the one for the leaf node v used to make the prediction for $\vec{x}$.

- As a consequence, $m_t^{\text{joint}}(\vec{x}) = m_{t-1}^{\text{joint}}(\vec{x}) + m_t(\vec{x})$ merely means that the regression tree $m_t = T_{\Theta_t | \mathbf{D}_t}$ is added to the forest m_{t-1}^{joint}.

Gradient Boosting

- **Shrinkage:** Add a "learning rate" η to the model update:

$$m_t^{\text{joint}}(\vec{x}) \;=\; m_{t-1}^{\text{joint}}(\vec{x}) + \eta\beta_t m_t(\vec{x}) \qquad \text{with } \eta \in (0,1], \text{often } \eta < 0.1.$$

  Improves generalization at the price of increased costs (higher τ needed).

- **Multiple Classes** are treated in analogy to multinomial logistic classification. A separate tree is built for each class (discriminating against all other classes). To obtain a class probability, predictions are normalized over these trees.

Gradients of Typical Loss Functions

setting	loss $\mathcal{L}(m(\vec{x}), y)$	gradient $\partial\mathcal{L}(m(\vec{x}), y)/\partial m(\vec{x})$
regression	$\frac{1}{2}(y - m(\vec{x}))^2$	$m(\vec{x}) - y$
regression	$\lvert y - m(\vec{x})\rvert$	$\text{sign}(m(\vec{x}) - y)$
classification	exponential	$-y\exp(-y\,m(\vec{x}))$
classification	binomial deviance	$p_{\pm}(\vec{x}) - \mathbb{1}(y = \pm 1)$
classification	deviance	$p_k(\vec{x}) - \mathbb{1}(y = c_k)$

Reminder: Newton–Raphson Method for Finding Optima

- The standard Newton-Raphson method finds roots of functions.
- By **applying the Newton-Raphson method to the gradient** of a function, it may be used to **find optima** (minima, maxima, or saddle points), because a vanishing gradient is a necessary condition for an optimum.
- In this case the update formula is

$$\vec{x}_{t+1} = \vec{x}_t + \Delta\vec{x}_t, \qquad \Delta\vec{x}_t = -\left(\nabla^2_{\vec{x}} f(\vec{x}_t)\right)^{-1} \cdot \nabla_{\vec{x}} f(\vec{x}_t),$$

  where $\nabla^2_{\vec{x}} f(\vec{x}_t)$ is the so-called **Hessian matrix**, that is, the matrix of second-order partial derivatives of the scalar-valued function f.
- In one dimension:

$$x_{t+1} = x_t + \Delta x_t, \qquad \Delta x_t = -\frac{\frac{\partial f}{\partial x}(x_t)}{\frac{\partial^2 f}{\partial x^2}(x_t)}.$$

- For a quadratic function, this method finds the optimum in one step (. . . and this is what was referred to before).

Extreme Gradient Boosting (XGBoost): Cookbook Recipe

1. Initialize $m_0(\vec{x}) = \operatorname{argmin}_{\beta \in \mathbb{R}} \sum_{(\vec{x},y) \in \mathbf{D}} \mathcal{L}(\beta, y)$. (e.g. majority class or zero)

2. Compute the "**gradient**" and "**Hessian**" of the current model m_{t-1}^{joint}, that is,

$$\vec{g}_t = \nabla_m \mathcal{L}(m_{t-1}^{\text{joint}}, \mathbf{D}) = \left(\frac{\partial \mathcal{L}(m(\vec{x}), y)}{\partial m(\vec{x})} \right)^\top_{(\vec{x},y) \in \mathbf{D}} \Bigg|_{m = m_{t-1}^{\text{joint}}} \quad \text{and}$$

$$\vec{h}_t = \left(\frac{\partial^2 \mathcal{L}(m(\vec{x}), y)}{\partial m(\vec{x})^2} \right)^\top_{(\vec{x},y) \in \mathbf{D}} \Bigg|_{m = m_{t-1}^{\text{joint}}} .$$

   Note that $\vec{h}_t$ is **not** a Hessian matrix, but only a vector with the second derivatives for the individual data points (coinciding with the diagonal of the Hessian matrix).

   Why not $\mathbf{H}_t = \nabla_m^2 \mathcal{L}(m_{t-1}^{\text{joint}}, \vec{y})$ as in the multivariate Newton–Raphson method?
   $\Rightarrow$ next slide ($\mathbf{H}_t$ is actually a diagonal matrix)

3. Fit a (reg.) model m_t to the "**data points**" $\mathbf{D}_t = \left\{ \left(\vec{x}, -g_{t,(\vec{x},y)} / h_{t,(\vec{x},y)} \right) \right\}_{(\vec{x},y) \in \mathbf{D}}$.

   Note again that $\vec{h}_t$ is **not** a Hessian matrix, but only a vector of second derivatives. However, using $-\mathbf{H}_t^{-1} \vec{g}_t$ as the data values yields exactly the same result. $\Rightarrow$ next slide

Side Remark on the Hessian Matrix

- Let $f(\vec{x}) = f(x_1, \ldots, x_m)$ be a scalar function, that is, $f : \mathbb{R}^m \to \mathbb{R}$.
- The **Hessian matrix** $\mathbf{H}^{(f,\vec{x})}$ of f at a point $\vec{x}$ is

$$\mathbf{H}^{(f,\vec{x})} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\vec{x}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\vec{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m}(\vec{x}) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(\vec{x}) & \frac{\partial^2 f}{\partial x_2^2}(\vec{x}) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_m}(\vec{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_m \partial x_1}(\vec{x}) & \frac{\partial^2 f}{\partial x_m \partial x_2}(\vec{x}) & \cdots & \frac{\partial^2 f}{\partial x_m^2}(\vec{x}) \end{bmatrix}$$

or elementwise

$$\mathbf{H}_{ij}^{(f,\vec{x})} = \frac{\partial f}{\partial x_i \partial x_j}(\vec{x}).$$

- Could extreme gradient boosting be improved by using the full Hessian matrix?
- No, because the loss function is a sum over the data points;
  each term refers to just one data point, and there are no mixed terms:

$$\mathcal{L}(m, \mathbf{D}) = \sum_{(\vec{x},y) \in \mathbf{D}} \mathcal{L}(m(\vec{x}), y).$$

- All mixed derivatives vanish, the Hessian matrix is a diagonal matrix.

Extreme Gradient Boosting (XGBoost): Cookbook Recipe

3. (continued)
   Fit a model m_t to the "**data points**" $\mathbf{D}_t = \{(\vec{x}, -g_{t,(\vec{x},y)}/h_{t,(\vec{x},y)})\}_{(\vec{x},y)\in\mathbf{D}}$ by

$$m_t \;=\; \operatorname*{argmin}_{m\in\mathcal{M}} \sum_{(\vec{x},z)\in\mathbf{D}_t} \big(m(\vec{x}) - z\big)^2.$$

4. Find a model weight/multiplier β_t by solving

$$\beta_t \;=\; \operatorname*{argmin}_{\beta\in\mathbb{R}} \sum_{(\vec{x},y)\in\mathbf{D}} \mathcal{L}(m_{t-1}^{\text{joint}} + \beta m_t(\vec{x}), y).$$

5. Set $\quad m_t^{\text{joint}}(\vec{x}) \;=\; m_{t-1}^{\text{joint}}(\vec{x}) + \beta_t m_t(\vec{x}).$ (update the additive model)

6. Repeat steps 2 to 5 until some termination criterion is satisfied
   (e.g., certain number of steps executed, or loss sufficiently small).

- A second-order Taylor approximation is used in the model fitting,
  to make the connection to the Newton–Raphson method.
  (see details on next slide)

Extreme Gradient Boosting (XGBoost)

- Consider that we have a model $m_{t-1}^{\text{joint}}(\vec{x})$ and seek an improved model

$$m_t^{\text{joint}}(\vec{x}) \;=\; m_{t-1}^{\text{joint}}(\vec{x}) + m_t(\vec{x}).$$

- The loss function can be seen as a function of possible new models $m_t^{\text{joint}}(\vec{x})$:

$$f(u) \;=\; f\Big(\underbrace{m_{t-1}^{\text{joint}}(\vec{x}) + m(\vec{x})}_{u\,=\,m_t^{\text{joint}}(\vec{x})}\Big) \;=\; \mathcal{L}\big(m_{t-1}^{\text{joint}}(\vec{x}) + m(\vec{x}), y\big).$$

- A second-order Taylor expansion of f at/around the old model $m_{t-1}^{\text{joint}}(\vec{x})$ yields

$$\mathcal{L}\big(m_{t-1}^{\text{joint}}(\vec{x}) + m(\vec{x}), y\big) \;\approx\; \mathcal{L}\big(m_{t-1}^{\text{joint}}(\vec{x}), y\big) + g_{t,(\vec{x},y)}\, m(\vec{x}) + \tfrac{1}{2} h_{t,(\vec{x},y)}\, m(\vec{x})^2.$$

- At the optimum model $m_t(\vec{x})$, the derivative must vanish:

$$\tfrac{d}{dm(\vec{x})}\Big(\mathcal{L}\big(m_{t-1}^{\text{joint}}(\vec{x}), y\big) + g_{t,(\vec{x},y)}\, m(\vec{x}) + \tfrac{1}{2} h_{t,(\vec{x},y)}\, m(\vec{x})^2\Big) = g_{t,(\vec{x},y)} + h_{t,(\vec{x},y)}\, m(\vec{x}) \stackrel{!}{=} 0.$$

- Thus we get

$$m_t(\vec{x}) = -\frac{g_{t,(\vec{x},y)}}{h_{t,(\vec{x},y)}}.$$

Extreme Gradient Boosted Trees (XGBoost)

- Suppose the regression models m_t are regression trees $T_{\Theta_t|\mathbf{D}_t}$.
  (This is the case for both regression and classification, since for both $-g_{t,(\vec{x},y)}/h_{t,(\vec{x},y)} \in \mathbb{R}$.)

- Here Θ_t represents hyper-parameters used for the regression tree induction, for example, a maximum tree height or a maximum number of leaves.
  (While for standard random forests the trees are grown without restriction or pruning, gradient boosted trees are usually constrained in height or number of leaves.)

- Let V_{m_t} be the set of leaves of the regression tree $m_t = T_{\Theta_t|\mathbf{D}_t}$.
  (Note the difference to V_T on slide 137.)

  Let π_{m_t} describe the partitioning of the data space induced by $m_t = T_{\Theta_t|\mathbf{G}_t}$, that is, $\pi_{m_t}(v)$ for $v \in V_{m_t}$ is the region of the data space covered by leaf v.

$$\forall v \in V_{m_t}: \qquad \beta_v = \operatorname*{argmin}_{\beta \in \mathbb{R}} \sum_{(\vec{x},y)\in\mathbf{D},\, \vec{x}\in\pi_{m_t}(v)} \mathcal{L}(m_{t-1}^{\text{joint}}(\vec{x}) + \beta, y).$$

- Set $m_t^{\text{joint}}(\vec{x}) = m_{t-1}^{\text{joint}}(\vec{x}) + m_t(\vec{x}) = m_{t-1}^{\text{joint}}(\vec{x}) + \sum_{v\in V_{m_t}} \beta_v \mathbb{1}(\vec{x} \in \pi_{m_t}(v))$.

  (Note that $m_t(\vec{x}) = \sum_{v\in V_{m_t}} \beta_v \mathbb{1}(\vec{x} \in \pi_{m_t}(v))$ is the prediction of the regression tree $m_t = T_{\Theta_t|\mathbf{D}_t}$ at the point $\vec{x}$.)

Random Forests and Gradient Boosted Trees
Experiments / Examples

Random Forests and Gradient Boosted Trees: Data

- Experiments were conducted according to:
  Trevor Hastie, Robert Tibshirani & Jerome Friedman
  The Elements of Statistical Learning
  Springer, New York, NY, USA 2001/2009/2017

picture not available
in online version

- **Spambase (Spam Email Database)**
  - frequencies af 48 words and 6 characters
  - avg./longest/total run length of capitals
  - target: whether email is spam or not

`https://archive.ics.uci.edu/dataset/94/spambase`

- **California Housing Prices**
  - house: location, age, rooms, bedrooms
  - district: population, households, median income
  - target: median house prices derived from the 1990 census

`https://www.kaggle.com/datasets/camnugent/california-housing-prices`

Random Forests and Gradient Boosted Trees: Libraries

- Experiments can fairly easily be run with the Python library **SciKit-learn**.
- **SciKit-learn** (`sklearn`) is a free software machine learning library for the Python programming language.

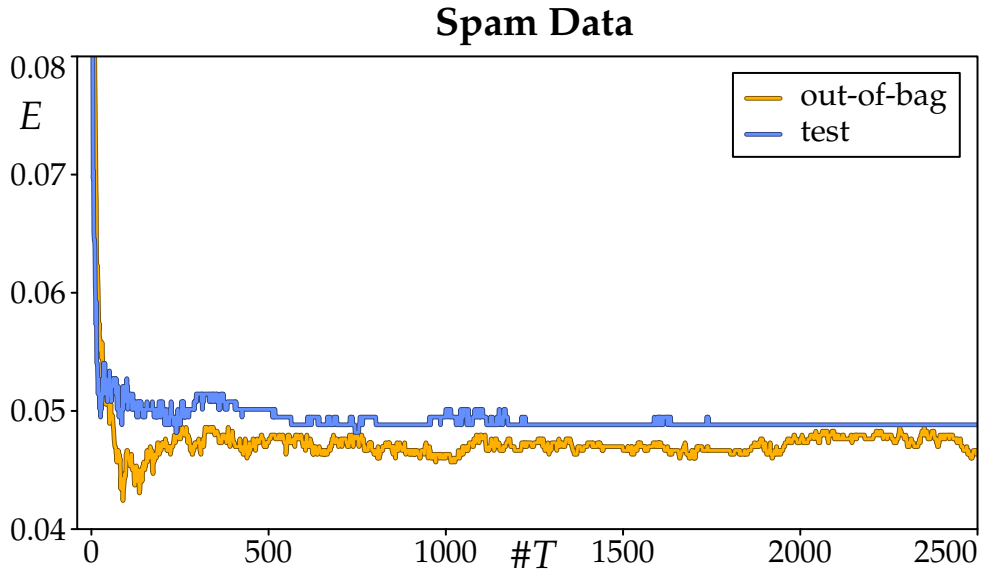
picture not available in online version

- It features various classification, regression and clustering algorithms.
- Ensemble methods are available in the sub-library `sklearn.ensemble`.
- For the experiments reported here, the following methods were used:
  - `sklearn.ensemble.BaggingClassifier`
  - `sklearn.ensemble.RandomForestClassifier`
  - `sklearn.ensemble.ExtraTreesClassifier`
  - `sklearn.ensemble.GradientBoostingClassifier`
  - `sklearn.ensemble.RandomForestRegressor`
  - `sklearn.ensemble.GradientBoostingRegressor`

Random Forests and Gradient Boosted Trees: Experiments

Spam Data

picture not available
in online version



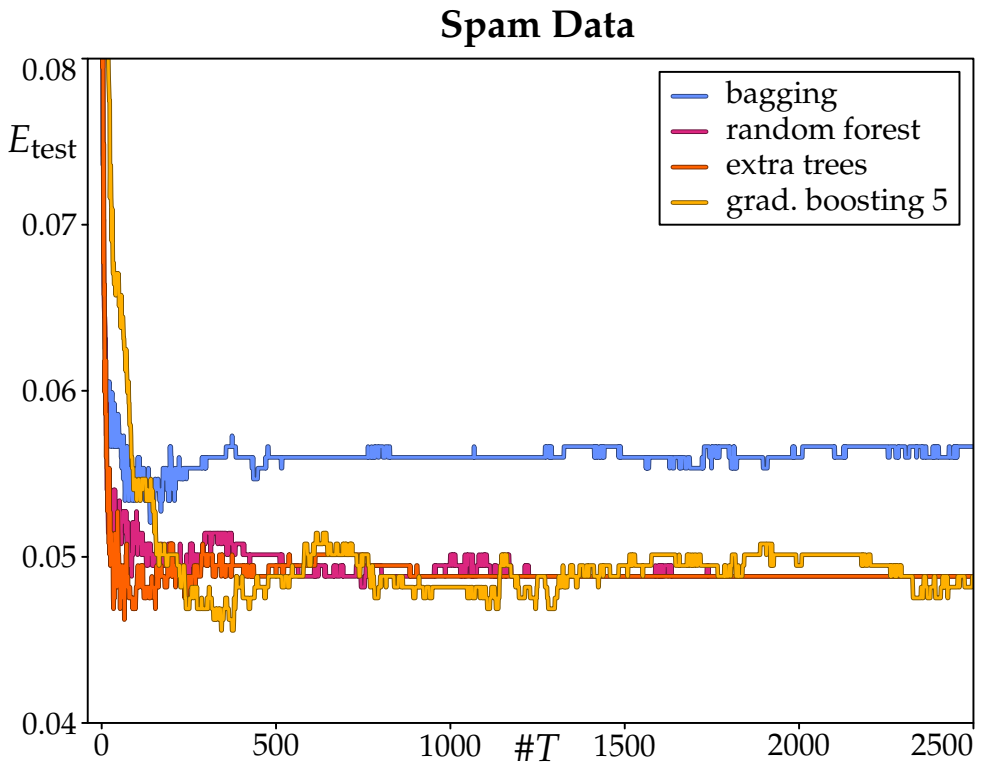
- Figure 15.4 from [Hastie *et al.* 2009]
- Both out-of-bag and test error stabilize at around 500 trees, but continue to decline.
- Out-of-bag error is always higher than test error.

- Computation with SciKit-learn.
- Both out-of-bag and test error stabilize at around 500 trees.
- Test error is always higher than out-of-bag error.

Random Forests and Gradient Boosted Trees: Experiments

Spam Data

picture not available
in online version

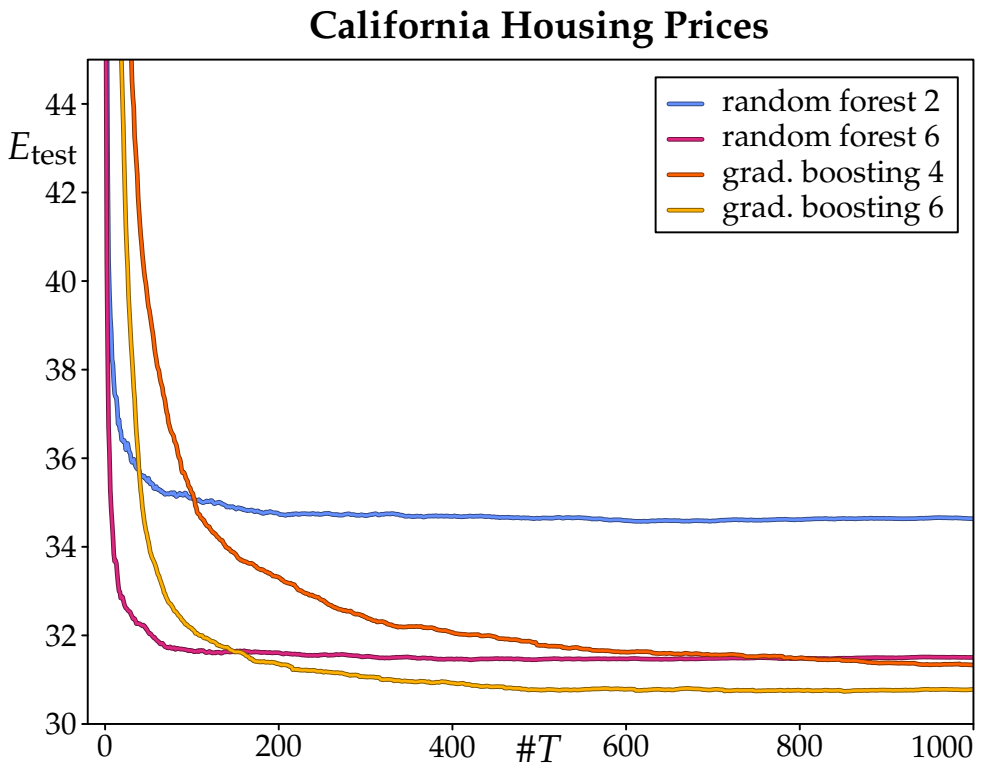


- Figure 15.1 from [Hastie *et al.* 2009]
- Gradient boosting outperforms standard random forests, which outperform bagging.
- Computation with SciKit-learn.
- Bagging is outperformed, other methods converge to same performance.

Random Forests and Gradient Boosted Trees: Experiments

California Housing Prices

picture not available
in online version



- Figure 15.3 from [Hastie *et al.* 2009]
- Gradient boosting continues to improve for > 1000 trees.
- Computation with SciKit-learn.
- Gradient boosting continues to improve only for depth 4.

Advanced Clustering

Cluster-Specific Distance Functions

The similarity of a data point to a prototype depends on their distance.

- If the cluster prototype is a simple cluster center,
  a general distance measure can be defined on the data space.

  In this case the **Euclidean distance** is most often used
  due to its rotation invariance. It leads to (hyper-)spherical clusters.

- However, more flexible clustering approaches (with size and shape parameters)
  use **cluster-specific distance functions**.

  The most common approach is to use a **Mahalanobis distance**
  with a **cluster-specific covariance matrix $\boldsymbol{\Sigma}$**.

$$d(\vec{x}, \vec{y}; \boldsymbol{\Sigma}) = \sqrt{(\vec{x} - \vec{y})^\top \boldsymbol{\Sigma}^{-1} (\vec{x} - \vec{y})}.$$

  The covariance matrix comprises **shape** and **size** parameters.

  The Euclidean distance is a special case that results for $\boldsymbol{\Sigma} = \mathbf{1}$.

Reminder: Standard k-Means Clustering

- Given: a data set $\mathbf{X} = \{x_1, \ldots, x_n\}$ of m-dimensional vectors $x_i \in \mathbb{R}^m$, in which k clusters are to be found (k to be specified by a user).
- k-means clustering starts by choosing an initial set of k centers (e.g. by sampling uniformly at random from the given data points).
- Subsequently, two update steps are executed alternatingly: [Lloyd 1957]
  - each data point is assigned to the cluster center that is closest to it,
  - cluster centers are recomputed as means of assigned data points.
- If $\nu_m(x)$ denotes the m-th closest center to a point x, this can be written

$$\forall i; 1 \leq i \leq k: \qquad c_i^{t+1} = \frac{\sum_{j=1}^{n} \mathbb{1}(\nu_1^t(x_j) = c_i^t) \cdot x_j}{\sum_{j=1}^{n} \mathbb{1}(\nu_1^t(x_j) = c_i^t)},$$

  where t denotes the time step and $\mathbb{1}(\varphi)$ yields 1 if φ if true and 0 otherwise, $\nu_1^t(x_j)$ represents the assignment step, the fraction the mean computation.
- It can be shown that this scheme must converge. [Selim & Ismail 1984]

Fuzzy Clustering

Prototype-based Clustering

Prototype-based Clustering:

Each group/cluster is described by a **prototype** comprising information about

- the **location** of the group/cluster (its "center"),
- the **size** of the group/cluster (its "reference radius"),
- the **shape** (and orientation) of the group/cluster, and
- the (relative) **weight** the group/cluster.

The (degree of) membership of a data point to a given group/cluster
or the probability that it belongs to this group/cluster depends on

- the **distance** of the data point to the prototype
  (taking into account the prototype properties) and
- a **similarity** function or **probability density** function.

The latter is usually specified as a **radial function** on the distance.
It is the same for all clusters (they differ only by the prototype properties).

Fuzzy Clustering

- Allow degrees of membership of a datum to different clusters.
  (Classical k-/c-means clustering assigns data crisply.)

Objective Function: (to be minimized)

$$\mathcal{J}(\mathbf{X}, \mathbf{B}, \mathbf{U}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} d^2(\beta_i, \vec{x}_j)$$

- $\mathbf{U} = [u_{ij}]$ is the $c \times n$ fuzzy partition matrix,

  $u_{ij} \in [0, 1]$ is the membership degree of the data point $\vec{x}_j$ to the i-th cluster.
- $\mathbf{B} = \{\beta_1, \ldots, \beta_c\}$ is the set of cluster prototypes.
- w is the so-called "fuzzifier" (the higher w, the softer the cluster boundaries).
- Constraints:

  $$\forall i \in \{1, \ldots, c\}: \quad \sum_{j=1}^{n} u_{ij} > 0 \qquad \text{and} \qquad \forall j \in \{1, \ldots, n\}: \quad \sum_{i=1}^{c} u_{ij} = 1.$$

Relation to Classical c-Means Clustering:

- Classical c-means clustering can be seen as optimizing the objective function

$$\mathcal{J}(\mathbf{X}, \mathbf{B}, \mathbf{U}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}\, d^2(\beta_i, \vec{x}_j),$$

  where $\forall i, j : u_{ij} \in \{0, 1\}$ (i.e. hard assignment of the data points) and the cluster prototypes β_i consist only of cluster centers.

- To obtain a fuzzy assignment of the data points, it is **not** enough to extend the range of values for the u_{ij} to the unit interval $[0, 1]$: The objective function $\mathcal{J}$ is optimized for a hard assignment (each data point is assigned to the closest cluster center).

- **To achieve actual degrees of membership:** (this is not the only method)
  Apply a convex function $h : [0, 1] \to [0, 1]$ to the membership degrees u_{ij}.
  Most common choice: $h(u) = u^w$, usually with $w = 2$.

Reminder: Function Optimization

Task: Find values $\vec{x} = (x_1, \ldots, x_m)$ such that $f(\vec{x}) = f(x_1, \ldots, x_m)$ is optimal.

Often feasible approach:

- A necessary condition for a (local) optimum (minimum/maximum) is that the partial derivatives w.r.t. the parameters vanish [Pierre de Fermat, 1607–1665].
- Therefore: (Try to) Solve the equation system that results from setting all partial derivatives w.r.t. the parameters equal to zero.

Example task: Minimize $f(x, y) = x^2 + y^2 + xy - 4x - 5y$.

Solution procedure:

1. Take the partial derivatives of the objective function and set them equal to zero:
$$\frac{\partial f}{\partial x} = 2x + y - 4 = 0, \qquad \frac{\partial f}{\partial y} = 2y + x - 5 = 0.$$
2. Solve the resulting (here: linear) equation system: $x = 1, \quad y = 2.$

Function Optimization with Constraints

Often a function has to be optimized subject to certain **constraints**.

First: restriction to k **equality constraints** $C_i(\vec{x}) = 0, i = 1, \ldots, k$.

Note: the equality constraints describe a subspace of the domain of the function.

Problem of optimization with constraints:

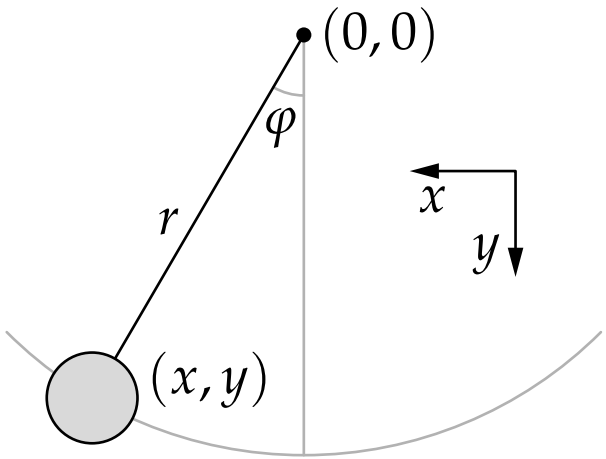
- The gradient of the objective function f may vanish outside the constrained subspace, leading to an unacceptable solution (violating the constraints).
- At an optimum *in the constrained subspace* the derivatives need not vanish.

One way to handle this problem are **generalized coordinates**:

- Exploit the dependence between the parameters specified in the constraints to express some parameters in terms of the others and thus reduce the set $\vec{x}$ to a set $\vec{x}'$ of independent parameters (*generalized coordinates*).
- Problem: Can be clumsy and cumbersome, if possible at all, because the form of the constraints may not allow for expressing some parameters as proper functions of the others.

Generalized Coordinates: Example

- Consider a simple **stick pendulum**, that is, a mass at the end of a stick suspended from a hub at the other end (like in the diagram on the right).



- Let the location of the mass be described by **Cartesian coordinates** x and y (coordinate origin is at the hub).
- Any optimization problem involving the location (x, y) of the mass is constrained by $x^2 + y^2 = r^2$ (because the stick is rigid, its length is constant). $\Rightarrow y = \sqrt{r^2 - x^2}$
- This constraint may also be eliminated by switching to **polar coordinates**:

$$r = \text{hypot}(x, y) = \sqrt{x^2 + y^2} \qquad \text{and} \qquad \varphi = \text{atan2}(x, y).$$

- Because r is constant (since the stick length is fixed), one only needs to consider the angle φ (which may be varied freely). Hence the angle φ may be used as a **generalized coordinate**.

Function Optimization with Constraints

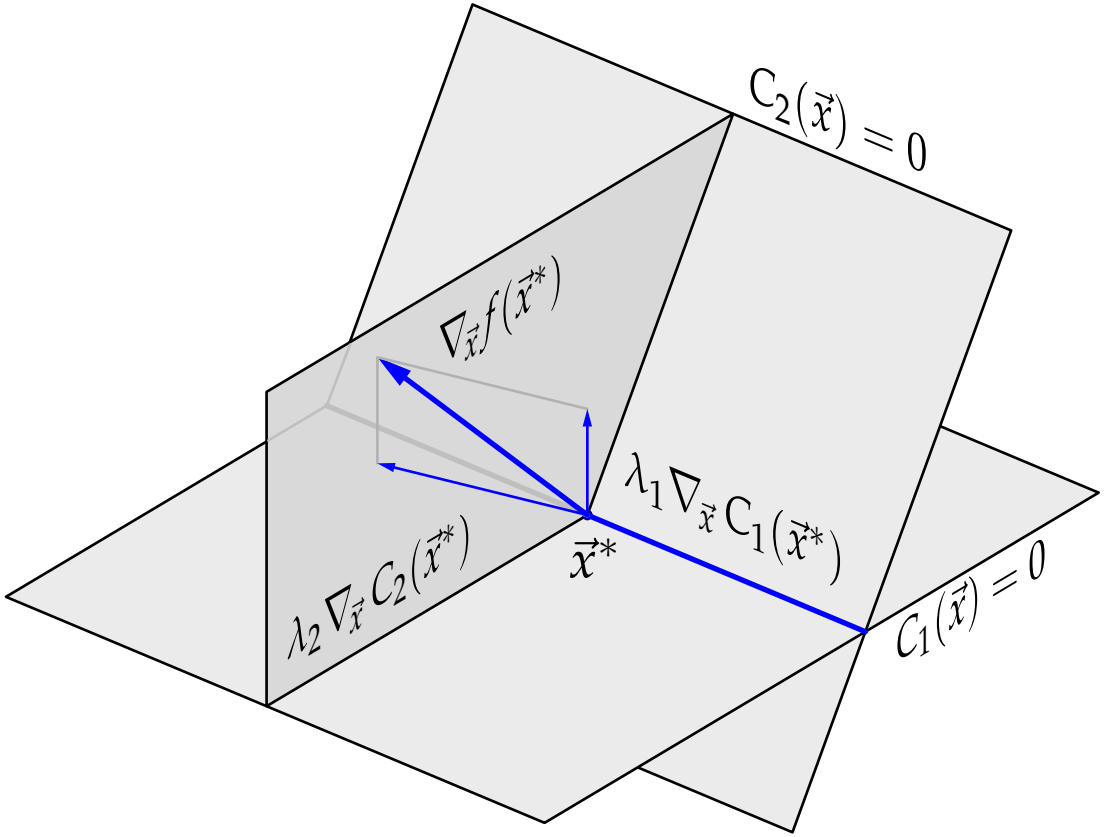
A much more elegant approach is based on the following nice insights:
Let $\vec{x}^*$ be a (local) optimum of $f(\vec{x})$ *in the constrained subspace*. Then:

- The gradient $\nabla_{\vec{x}} f(\vec{x}^*)$, if it does not vanish, must be **perpendicular** to the constrained subspace. (If $\nabla_{\vec{x}} f(\vec{x}^*)$ had a component in the constrained subspace, $\vec{x}^*$ would not be a (local) optimum in this subspace.)
- The gradients $\nabla_{\vec{x}} C_j(\vec{x}^*)$, $1 \leq j \leq k$, must all be **perpendicular** to the constrained subspace, because they are constant, namely 0, in this subspace. Together they span the subspace perpendicular to the constrained subspace.
- Therefore it must be possible to find values λ_j, $1 \leq j \leq k$, such that

$$\nabla_{\vec{x}} f(\vec{x}^*) - \sum_{j=1}^{k} \lambda_j \nabla_{\vec{x}} C_j(\vec{x}^*) = \vec{0}.$$

  If the constraints (and thus their gradients) are linearly independent, the values λ_j are uniquely determined. This equation can be used to **compensate the gradient** of $f(\vec{x}^*)$ so that it vanishes at $\vec{x}^*$.

Function Optimization with Constraints



$$\nabla_{\vec{x}} f(\vec{x}^*) - \lambda_1 \nabla_{\vec{x}} C_1(\vec{x}^*) - \lambda_2 \nabla_{\vec{x}} C_2(\vec{x}^*) = \vec{0}$$

- The gradient $\nabla_{\vec{x}} f(\vec{x}^*)$ can be compensated with the gradients $\nabla_{\vec{x}} C_1(\vec{x}^*)$ and $\nabla_{\vec{x}} C_2(\vec{x}^*)$ using suitable factors λ_1 and λ_2.
- These factors are called **Lagrange multipliers**.
- Due to these factors, the gradient of an enhanced function vanishes at the (local) optimum $\vec{x}^*$ in the constrained subspace.

- The planes $C_1(\vec{x}) = 0$ and $C_2(\vec{x}) = 0$ represent two (here: linear) constraints.
- Their intersection (both conditions hold, blue line) is the constrained subspace.

Function Optimization: Lagrange Theory

As a consequence of these insights we obtain the

Method of Lagrange Multipliers: [Joseph-Louis Lagrange, 1736–1813]

Given:
- a function $f(\vec{x})$, which is to be optimized,
- k equality constraints $C_j(\vec{x}) = 0, 1 \leq j \leq k$.

Procedure:

1. Construct the so-called **Lagrange function** by incorporating the equality constraints $C_i, i = 1, \ldots, k$, with (unknown) **Lagrange multipliers** λ_i:

$$\mathcal{L}(\vec{x}, \lambda_1, \ldots, \lambda_k) = f(\vec{x}) - \sum_{i=1}^{k} \lambda_i C_i(\vec{x}).$$

2. Set the partial derivatives of the Lagrange function equal to zero:

$$\frac{\partial \mathcal{L}}{\partial x_1} = 0, \quad \ldots, \quad \frac{\partial \mathcal{L}}{\partial x_m} = 0, \qquad \frac{\partial \mathcal{L}}{\partial \lambda_1} = 0, \quad \ldots, \quad \frac{\partial \mathcal{L}}{\partial \lambda_k} = 0.$$

3. (Try to) Solve the resulting equation system.

Function Optimization: Lagrange Theory

Observations:

- Due to the representation of the gradient of $f(\vec{x})$ at a local optimum $\vec{x}^*$ in the constrained subspace (see above) the gradient of $\mathcal{L}$ w.r.t. $\vec{x}$ vanishes at $\vec{x}^*$.
  $\rightarrow$ The standard approach works again!
- If the constraints are satisfied, the additional terms have no influence.
  $\rightarrow$ The original task is not modified (same objective function).
- Taking the partial derivative w.r.t. a Lagrange multiplier reproduces the corresponding equality constraint (though negated):

$$\forall j; 1 \leq j \leq k: \qquad \frac{\partial}{\partial \lambda_j} \mathcal{L}(\vec{x}, \lambda_1, \ldots, \lambda_k) \;=\; -C_j(\vec{x}) \;\stackrel{!}{=}\; 0,$$

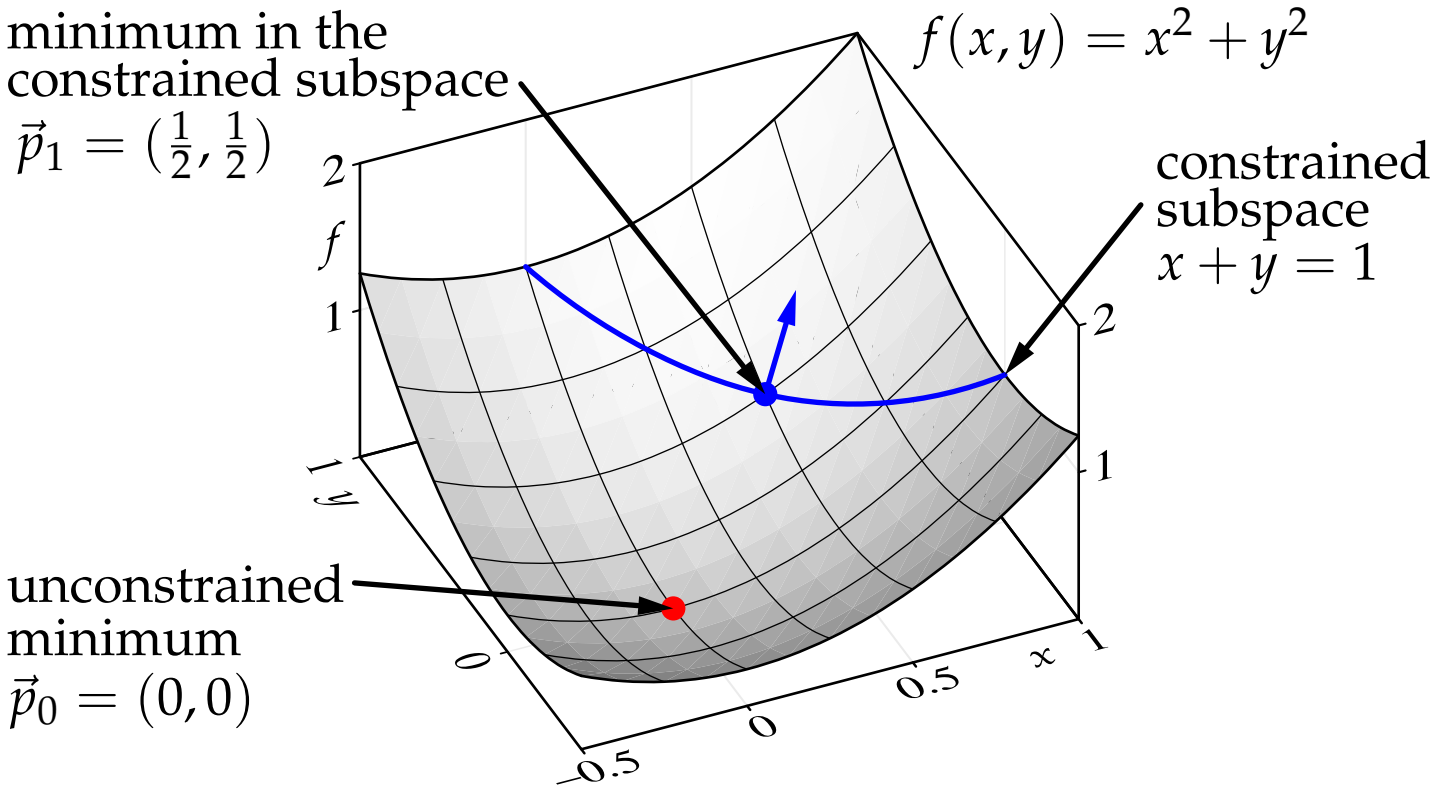
  $\rightarrow$ Constraints enter the equation system to solve in a natural way.

Remark: Up to now we considered only equality contraints.

- **Inequality constraints** can be handled with **Karush–Kuhn–Tucker theory**.

Lagrange Theory: Example 1

Example task: Minimize $f(x,y) = x^2 + y^2$ subject to $x + y = 1$.



The unconstrained minimum is not in the constrained subspace.
At the minimum in the constrained subspace the gradient of f does not vanish.

Lagrange Theory: Example 1

Example task: Minimize $f(x, y) = x^2 + y^2$ subject to $x + y = 1$.

Solution procedure:

1. Rewrite the constraint, so that one side becomes zero: $x + y - 1 = 0$.
2. Construct the Lagrange function by incorporating the constraint into the objective function with a Lagrange multiplier λ:

$$\mathcal{L}(x, y, \lambda) = x^2 + y^2 - \lambda(x + y - 1).$$

3. Take the partial derivatives of the Lagrange function and set them to zero (necessary conditions for a minimum):

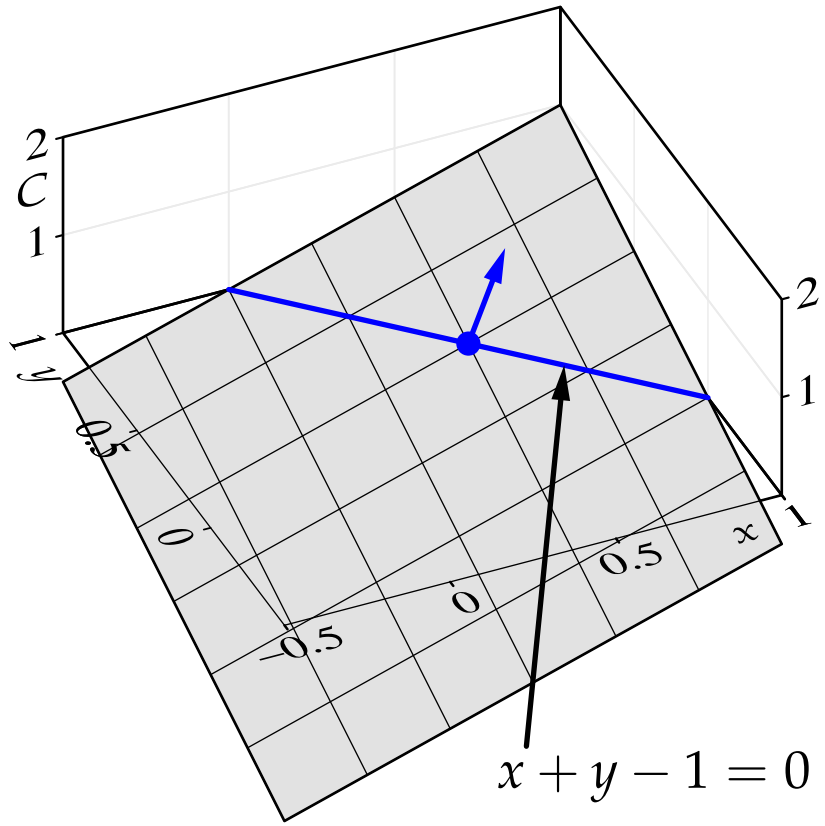
$$\frac{\partial \mathcal{L}}{\partial x} = 2x - \lambda \stackrel{!}{=} 0, \qquad \frac{\partial \mathcal{L}}{\partial y} = 2y - \lambda \stackrel{!}{=} 0, \qquad \frac{\partial \mathcal{L}}{\partial \lambda} = -(x + y - 1) \stackrel{!}{=} 0.$$

4. Solve the resulting (here: linear) equation system:

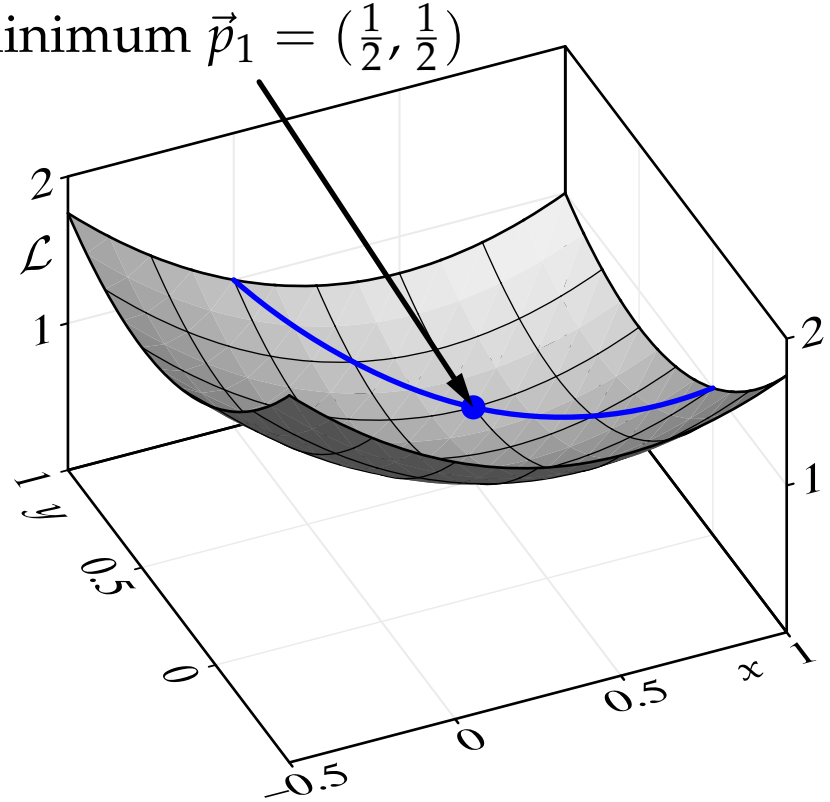
$$\lambda = 1, \qquad x = y = \tfrac{1}{2}.$$

Lagrange Theory: Example 1

$C(x,y) = x + y - 1$

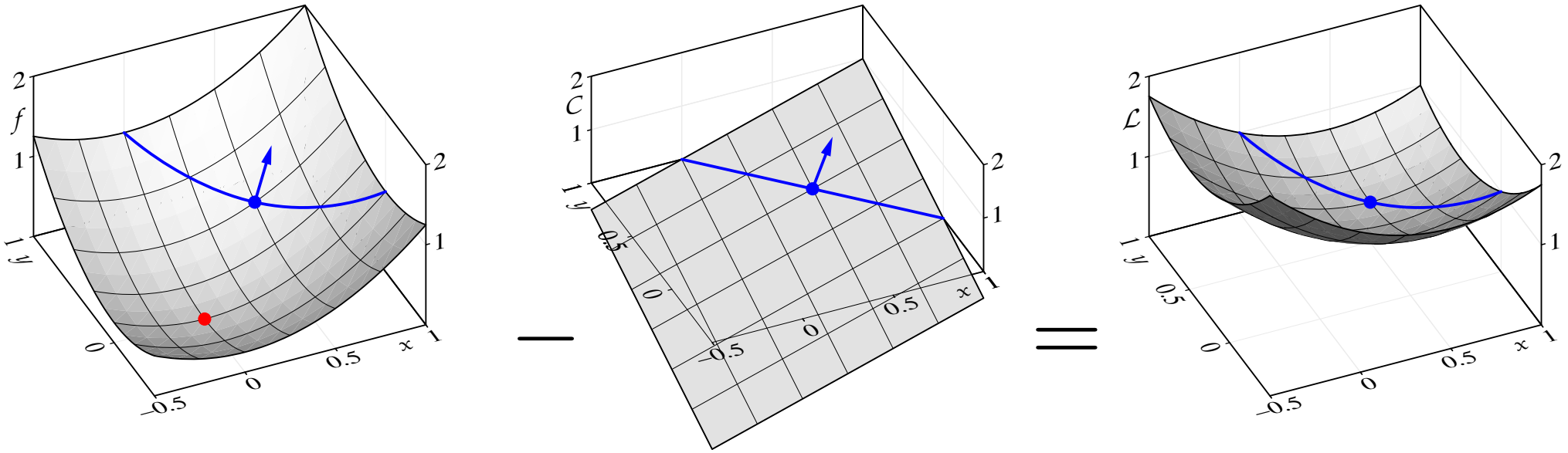


$\mathcal{L}(x,y,1) = x^2 + y^2 - (x + y - 1)$



The gradient of the constraint is perpendicular to the constrained subspace.
The (unconstrained) minimum of the Lagrange function $\mathcal{L}(x,y,\lambda)$
is the minimum of the objective function $f(x,y)$ in the constrained subspace.

Lagrange Theory: Example 1



$f(x,y) = x^2 + y^2$ $\quad C(x,y) = x + y - 1$ $\quad \mathcal{L}(x,y,1) = x^2 + y^2 - (x + y - 1)$

The gradient of the function f is perpendicular to the constrained subspace.
The gradient of the constraint C is perpendicular to the constrained subspace.
With a proper Lagrange multiplier λ, the gradient of the function f
can be compensated by the gradient of the constraint C.
The (unconstrained) minimum of the Lagrange function $\mathcal{L}(x,y,\lambda)$
is the minimum of the objective function $f(x,y)$ in the constrained subspace.

Lagrange Theory: Example 2

Example task: Find the side lengths x, y, z of a box with maximum volume for a given area S of the surface.

Formally: Maximize $f(x, y, z) = xyz$
subject to $2xy + 2xz + 2yz = S$.

Solution procedure:

1. The constraint is $C(x, y, z) = 2xy + 2xz + 2yz - S = 0$.
2. The Lagrange function is
$$\mathcal{L}(x, y, z, \lambda) = xyz - \lambda(2xy + 2xz + 2yz - S).$$
3. Taking the partial derivatives yields (in addition to the constraint):
$$\frac{\partial \mathcal{L}}{\partial x} = yz - 2\lambda(y+z) \stackrel{!}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial y} = xz - 2\lambda(x+z) \stackrel{!}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial z} = xy - 2\lambda(x+y) \stackrel{!}{=} 0.$$
4. The solution is: $\lambda = \frac{1}{4}\sqrt{\frac{S}{6}}, \quad x = y = z = \sqrt{\frac{S}{6}}$ (i.e., the box is a cube).

Fuzzy Clustering: Alternating Optimization

Objective function: (to be minimized)

$$\mathcal{J}(\mathbf{X}, \mathbf{B}, \mathbf{U}) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} d^2(\vec{x}_j, \beta_i)$$

Constraints:

$$\forall i \in \{1, \ldots, c\}: \quad \sum_{j=1}^{n} u_{ij} > 0 \qquad \text{and} \qquad \forall j \in \{1, \ldots, n\}: \quad \sum_{i=1}^{c} u_{ij} = 1.$$

- **Problem:** The objective function $\mathcal{J}$ cannot be minimized directly.
- Therefore: **Alternating Optimization**
  - Optimize membership degrees for fixed cluster parameters.
  - Optimize cluster parameters for fixed membership degrees.
    (Update formulae are derived by differentiating the objective function $\mathcal{J}$.)
  - Iterate until convergence (checked, e.g., by change of cluster center).

Fuzzy Clustering: Alternating Optimization

First Step: Fix the cluster parameters.

Introduce Lagrange multipliers λ_j, $0 \leq j \leq n$, to incorporate the constraints $\forall j; 1 \leq j \leq n : \sum_{i=1}^{c} u_{ij} = 1$. This yields the Lagrange function (to be minimized)

$$\mathcal{L}(\mathbf{X}, \mathbf{B}, \mathbf{U}, \Lambda) = \underbrace{\sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{w} \, d_{ij}^2}_{=\mathcal{J}(\mathbf{X},\mathbf{B},\mathbf{U})} - \sum_{j=1}^{n} \lambda_j \left(\sum_{i=1}^{c} u_{ij} - 1 \right),$$

A necessary condition for a minimum is that the partial derivatives of the Lagrange function w.r.t. the membership degrees vanish, i.e.,

$$\frac{\partial}{\partial u_{kl}} \mathcal{L}(\mathbf{X}, \mathbf{B}, \mathbf{U}, \Lambda) = w \, u_{kl}^{w-1} \, d_{kl}^2 - \lambda_l \stackrel{!}{=} 0,$$

which leads to

$$\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n : \qquad u_{ij} = \left(\frac{\lambda_j}{w \, d_{ij}^2} \right)^{\frac{1}{w-1}}.$$

Fuzzy Clustering: Alternating Optimization

Summing these equations over the clusters (in order to be able to exploit the corresponding constraints on the membership degrees), we get

$$1 = \sum_{i=1}^{c} u_{ij} = \sum_{i=1}^{c} \left(\frac{\lambda_j}{w\, d_{ij}^2} \right)^{\frac{1}{w-1}}.$$

Consequently the λ_j, $1 \leq j \leq n$, are

$$\lambda_j = \left(\sum_{i=1}^{c} \left(w\, d_{ij}^2 \right)^{\frac{1}{1-w}} \right)^{1-w}.$$

Inserting this into the equation for the membership degrees yields

$$\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n : \qquad u_{ij} = \frac{d_{ij}^{\frac{2}{1-w}}}{\sum_{k=1}^{c} d_{kj}^{\frac{2}{1-w}}}.$$

This update formula results regardless of the distance measure.

Fuzzy C-Means Algorithm: Euclidean distance

$$d^2_{\text{fcm}}(\vec{x}_j, \beta_i) = (\vec{x}_j - \vec{\mu}_i)^\top (\vec{x}_j - \vec{\mu}_i)$$

Necessary condition for a minimum: gradients w.r.t. cluster centers vanish.

$$\begin{aligned}
\nabla_{\vec{\mu}_k} \mathcal{J}_{\text{fcm}}(\mathbf{X}, \mathbf{B}, \mathbf{U}) &= \nabla_{\vec{\mu}_k} \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^w \, (\vec{x}_j - \vec{\mu}_i)^\top (\vec{x}_j - \vec{\mu}_i) \\
&= \sum_{j=1}^{n} u_{kj}^w \, \nabla_{\vec{\mu}_k} (\vec{x}_j - \vec{\mu}_k)^\top (\vec{x}_j - \vec{\mu}_k) \\
&= -2 \sum_{j=1}^{n} u_{kj}^w \, (\vec{x}_j - \vec{\mu}_k) \stackrel{!}{=} \vec{0}
\end{aligned}$$

Resulting update rule for the cluster centers (**second step** of alt. optimization):

$$\forall i; 1 \leq i \leq c: \qquad \vec{\mu}_i = \frac{\sum_{j=1}^{n} u_{ij}^w \vec{x}_j}{\sum_{j=1}^{n} u_{ij}^w}$$

Standard Fuzzy Clustering Algorithms

Gustafson–Kessel Algorithm: cluster-specific Mahalanobis distance

$$d_{\mathrm{gk}}^2(\vec{x}_j, \beta_i) = (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{C}_i^{-1} (\vec{x}_j - \vec{\mu}_i)$$

Additional constraints: $|\mathbf{C}_i| = 1$ (all cluster have unit size).
These constraints are incorporated again by Lagrange multipliers.

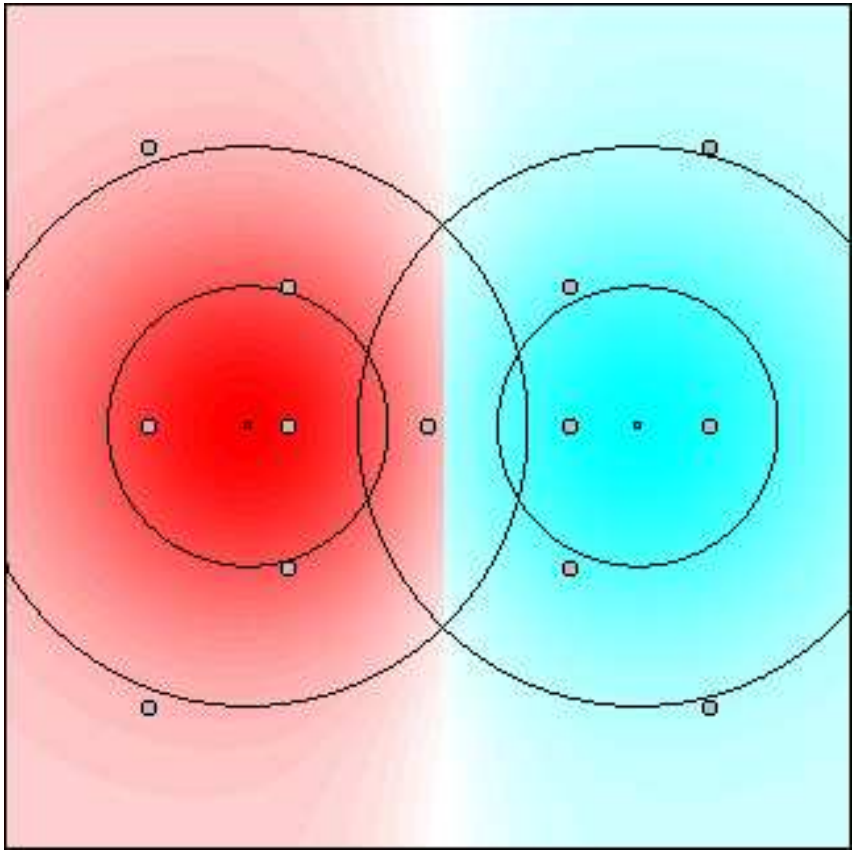
A similar derivation as for the fuzzy c-means algorithm
yields the same update rule for the cluster centers:

$$\forall i; 1 \leq i \leq c: \qquad \vec{\mu}_i = \frac{\sum_{j=1}^n u_{ij}^w \vec{x}_j}{\sum_{j=1}^n u_{ij}^w}$$

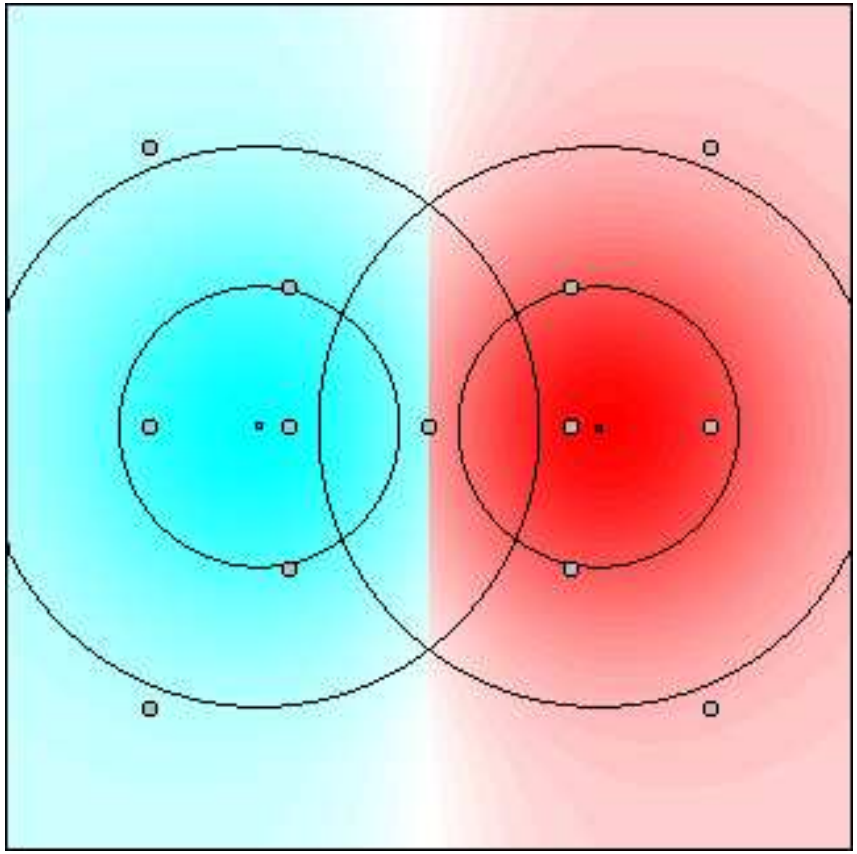
Update rule for the covariance matrices (m is the number of dimensions):

$$\mathbf{C}_i = \frac{1}{\sqrt[m]{|\mathbf{\Sigma}_i|}} \mathbf{\Sigma}_i \qquad \text{where} \qquad \mathbf{\Sigma}_i = \sum_{j=1}^n u_{ij}^w (\vec{x}_j - \vec{\mu}_i)(\vec{x}_j - \vec{\mu}_i)^\top.$$

Fuzzy Clustering: Overlapping Clusters



Classical *c*-Means



Fuzzy *c*-Means

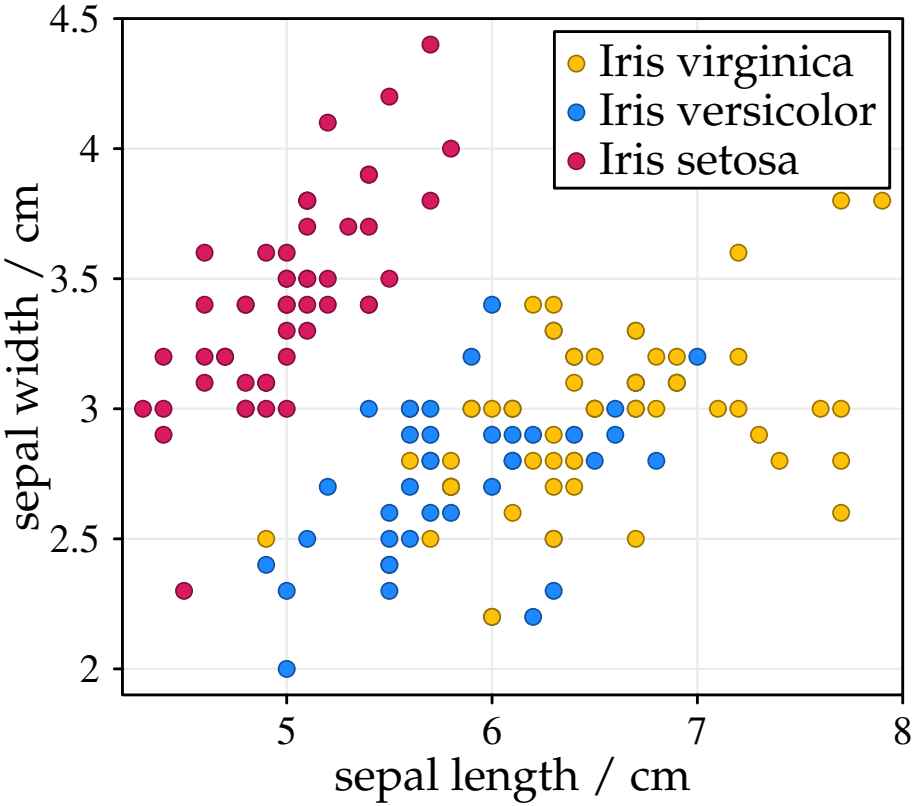
So-called “butterfly” data set.

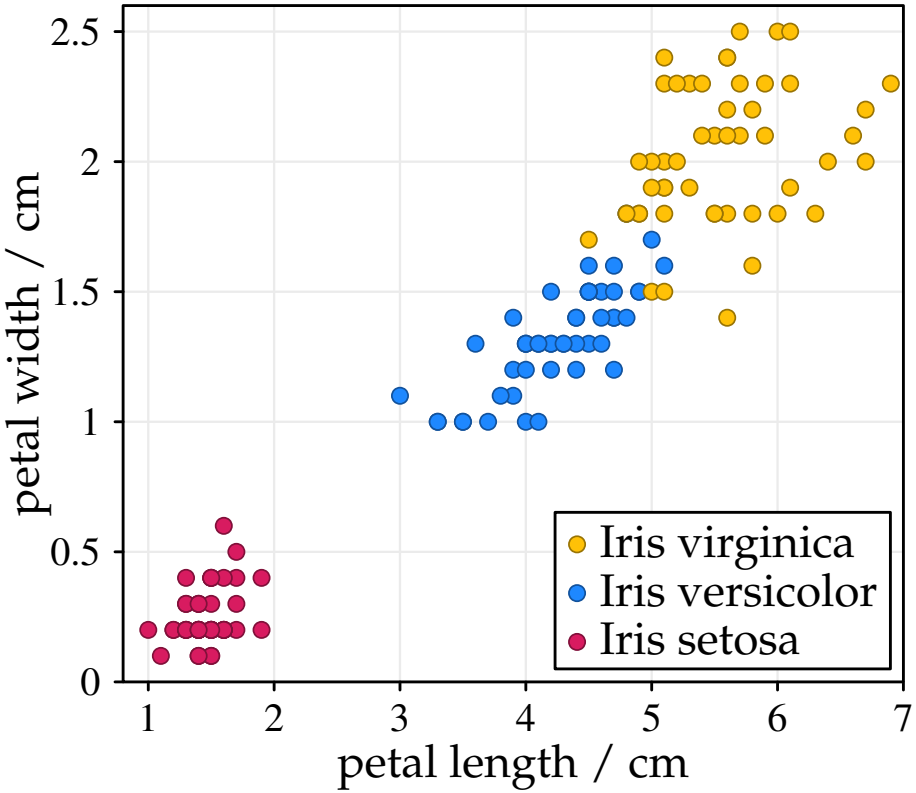
Reminder: The Iris Data

pictures not available in online version

- Collected by Edgar Anderson on the Gaspé Peninsula (Canada).
- First data-analyzed by Ronald Aylmer Fisher (famous statistician).
- 150 cases in total, 50 cases per Iris flower type (specie).
- Measurements of sepal length and width and petal length and width (in cm).
- Most famous data set in pattern recognition and data analysis.

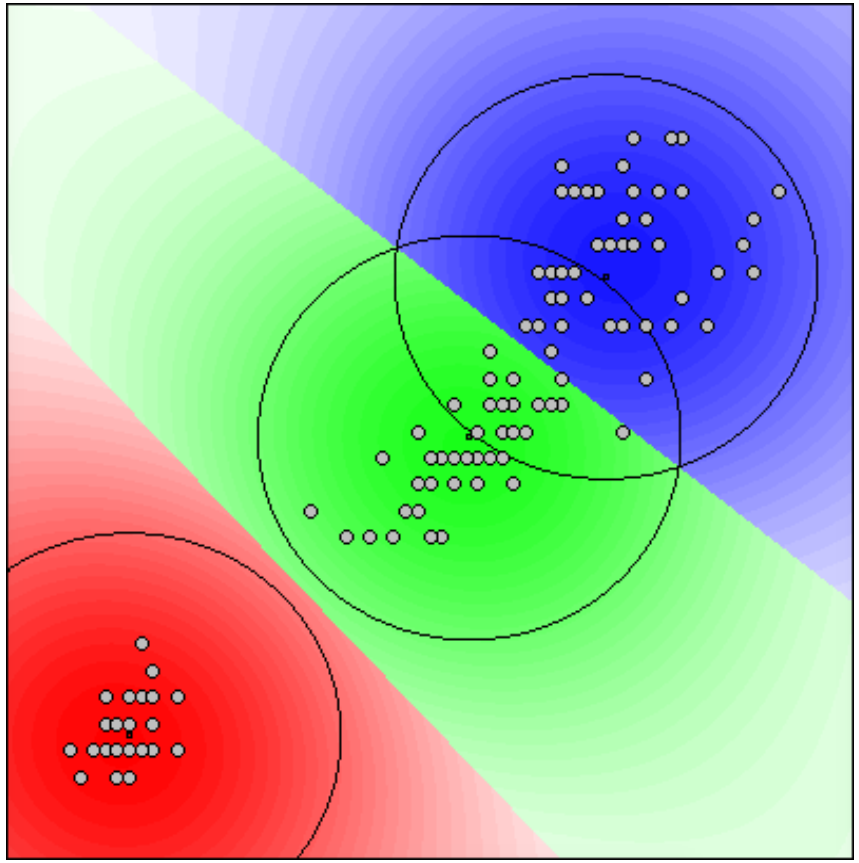
Reminder: The Iris Data



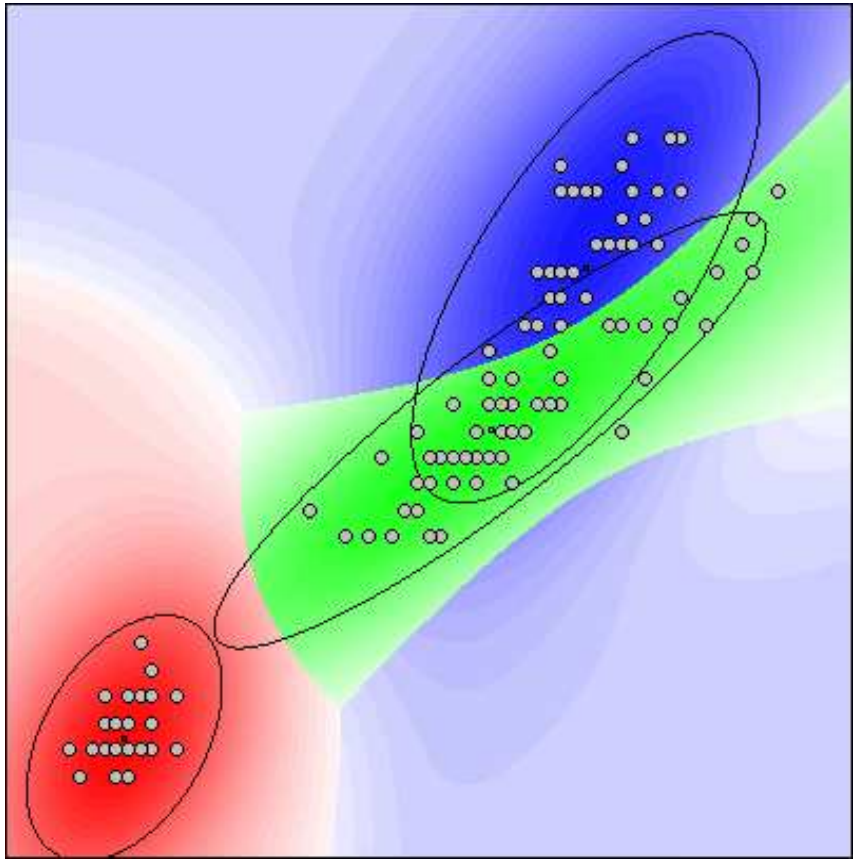


- Scatter plots of the iris data set for sepal length vs. sepal width (left) and for petal length vs. petal width (right).
  All quantities are measured in centimeters (cm).

Fuzzy Clustering of the Iris Data



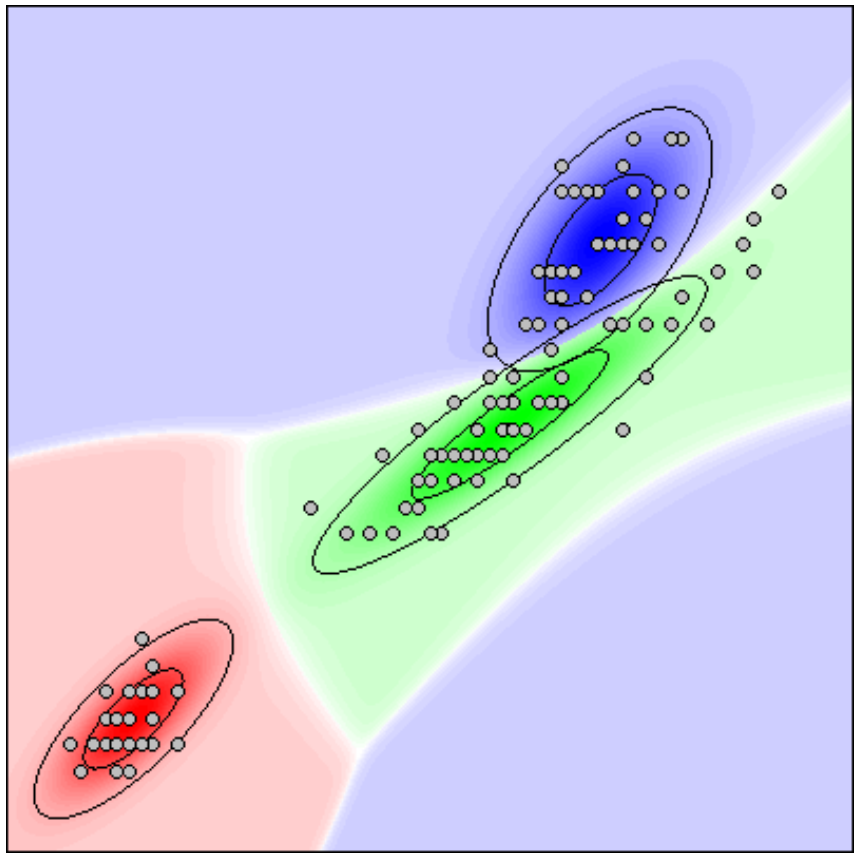
Fuzzy c-Means



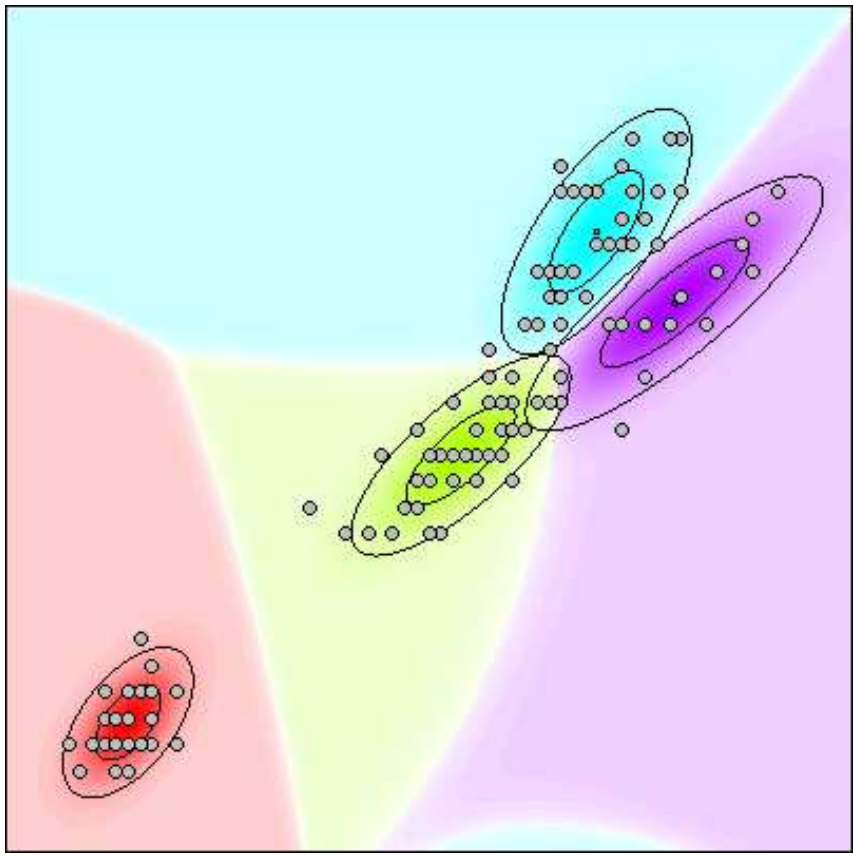
Gustafson–Kessel

All four attributes used (sepal width and length, petal width and length)

Fuzzy Clustering of the Iris Data



3 clusters



4 clusters

Gustafson–Kessel, only petal width and petal length

Expectation Maximization: Mixture of Gaussians

Expectation Maximization: Mixture of Gaussians

- **Assumption:** Data was generated by sampling a set of normal distributions. (The probability density is a mixture of Gaussian distributions.)
- **Formally:** We assume that the probability density can be described as

$$f_{\vec{X}}(\vec{x}; \mathbf{C}) = \sum_{y=1}^{c} f_{\vec{X},Y}(\vec{x}, y; \mathbf{C}) = \sum_{y=1}^{c} p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C}).$$

$\mathbf{C}$	is the set of cluster parameters
$\vec{X}$	is a random vector that has the data space as its domain
Y	is a random variable that has the cluster indices as possible values (i.e., $\mathrm{dom}(\vec{X}) = \mathbb{R}^m$ and $\mathrm{dom}(Y) = \{1, \ldots, c\}$)
$p_Y(y; \mathbf{C})$	is the probability that a data point belongs to (is generated by) the y-th component of the mixture
$f_{\vec{X}\|Y}(\vec{x}\|y; \mathbf{C})$	is the conditional probability density function of a data point given the cluster (specified by the cluster index y)

Expectation Maximization

- **Basic idea:** Do a maximum likelihood estimation of the cluster parameters.
- **Problem:** The likelihood function,

$$L(\mathbf{X};\mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j}(\vec{x}_j;\mathbf{C}) = \prod_{j=1}^{n} \sum_{y=1}^{c} p_Y(y;\mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}_j|y;\mathbf{C}),$$

  is difficult to optimize, even if one takes the natural logarithm (cf. the maximum likelihood estimation of the parameters of a normal distribution), because

$$\ln L(\mathbf{X};\mathbf{C}) = \sum_{j=1}^{n} \ln \sum_{y=1}^{c} p_Y(y;\mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}_j|y;\mathbf{C})$$

  contains the natural logarithms of complex sums.
- **Approach:** Assume that there are "hidden" variables Y_j stating the clusters that generated the data points $\vec{x}_j$, so that the sums reduce to one term.
- **Problem:** Since the Y_j are hidden, we do not know their values.

Expectation Maximization

- **Formally:** Maximize the likelihood of the "completed" data set $(\mathbf{X}, \vec{y})$, where $\vec{y} = (y_1, \ldots, y_n)$ combines the values of the variables Y_j. That is,

$$L(\mathbf{X}, \vec{y}; \mathbf{C}) = \prod_{j=1}^{n} f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) = \prod_{j=1}^{n} p_{Y_j}(y_j; \mathbf{C}) \cdot f_{\vec{X}_j | Y_j}(\vec{x}_j | y_j; \mathbf{C}).$$

- **Problem:** Since the Y_j are hidden, the values y_j are unknown (and thus the factors $p_{Y_j}(y_j; \mathbf{C})$ cannot be computed).

- **Approach to find a solution nevertheless:**
  - See the Y_j as random variables (the values y_j are not fixed) and consider a probability distribution over the possible values.
  - As a consequence $L(\mathbf{X}, \vec{y}; \mathbf{C})$ becomes a random variable, even for a fixed data set $\mathbf{X}$ and fixed cluster parameters $\mathbf{C}$.
  - Try to **maximize the expected value** of $L(\mathbf{X}, \vec{y}; \mathbf{C})$ or $\ln L(\mathbf{X}, \vec{y}; \mathbf{C})$ (hence the name **expectation maximization**).

Expectation Maximization

- **Formally:** Find the cluster parameters as

$$\hat{\mathbf{C}} = \underset{\mathbf{C}}{\operatorname{argmax}}\, E([\ln] L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}),$$

  that is, maximize the expected likelihood

$$E(L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}) \cdot \prod_{j=1}^{n} f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C})$$

  or, alternatively, maximize the expected log-likelihood

$$E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y}\,|\mathbf{X}; \mathbf{C}) \cdot \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}).$$

- Unfortunately, these functionals are still difficult to optimize directly.
- **Solution:** Use the equation as an iterative scheme, fixing $\mathbf{C}$ in some terms (iteratively compute better approximations, similar to Heron's algorithm).

Excursion: Heron's Algorithm

- **Task:** Find the square root of a given number x, i.e., find $y = \sqrt{x}$.
- **Approach:** Rewrite the defining equation $y^2 = x$ as follows:

$$y^2 = x \quad \Leftrightarrow \quad 2y^2 = y^2 + x \quad \Leftrightarrow \quad y = \frac{1}{2y}(y^2 + x) \quad \Leftrightarrow \quad y = \frac{1}{2}\left(y + \frac{x}{y}\right).$$

- Use the resulting equation as an iteration formula, i.e., compute the sequence

$$y_{k+1} = \frac{1}{2}\left(y_k + \frac{x}{y_k}\right) \qquad \text{with} \qquad y_0 = 1.$$

- It can be shown that $0 \leq y_k - \sqrt{x} \leq y_{k-1} - y_n$ for $k \geq 2$.
  Therefore this iteration formula provides increasingly better approximations of the square root of x and thus is a safe and simple way to compute it.
  Example: $x = 2$: $y_0 = 1$, $y_1 = 1.5$, $y_2 \approx 1.41667$, $y_3 \approx 1.414216$, $y_4 \approx 1.414213$.
- Heron's algorithm converges very quickly and is often used in pocket calculators and microprocessors to implement the square root.

Expectation Maximization

- **Iterative scheme for expectation maximization:**

  Choose some initial set $\mathbf{C}_0$ of cluster parameters and then compute

$$
\begin{aligned}
\mathbf{C}_{k+1} &= \operatorname*{argmax}_{\mathbf{C}} E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}_k) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{\vec{y} \in \{1,\ldots,c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y} \,|\mathbf{X}; \mathbf{C}_k) \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{\vec{y} \in \{1,\ldots,c\}^n} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l; \mathbf{C}_k) \right) \sum_{j=1}^{n} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
&= \operatorname*{argmax}_{\mathbf{C}} \sum_{i=1}^{c} \sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}).
\end{aligned}
$$

- It can be shown that each EM iteration increases the likelihood of the data and that the algorithm converges to a local maximum of the likelihood function (that is, EM is a safe way to maximize the likelihood function).

Justification of the last step on the previous slide:

$$\sum_{\vec{y}\in\{1,\ldots,c\}^n} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k) \right) \sum_{j=1}^{n} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,y_j;\mathbf{C})$$

$$= \sum_{y_1=1}^{c} \cdots \sum_{y_n=1}^{c} \left(\prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k) \right) \sum_{j=1}^{n} \sum_{i=1}^{c} \delta_{i,y_j} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C})$$

$$= \sum_{i=1}^{c} \sum_{j=1}^{n} \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C}) \sum_{y_1=1}^{c} \cdots \sum_{y_n=1}^{c} \delta_{i,y_j} \prod_{l=1}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k)$$

$$= \sum_{i=1}^{c} \sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}_k) \cdot \ln f_{\vec{X}_j,Y_j}(\vec{x}_j,i;\mathbf{C})$$

$$\underbrace{\sum_{y_1=1}^{c} \cdots \sum_{y_{j-1}=1}^{c} \sum_{y_{j+1}=1}^{c} \cdots \sum_{y_n=1}^{c} \prod_{l=1,l\neq j}^{n} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k)}_{= \prod_{l=1,l\neq j}^{n} \sum_{y_l=1}^{c} p_{Y_l|\vec{X}_l}(y_l|\vec{x}_l;\mathbf{C}_k) \;=\; \prod_{l=1,l\neq j}^{n} 1 \;=\; 1} .$$

Expectation Maximization

- The probabilities $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$ are computed as

$$p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) = \frac{f_{\vec{X}_j,Y_j}(\vec{x}_j, i; \mathbf{C}_k)}{f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}_k)} = \frac{f_{\vec{X}_j|Y_j}(\vec{x}_j|i; \mathbf{C}_k) \cdot p_{Y_j}(i; \mathbf{C}_k)}{\sum_{l=1}^{c} f_{\vec{X}_j|Y_j}(\vec{x}_j|l; \mathbf{C}_k) \cdot p_{Y_j}(l; \mathbf{C}_k)},$$

  that is, as the relative probability densities of the different clusters (as specified by the cluster parameters) at the location of the data points $\vec{x}_j$.

- The $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$ are the posterior probabilities of the clusters given the data point $\vec{x}_j$ and a set of cluster parameters $\mathbf{C}_k$.

- They can be seen as **case weights** of a "completed" data set:
  - Split each data point $\vec{x}_j$ into c data points $(\vec{x}_j, i)$, $i = 1, \ldots, c$.
  - Distribute the unit weight of the data point $\vec{x}_j$ according to the above probabilities, i.e., assign to $(\vec{x}_j, i)$ the weight $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$, $i = 1, \ldots, c$.

Expectation Maximization: Cookbook Recipe

Core Iteration Formula

$$\mathbf{C}_{k+1} = \underset{\mathbf{C}}{\operatorname{argmax}} \sum_{i=1}^{c} \sum_{j=1}^{n} p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C})$$

Expectation Step

- For all data points $\vec{x}_j$:
  Compute for each normal distribution the probability $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$ that the data point was generated from it
  (ratio of probability densities at the location of the data point).
  $\rightarrow$ "weight" of the data point for the estimation.

Maximization Step

- For all normal distributions:
  Estimate the parameters by standard maximum likelihood estimation using the probabilities ("weights") assigned to the data points w.r.t. the distribution in the expectation step.

Expectation Maximization: Mixture of Gaussians

Expectation Step: Use Bayes' rule to compute

$$p_{C|\vec{X}}(i|\vec{x};\mathbf{C}) = \frac{p_C(i;\mathbf{c}_i)\cdot f_{\vec{X}|C}(\vec{x}|i;\mathbf{c}_i)}{f_{\vec{X}}(\vec{x};\mathbf{C})} = \frac{p_C(i;\mathbf{c}_i)\cdot f_{\vec{X}|C}(\vec{x}|i;\mathbf{c}_i)}{\sum_{k=1}^{c} p_C(k;\mathbf{c}_k)\cdot f_{\vec{X}|C}(\vec{x}|k;\mathbf{c}_k)}.$$

$\rightarrow$ "weight" of the data point $\vec{x}$ for the estimation.

Maximization Step: Use maximum likelihood estimation to compute

$$\varrho_i^{(t+1)} = \frac{1}{n}\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)}), \qquad \vec{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})\cdot\vec{x}_j}{\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})},$$

$$\text{and} \qquad \mathbf{\Sigma}_i^{(t+1)} = \frac{\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})\cdot\left(\vec{x}_j-\vec{\mu}_i^{(t+1)}\right)\left(\vec{x}_j-\vec{\mu}_i^{(t+1)}\right)^\top}{\sum_{j=1}^{n} p_{C|\vec{X}_j}(i|\vec{x}_j;\mathbf{C}^{(t)})}$$

Iterate until convergence (checked, e.g., by change of mean vector).

Expectation Maximization: Technical Problems

- If a fully general mixture of Gaussian distributions is used,
  the likelihood function is truly optimized if
  - all normal distributions except one are contracted to single data points and
  - the remaining normal distribution is the maximum likelihood estimate for the remaining data points.
- This undesired result is rare,
  because the algorithm gets stuck in a local optimum.
- Nevertheless it is recommended to take countermeasures,
  which consist mainly in reducing the degrees of freedom, like
  - Fix the determinants of the covariance matrices to equal values.
  - Use a diagonal instead of a general covariance matrix.
  - Use an isotropic variance instead of a covariance matrix.
  - Fix the prior probabilities of the clusters to equal values.