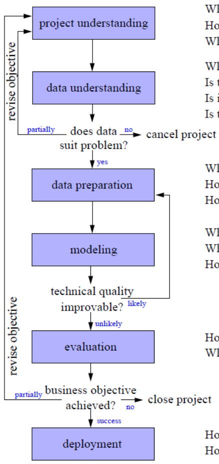
Data Analysis and Knowledge Discovery

Antti Airola

University of Turku
Department of Computing

Where are we now?



What exactly is the problem, the expected benefit?
How would a solution look like?
What is known about the domain?

What data do we have available?
Is the data relevant to the problem?
Is it valid? Does it reflect our expectations?
Is the data quality, quantity, recency sufficient?

Which data should we concentrate on?
How is the data best transformed for modeling?
How may we increase the data quality?

What kind of model architecture suits the problem best?
What is the best technique/method to get the model?
How good does the model perform technically?

How good is the model in terms of project requirements?
What have we learned from the project?

How is the model best deployed?
How do we know that the model is still valid?

Outline

Model selection and evaluation

Cross-validation

Performance measures for classification and regression

Evaluating the prediction performance

Two goals

1. Model selection: almost all methods have free parameters that need to be chosen
   - hyperparameters: number of neighbors, regularization parameter, kernel type, network architecture...
   - choice of learning algorithm, feature extraction, selection and normalization...
2. Model evaluation / assessment: after selecting your hypothesis, estimate how well it generalizes to new data

Loss function

- Hypothesis: $h(x) = y'$
  - learned from training set of (x_i, y_i) pairs
  - e.g. trained k-nn model, linear model, decision tree...
- Loss (error) function $l(y', y)$ tells the price we pay for predicting y', when true value is y
  - equivalently: scoring function where large values are good (e.g. misclassification rate = 1 - classification accuracy)
- What we want to know: what would be the expected loss of h on (randomly drawn) new data

Example: zero-one loss

$$l(y', y) = \begin{cases} 1 & \text{if } y' \neq y, \\ 0 & \text{otherwise} \end{cases}$$

- typical loss for classification problems
- you get 1 error point for incorrect classification, 0 for correct
- average zero-one loss over data set called misclassification rate
- if 10% classified incorrectly, then misclassification rate 0.1
- conversely, classification accuracy defined as the fraction of correctly classified instances

Generalization error

$$\mathrm{Err}(h) = \mathsf{E}[l(h(x), y)] = \int_{X \times Y} l(h(x), y)\rho(x, y)dxdy$$

- expected loss over the probability distribution of our data
- interpretation: loss you would on average get for a random new instance
- we can (almost) never compute this, since we do not know the probability distribution
- need to estimate it somehow from a finite sample of data

Generalization error for zero-one loss

$$\mathrm{Err}(h) = \mathsf{E}[l(h(x), y)] = P[h(x) \neq y]$$

- for zero-one loss the generalization error can be interpreted as the true misclassification error of the classifier
- probability of making a mistake when classifying a randomly drawn test instance

Estimating the generalization error

$$\overline{\mathrm{Err}}(h) = \frac{1}{n}\sum_{i=1}^{n} l(h(x_i), y_i)$$

- where $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ is a finite sample of data
- provides us with an estimate of the generalization error
- training set estimate is unreliable
- standard approaches based on randomly sampling separate training and test data from the available data
- in many techniques, take average of several error estimates

Estimators of generalization error

- Training set error (aka resubstitution error)
- Test set error (aka holdout error)
- Cross-validation error (leave-one-out CV, K-fold CV...)
- Bootstrap error (classical methods, leave-one-out bootstrap, .632 estimator, .632+ estimator...)

What makes a good error estimator

- Let $\mathrm{Err}(h)$ be the true generalization error of hypothesis h
- Let $\overline{\mathrm{Err}}(h)$ be some estimate of this error
- We would like $\overline{\mathrm{Err}}(h) - \mathrm{Err}(h)$ to be as small as possible
  - If true error larger than estimate, we think that h is better than it really is
  - If true error smaller than estimate, we think that h is worse than it really is
- We can analyze the difference in terms of bias and variance

Bias

$$\mathsf{E}[\overline{\mathrm{Err}}(h) - \mathrm{Err}(h)]$$

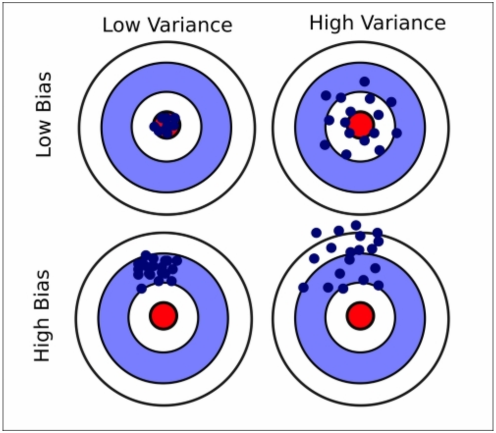
- Expected difference between estimated and true error
- Ideally close to zero
- Optimistic bias: systematically estimate error to be smaller than it really is
    - Typical example: training set error
- Pessimistic bias: systematically estimate error to be larger than it really is
    - Typical example: large test set that is after error estimation merged back to training set before training final model

Variance

$$\mathrm{Var}[\overline{\mathrm{Err}}(h) - \mathrm{Err}(h)]$$

- How large the magnitudes of the differences tend to be on average
- Ideally close to zero
- Method with zero bias can still have large variance!
  - Errors in the positive and negative direction can "cancel each other out"
- Usually the more data we have the smaller the variance

Bias and variance



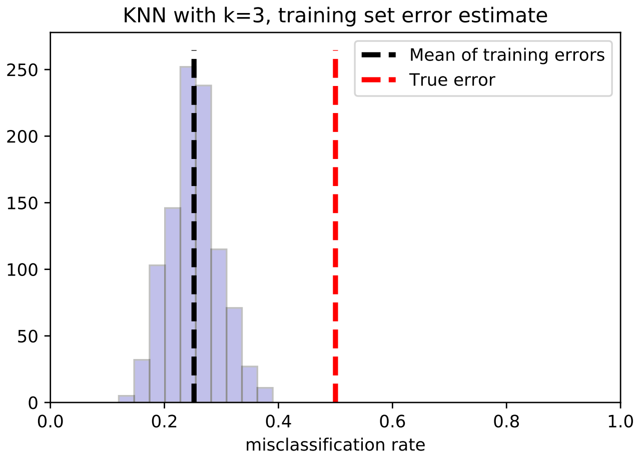
Simulation study

- Simple simulation study for different error estimation methods
- **x** has 10 normally distributed features
- $y \in \{0, 1\}$ binary class value, generated by (fair) coin flips
- $P(y = 1) = P(y = 0) = 0.5$
- $P(y|x) = P(y)$, that is x provides no information about y
- True expexted misclassification rate is 0.5 for all possible classifiers, since whatever you predict you have 50% chance of guessing right
- 1000 repetitions, 100 training and test instances in first experiments

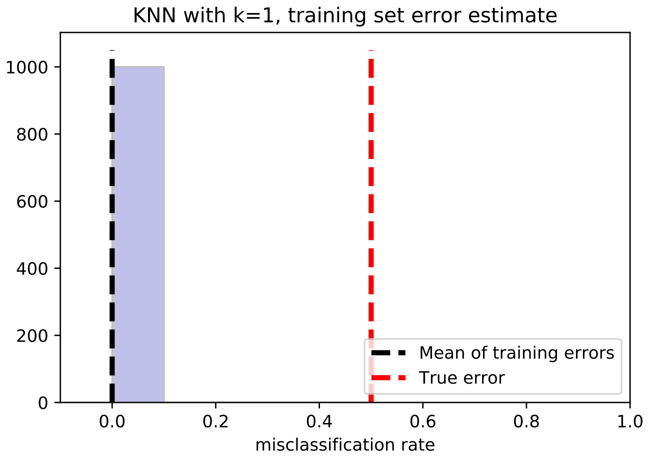
Training set error

- Also known as resubstitution error
- Compute average loss on the training examples
- Has high optimistic bias, since your model was already chosen due to good fit to this particular data set!
- In model selection will tend to favor the most complex models that can overfit the most
- Do not use this!

Training set error



Training set error



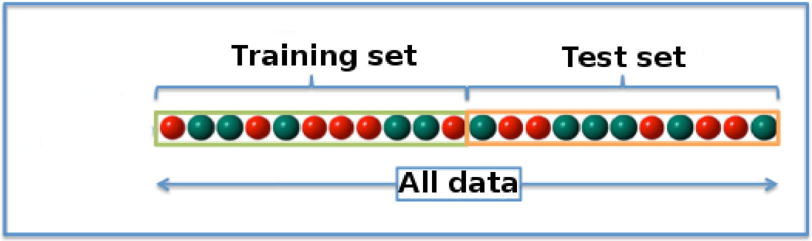
Training set error

- seriously, never ever report your model's performance on its training data and claim that this is how well it predicts!
- if you use very restricted and simple models and you have lots of data, you might get from this a reasonable estimate of the true error, but...
- when you have overfitted a complex model to your data, you may get close to zero error, even when the model is no better than random
- (machine) learning is not about memorization, learning is about generalization

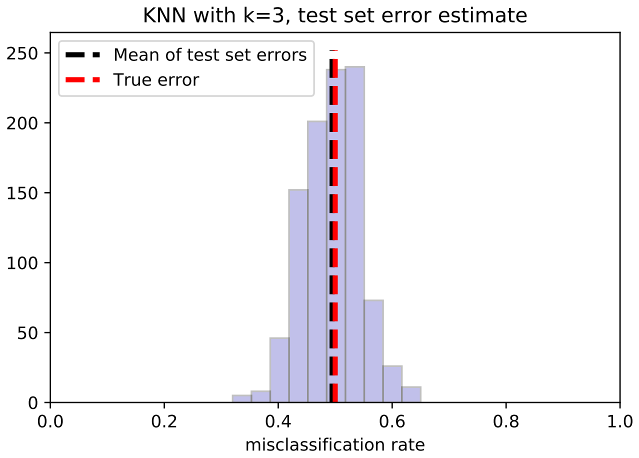
Test set error

- Also known as holdout estimate
- Split your data randomly to training and test set (e.g. 70% and 30%)
- Train on the training set, compute average loss on test set
- Unbiased estimate of how well the model trained on the training set works
- Problem: we are not usually interested in testing only one model, but need to do model selection

Test set



Test set error



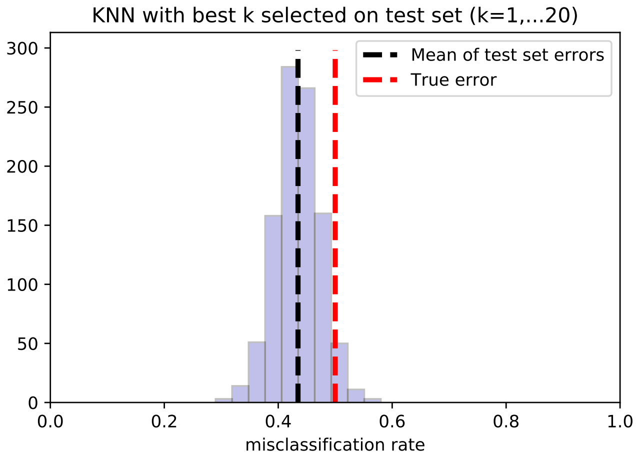
Overfitting to test set

- Test set simulates unseen future data
- No modeling decisions should be based on the test set
- Pre-processing should be done solely based on information from training set
  - When normalizing data, imputing missing values, selecting relevant features etc. compute the needed statistics only from training set
  - See recommended practices with sklearn: `https://scikit-learn.org/stable/common_pitfalls.html`
- Selecting learning algorithm or hyperparameters against test set will also leak information

Overfitting to test set

- Test set error only unbiased, if used for testing a single hypothesis
- In model selection, we test many different hyperparameter etc. combinations
  - This multiple hypothesis testing leads to optimistic bias, since we select best out of many alternatives based on test set error
- example: 6 models, true expected error rates are [0.40, 0.20, 0.10, 0.10, 0.10]
- test set estimates: [0.42, 0.18, 0.13, 0.10, 0.07]
- we pick the last one, leading to a good choice of model, but the test set estimate is now biased (0.07 smaller than the true error rate 0.10)
- Solution: two independent test sets, first one used for model selection, and second used for testing the final hypothesis

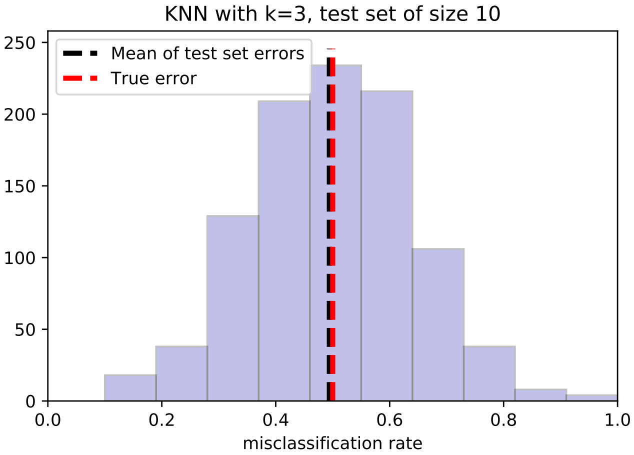
Overfitting to test set



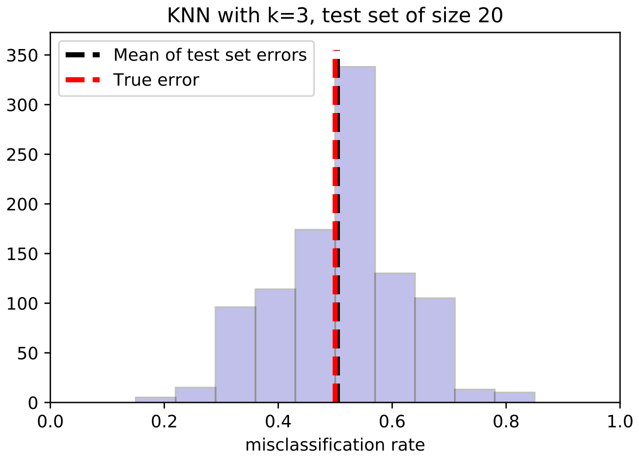
Training, validation and test set

- Split your data randomly to three non-overlapping parts (e.g. 50%, 25%, 25%)
- Training set: used for training the models
- Validation (or model assessment) set: when comparing different models trained on training set, select one with lowest error on validation set
- Test set: test the final hypothesis on test set to get unbiased error estimate for it

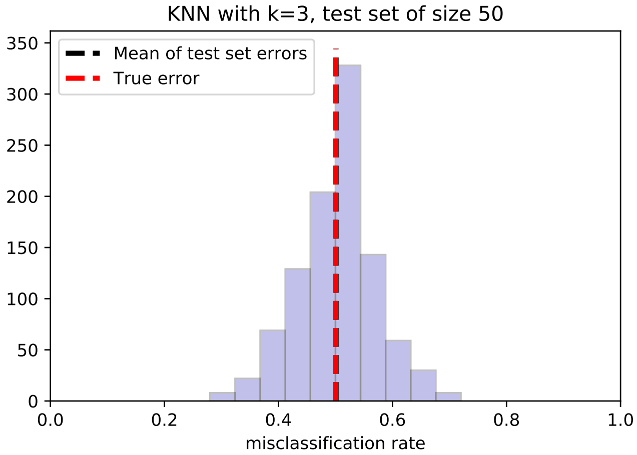
Test set size and variance



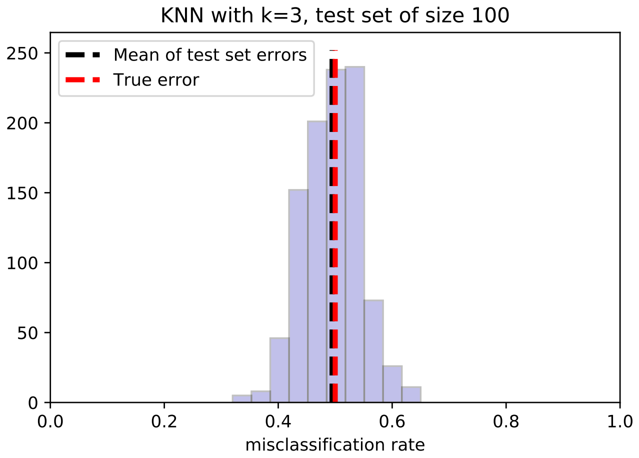
Test set size and variance



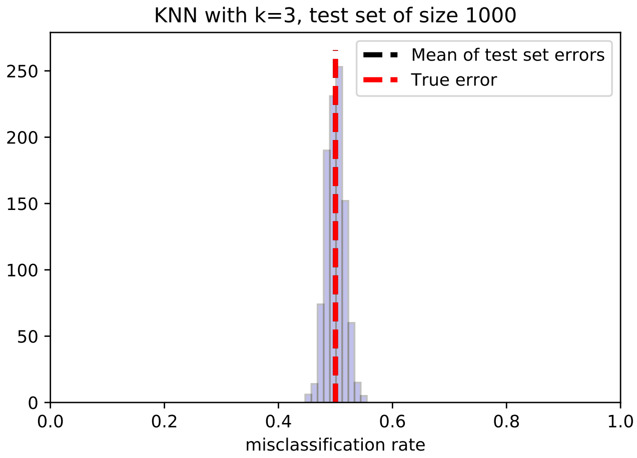
Test set size and variance



Test set size and variance



Test set size and variance



Test set size and variance

- as test set size grows large enough, test set error likely to be close to true error
- for small data sets, our estimates can be far from the truth
- solutions:
  - don't trust results obtained on small data sets - get more data!
  - cross-validation methods make more efficient use of the data than a single training/validation/test split
  - report confidence intervals for your results (different ways to compute these depending on the error/accuracy measure used)

Tricks of the trade

- Stratification: for classification problems do the random split for each class separately to guarantee similar distribution of classes in each set
  - Otherwise one of the sets might contain no instances from some rare class!
  - Stratification possible but more complicated for regression problems
- Never assume that the data set is originally in random order
- After error estimation the final model is often trained on combined training, validation and test set, using best hyperparameters found during model selection
  - Justification: this is likely to be better than the one learned from the smaller training set
  - Complication: randomized optimization approaches where different runs with same hyperparameters can lead to very different quality solutions (e.g. neural network)

Outline

Model selection and evaluation

Cross-validation

Performance measures for classification and regression

Motivation for cross-validation

- Using a separate test set gives an unbiased estimate of the true expected error (or AUC, or...) on new test data
- Can still be unreliable on too small test sets due to variance, we might get very "lucky" or "unlucky" in which instances happen to fall in the test set
- two conflicting goals
  - we want as much data as possible for training in order to be able to discover the possible pattern as well as possible
  - we want as much data as possible for testing in order to get as reliable estimate of true error rate as possible
- when working with Big Data not a problem, we have as much training and test data as we want

Limitations of using a test set

- What if we have only a data set of 100 instances
- Using e.g. 70 for training and 30 for testing, we get a very unreliable estimate of test performance (too small test set)
- Using e.g. 30 for training and 70 for testing does not help much, now too little training data
- Even worse considering the need for validation set
- especially bad on imbalanced data, where we could have e.g. 90 instances from majority and only 10 from minority class
- would be nice to use all data for both training and testing, but we can't use training error directly due to overfitting
- Cross-validation to the rescue!

Leave-one-out cross-validation

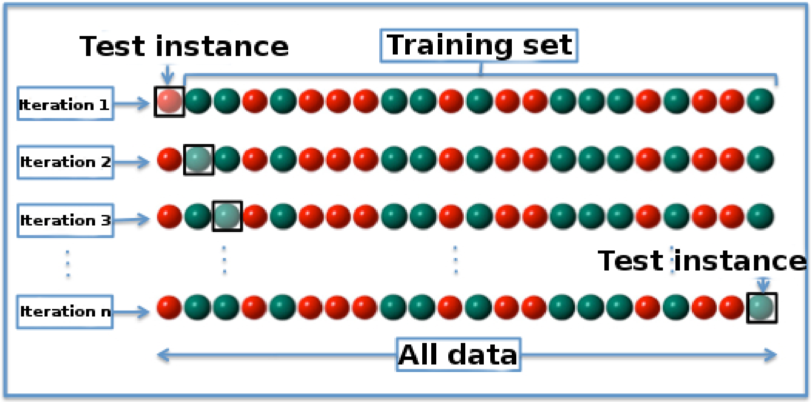
Basic idea

- we have a data set of n instances
- a model cannot be tested on its own training data due to overfitting, so need to divide into training and test set
- question we ask for each instance $i \in \{1, ..., n\}$
  - "what would the model have predicted for the i:th instance, if it had not seen it during training?"
- we can answer this question by removing the i:th instance, training the model on the $n - 1$ remaining instances, and making a prediction for the i:th one (a test set of size 1)
- this is known as leave-one-out cross-validation

Leave-one-out cross-validation

```
1. for $i = 1$ to $n$ do
2.         train model on all but the $i$th instance
3.         predict output for the $i$th instance with the model
4. end for
5. compute error between true and predicted outputs
6. train final model on all the data
```

Leave-one-out cross-validation



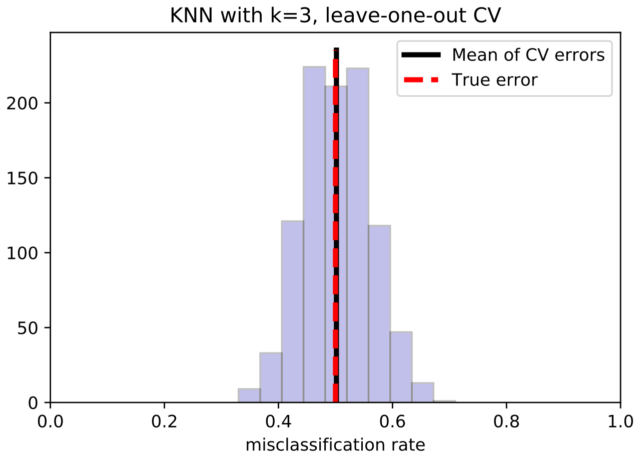
Analyzing leave-one-out cross-validation

- validating the models trained on $n-1$ training instances gives us a good (almost unbiased) estimate of how the final model trained on all data will behave
- one each iteration, the model cannot overfit to the test instance since it is not part of training set
- leave-one-out allows us to use the whole data set for both training and testing simultaneously
- parameter selection: pick the model with lowest leave-one-out error (or highest AUC or correlation or...)
- not a perfect method: can find counter examples where cross-validation fails

Implementing leave-one-out

- for nearest neighbor extremely simple to implement
  - for each instance in training set, compute prediction from k nearest neigbors, excluding the instance itself
- generally, for other types of learning methods you need a loop where the machine learning method is trained n times
- can be computationally expensive
  - 5000 instances, 1 minute training time $\rightarrow$ leave-one-out cross-validation takes 3.5 days
- Advanced topic: some machine learning methods allow computational shortcuts for computing leave-one-out much faster (k-nn, ridge regression, Naive Bayes...)

Leave-one-out CV simulation



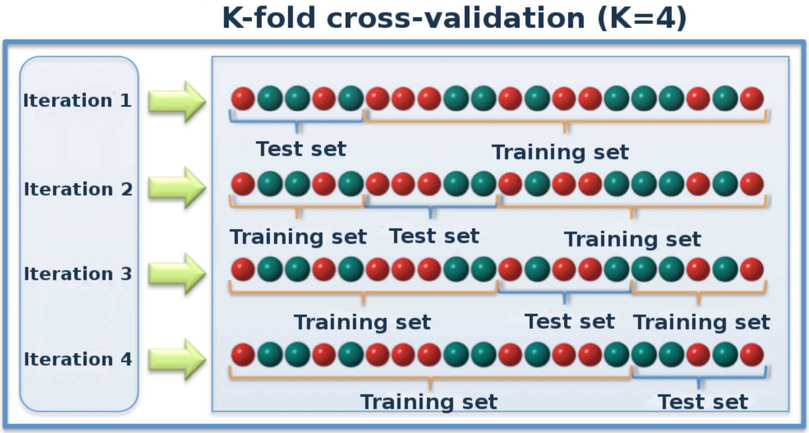
K-fold cross-validation

- for larger datasets leave-one-out can be impractical due to high computational costs
- K-fold cross-validation: same basic idea, but instead of leaving a single instance out at a time, leave several
- data is randomly divided into K (approximately) equally sized non-overlapping parts called folds
- on each round of cross-validation, use one fold for testing and remaining $K-1$ for training
- usual choices for K: 10 or 5
- leave-one-out as extreme case, where $K = n$

K-fold cross-validation

```
1. split the data randomly into $K$ equally sized folds
2. for $i = 1$ to $K$ do
3.         train model on data from all but the $i$th fold
4.         predict outputs for the instances belonging to the $i$th
   fold with the model
5. end for
6. compute error between true and predicted outputs
7. train final model on all the folds combined
```

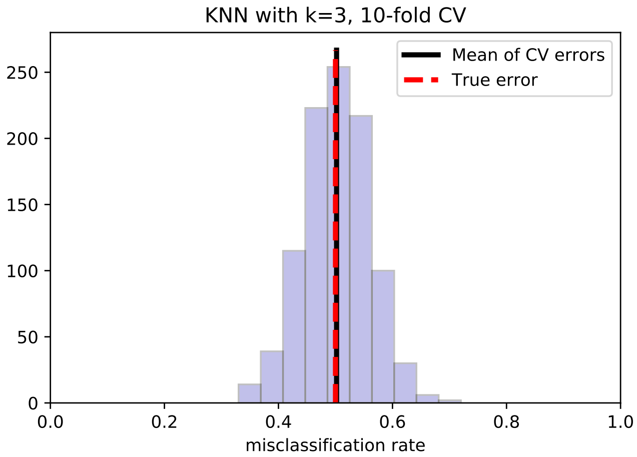
K-fold cross-validation



Analyzing K-fold cross-validation

- much faster than leave-one-out; 5000 instances, 1 minute training time $\rightarrow$ 10-fold CV takes 10 minutes
- small pessimistic bias, the model evaluated can be slightly less accurate than the final model trained on all data
- some internal variance depending on the random fold split

K-fold CV simulation



Some variants of K-fold cross-validation

stratified K-fold cross-validation

- especially imbalanced classification problems
- stratification: split the data into folds so, that each fold roughly has the same fraction of members from each class as the full data set
- guarantees that on all the iterations, the class distributions of the training and test set resemble that of the full data set
- otherwise with very bad luck, it might happen that all instances from minority class end up in one fold

Some variants of K-fold cross-validation

M-times repeated K-fold cross-validation

- some variance in the K-fold estimate due to the randomness in how the folds are split
- can reduce the variance and thus obtain more reliable error estimates by repeating the whole procedure with different fold divisions M times (e.g. M=100) and taking the average
- 5000 instances, 1 minute training time $\rightarrow$ 100 times repeated 10-fold CV takes almost 17 hours (1000 minutes)

Overfitting to cross-validation

- In cross-validation, test fold simulates unseen future data
- No modeling decisions should be based on the test fold
- As before, pre-processing should be done solely based on information from training folds
    - Can be implemented with Pipelines with sklearn: `https://scikit-learn.org/stable/common_pitfalls.html`

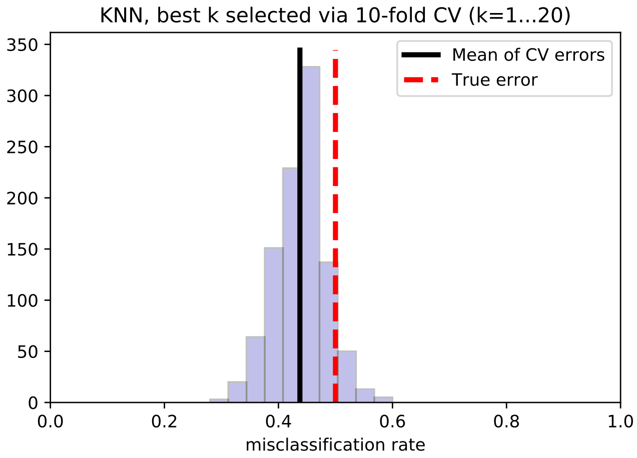
Combining model selection and final evaluation

- using cross-validation (or a single test set) we can do model selection
  - choose hyperparameter value / feature set / learning algorithm that gives lowest cross-validation error (or conversely, highest accuracy/AUC/correlation...)
  - train final model with these parameters on all folds combined
- test the final model on independent test data
- no longer need separate validation set for parameter selection
- what if we want to use cross-validation also to replace evaluation on separate test set?

Combining model selection and final evaluation

- The error rate estimate of the final model biased (smaller than the true error rate) since out of several choices we pick the one having smallest estimated error
- can be problematic especially on small datasets and when testing a large number of different models
- need additional independent test data, that was not used for model selection

Model selection biases CV optimistically



Cross-validation and separate test set

- Idea 1: Separate data to training and test set
- Use cross-validation on training set for model selection
- Estimate how well the model selected by CV generalizes on the test set
- Idea: cross-validation simulates having separate validation set
  - Instead of e.g. 50%, 25%, 25% split, can have 50%, 50% or 75%, 25% split to training / test set
  - Problem: If small data, still not enough for training or reliable testing
- Could we simulate both validation and test set with cross-validation?

Nested cross-validation

- nested, aka external cross-validation
- cross-validation within cross-validation
- inner loop for model selection, outer for evaluation
- Split the data to K non-overlapping folds, for $i = 1 ... K$ do:
    1. designate i:th fold as the outer cross-validation test fold, set it aside
    2. use the remaining K-1 folds to perform model selection with cross-validation as usual
    3. train the model with selected parameters on the $K - 1$ training folds
    4. compute predictions using the remaining ith fold as test set
- afterwards collect the together predictions made at step 4 for all the K folds and compute your error rate or other performance measure

Nested cross-validation

- can be computationally very expensive (combinatorial explosion)
- leave-one-out within leave-one-out: need to train model $n * (n-1) * (\#\text{tested params})$ times
- 5000 instances, 1 minute training time $\rightarrow$ nested leave-one-out cross-validation takes about 48 years
- nested-leave-one-out feasible only on very small data sets, more often nested K-fold
- nesting especially important when you test a really large number of hyperparameters

Software for cross-validation

- you should program cross-validation once by yourself to get the hang of it
- all major data analysis environments offer also this functionality
- see e.g. `http://scikit-learn.org/stable/modules/cross_validation.html`
- made in Turku - extremely fast CV methods for ridge regression (aka Regularized Least-Squares)
  - `https://github.com/aatapa/RLScore`
  - tutorials at `http://staff.cs.utu.fi/~aatapa/software/RLScore/`

Alternative criteria for model selection

- there are alternative approaches for model selection compared to using cross-validation
- information criteria, that balance fit to training data with model complexity
- e.g. Akaike's and the Bayesian information criterion (see book)
- general underlying principles, but the way the complexity can be evaluated is highly dependent on our assumptions about the type of models considered
- cross-validation more than competitive with these approaches, and generally applicable to any type of model used

Challenges in performance evaluation

- all of the above rests on the assumption that our data is an i.i.d. sample (independent and identically distributed)
- we can trust our randomized training/validation/test split or cross-validation to work correctly as long as this assumption is not violated too much
- what if there are strong dependencies between our instances?
- next some typical examples

Grouped data

- data has group structure, where instances inside group more similar to each other than compared to other groups
- with random data split, instances from same group likely to end up in both training and test sets!
- much easier to predict for group that has members in training set
- but can our model make accurate predictions for instances in new groups?
- solution: randomize data on level of whole groups, not individual instances

Grouped data - example

- task: predict the type of object appearing in an image
- groups: 10 pictures of each object, different angles and lighting conditions
- if you have 5 of the images in training data, easy to predict category for the 5 very similar test images
- unrealistic results - for new object you would not get this training information
- solution: each group of 10 images fully in training/test set

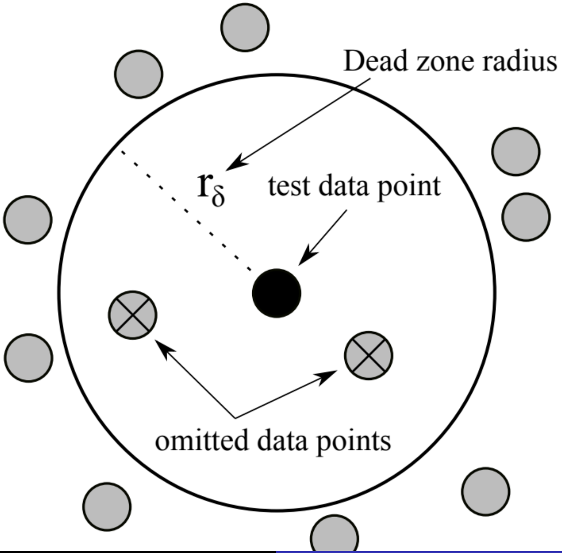
Time series data

- example: stock prices, each instance represents one day
- random split not suitable, easy to predict price on 3. May 2018, if we have 2. May 2018 and 4. May 2018 in training set!
- should not test predicting past from the future!
- also, much easier to predict dates close to last training instance, than several years to the future
- different training/test splits possible to simulate different assumptions about availability of training data, and how far into future predictions are needed

Spatial data

- example: predicting where gold can be found in Lapland, each instance represents on point in a map
- random split not suitable, easy to predict areas close to training points
- Tobler's first law of geography (1970): *"Everything is related to everything else, but near things are more related than distant things"*
- Idea: if you want to test how well your model predicts 50 km from your training data, use test data from that distance

Spatial data



Outline

Model selection and evaluation

Cross-validation

Performance measures for classification and regression

Evaluating the prediction performance

- Real thing we would like to measure is how much money we make from using the model, how many lives are saved, how many people are happier...
- Maybe sometimes possible for final evaluation of the model, but not during development
- For a predictive model, the natural criterion to measure is how accurately it predicts
- Basic measures: misclassification rate, squared error
- Advanced: area under ROC curve (AUC), for binary classification problems
- Many more in different application domains

Evaluating the predictive accuracy

We are given a test set of input-output pairs

$$\{(x_1, y_1), \ldots, (x_n, y_n)\}$$

- let $h(\cdot)$ be our model, x an input and y the correct output
- We would hope that $h(x) \approx y$, for any randomly chosen new (x, y) input-output pair
- Let $m(y', y)$ be some function that measures how well a predicted value y'matches the true value y
- Obviously, average performance on this test set is
$$\frac{1}{n}\sum_{i=1}^{n} m(h(x_i), y_i)$$

Performance measures

- Next, we go through some popular performance measures used to benchmark regression models or classifiers
- sometimes loss optimized during model training, and for testing it are the same (e.g. we fit regression model with least-squares method, and compute mean squared error on validation/test set)
- more often not - very difficult to directly optimize non-continuous performance measures, such as misclassification error or AUC
- trivial baselines for different performance measures
    - you should do better than the baselines to claim that you have learned something from the data
    - surprisingly often not checked!

Mean squared error

$$m(y', y) = (y' - y)^2$$

- regression error, zero if the prediction correct, otherwise grows quadratically, value not bounded
- sensitive to outliers, a single very bad prediction can lead to really bad mean squared error
- often also root mean squared error (square root of sum of mean squared errors)
- baseline: is your MSE smaller than predicting mean of training set y values

Mean absolute error

$$m(y', y) = |y' - y|$$

- regression error, zero if the prediction correct, otherwise grows linearly, value not bounded
- not as sensitive to outliers as squared error
- easier to interpret than squared error, harder to optimize and analyze,
- baseline: is your MAE smaller than predicting median of training set y values?

Misclassification rate

$$m(y', y) = \begin{cases} 1 & \text{if } y' \neq y, \\ 0 & \text{otherwise} \end{cases}$$

- misclassification rate is the fraction of incorrectly classified examples, a number between zero and one
- if 10% classified incorrectly, then misclassification rate 0.1
- conversely, classification accuracy defined as the fraction of correctly classified instances
- baseline: majority voter

Misclassification rate

- A low misclassification rate does not necessarily tell anything about the quality of a classifier.
- A low misclassification rate might be easily achieved when classes are extremely unbalanced, i.e. when the distribution of classes deviates strongly from a uniform distribution.
- Example. Data from production of tea cups where the classes are *broken* and *ok*.
  When 99% of the production are *ok*, a classifier always predicting *ok* will have a misclassification rate of 1%.

Cost matrix

- More general approach than the misclassification rate: cost function or cost matrix.
- The consequences (costs) for misclassifications for one class might be different than for another class.

Cost matrix

- Example: Tea cup production.

Cost matrix

	predicted class	
true class	OK	broken
OK	0	c_1
broken	c_2	0

- When an intact cup is classified as broken, the cup must be produced again. Therefore, the costs c_1 caused by this error causes are equal to the production costs for one cup.
- The costs c_2 for a broken cup classified as intact are more difficult to estimate.
  - c_2 must cover the mailing costs for the reproduced cup.
  - costs for loss of reputation of the company for delivering broken cups very difficult to estimate.

Cost matrix

General form of a cost matrix for a multi-class classification problem:

$$\begin{array}{c|cccc} & \multicolumn{4}{c}{\text{predicted class}} \\ \text{true class} & c_1 & c_2 & \dots & c_m \\ \hline c_1 & 0 & c_{1,2} & \dots & c_{1,m} \\ c_1 & c_{2,1} & 0 & \dots & c_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_m & c_{m,1} & c_{m,2} & \dots & 0 \end{array}$$

Cost matrix

When such a cost matrix is provided, the expected loss given by

$$\text{loss}(c_i|x) \ = \sum_{j=1}^{m} P(c_j|x)c_{ji}$$

should be minimized.

- x is the evidence, i.e. the observed values of the predictor attributes used for the classification.
- $P(c_j|x)$ is the predicted probability that the true class is c_j given observation x (e.g. fraction of neighbors of x belonging to class c_j for k-nn).

Cost matrix

Example. (Hypothecial) cost matrix for the tea cup production problem

	predicted class	
true class	OK	broken
OK	0	1
broken	10	0

A classifier might classify a specific cup with 80% to the class *ok* and with 20% to the class *broken*.

- Expected loss for choosing *ok*: $0.8 \cdot 0 + 0.2 \cdot 10 = 2$.
- Expected loss for choosing *broken*: $0.8 \cdot 1 + 0.2 \cdot 0 = 0.8$.

Cost matrix

Using the cost matrix

$$\begin{array}{c|cccc} & \multicolumn{4}{c}{\text{predicted class}} \\ \text{true class} & c_1 & c_2 & \dots & c_m \\ \hline c_1 & 0 & 1 & \dots & 1 \\ c_1 & 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_m & 1 & 1 & \dots & 0 \end{array}$$

corresponds to minimising the misclassification rate.

Confusion matrix

Classification result can be represented by a confusion matrix which is a table where the rows represent the true classes and the columns the predicted classes.

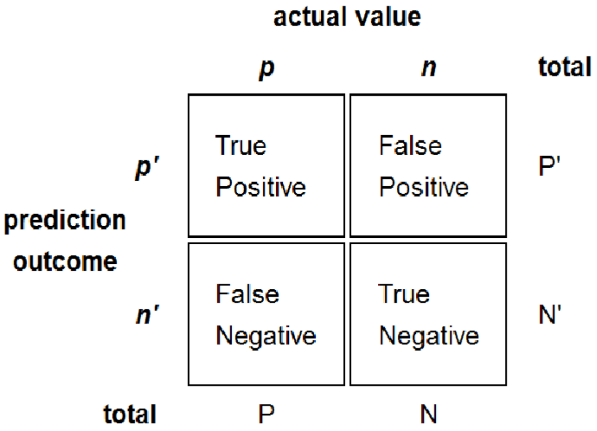
Each entry specifies how many objects from a given class are classified into the class of the corresponding column.

true class	predicted class		
	Iris setosa	Iris versicolor	Iris virginica
Iris setosa	50	0	0
Iris versicolor	0	47	3
Iris virginica	0	2	48

A possible confusion matrix for the Iris data set

Confusion matrix: true/false positives and negatives

A binary classifier case:



		actual value		
		p	*n*	total
prediction outcome	*p'*	True Positive	False Positive	P'
	n'	False Negative	True Negative	N'
	total	P	N	

True/ false positives and negatives

- True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN)
- Several (binary) classification performance measures defined in terms of these
- precision, recall and F-score
- ROC curve, sensitivity, specificity

Precision and recall

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- originally from field of information retrieval
  - positive: document relevant to our search
  - negative: document not relevant for our search
- Precision: proportion of relevant documents among those returned
- Recall: proportion of relevant documents found
- Example: search engine returns 100 documents
  - 40 of them relevant → 40% precision (TP=40, FP=60)
  - 160 relevant documents not returned → 20% recall (FN=160)

Balancing tradeoffs

How can we control the number of TP-FP-TN-FN predictions a classifier makes?

two class problem: 'positive' and 'negative' class, encoded as '1' and '0' (often '+1' and '-1', hence the names)

- classifier may predict probability of belonging to positive class
- more generally, real-valued confidence score
- rule minimizing misclassification rate: if $f(x) > 0.5$ predict positive class, else negative
- problems with imbalanced classes, and different misclassification costs
- $f(x) > t$, moving $t \in \mathbb{R}$ away from 0.5 means favoring one class over another

Maximizing precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- if there are no false positives, precision=100% (as long as there is at least one true positive)
- we can maximize precision by classifying only those instances we are most certain of as positive
- example: if $P(y = 1|x) > 0.98$ predict positive, else negative
- problem: very bad recall

Maximizing recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- maximizing recall is even easier!
- classify everything as positive and we get FN=0
- Recall 100%!
- problem: very bad precision

F-score

$$2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- balanced combination of precision and recall
- originated in information retrieval, widely used in natural language processing, image segmentation etc.
- baseline: predict all positive (100% recall, bad precision)

Precision, recall and threshold

h(x)	True class	TP/FP/TN/FN
0.99	1	TP
0.96	1	TP
0.87	0	FP
0.77	1	TP
0.60	1	TP
0.53	0	FP
0.48	0	TN
0.33	1	FN
0.20	0	TN
0.05	0	TN

Threshold $h(x) > 0.5$, precision 67%, recall 80%

Optimizing precision

h(x)	True class	TP/FP/TN/FN
0.99	1	TP
0.96	1	TP
0.87	0	TN
0.77	1	FN
0.60	1	FN
0.53	0	TN
0.48	0	TN
0.33	1	FN
0.20	0	TN
0.05	0	TN

Threshold $h(x) > 0.87$, precision 100%, recall 40%

Optimizing recall

h(x)	True class	TP/FP/TN/FN
0.99	1	TP
0.96	1	TP
0.87	0	FP
0.77	1	TP
0.60	1	TP
0.53	0	FP
0.48	0	FP
0.33	1	TP
0.20	0	TN
0.05	0	TN

Threshold $h(x) > 0.20$, precision 62.5%, recall 100%

ROC curves

- Medical diagnostics: classify sick (positive) vs. healthy (negative)
- True positive: sick person correctly diagnosed
- True negative: healthy person correctly diagnosed
- False positive: healthy person diagnosed as sick
- False negative: sick person diagnosed as healthy
- how should we set threshold $h(x) > t$ for sick/healthy diagnosis?

ROC curves

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

- True Positive Rate (TPR)
  - proportion of positives correctly identified as such
  - other names: recall, sensitivity
- False Positive Rate (FPR)
  - proportion of negatives incorrectly identified as positive
  - other names: 1 - specificity

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	FN		
0.96	1	FN		
0.87	0	TN		
0.77	1	FN		
0.60	1	FN		
0.53	0	TN		
0.48	0	TN		
0.33	1	FN		
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.99$, TPR 0%, FPR 0%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	FN		
0.87	0	TN		
0.77	1	FN		
0.60	1	FN		
0.53	0	TN		
0.48	0	TN		
0.33	1	FN		
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.96$, TPR 20%, FPR 0%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	TN		
0.77	1	FN		
0.60	1	FN		
0.53	0	TN		
0.48	0	TN		
0.33	1	FN		
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.87$, TPR 40%, FPR 0%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	FP	40%	20 %
0.77	1	FN		
0.60	1	FN		
0.53	0	TN		
0.48	0	TN		
0.33	1	FN		
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.77$, TPR 40%, FPR 20%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	FP	40%	20 %
0.77	1	TP	60%	20 %
0.60	1	FN		
0.53	0	TN		
0.48	0	TN		
0.33	1	FN		
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.60$, TPR 60%, FPR 20%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	FP	40%	20 %
0.77	1	TP	60%	20 %
0.60	1	TP	80%	20 %
0.53	0	TN		
0.48	0	TN		
0.33	1	FN		
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.53$, TPR 80%, FPR 20%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	FP	40%	20 %
0.77	1	TP	60%	20 %
0.60	1	TP	80%	20 %
0.53	0	FP	80%	40 %
0.48	0	TN		
0.33	1	FN		
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.48$, TPR 80%, FPR 40%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	FP	40%	20 %
0.77	1	TP	60%	20 %
0.60	1	TP	80%	20 %
0.53	0	FP	80%	40 %
0.48	0	FP	80%	60 %
0.33	1	FN		
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.33$, TPR 80%, FPR 60%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	FP	40%	20 %
0.77	1	TP	60%	20 %
0.60	1	TP	80%	20 %
0.53	0	FP	80%	40 %
0.48	0	FP	80%	60 %
0.33	1	TP	100%	60 %
0.20	0	TN		
0.05	0	TN		

Threshold $h(x) > 0.20$, TPR 100%, FPR 60%

TPR versus FPR trade-off

h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	FP	40%	20 %
0.77	1	TP	60%	20 %
0.60	1	TP	80%	20 %
0.53	0	FP	80%	40 %
0.48	0	FP	80%	60 %
0.33	1	TP	100%	60 %
0.20	0	FP	100%	80 %
0.05	0	TN		

Threshold $h(x) > 0.05$, TPR 100%, FPR 80%

TPR versus FPR trade-off

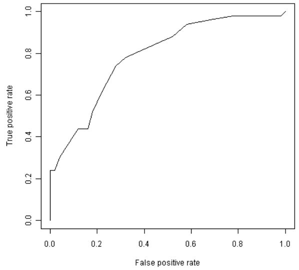
h(x)	True class	TP/FP/TN/FN	TPR	FPR
0.99	1	TP	20%	0%
0.96	1	TP	40%	0 %
0.87	0	FP	40%	20 %
0.77	1	TP	60%	20 %
0.60	1	TP	80%	20 %
0.53	0	FP	80%	40 %
0.48	0	FP	80%	60 %
0.33	1	TP	100%	60 %
0.20	0	FP	100%	80 %
0.05	0	FP	100%	100 %

Threshold $h(x) > 0.00$, TPR 100%, FPR 100%

ROC curves

The ROC curve (receiver operating characteristic curve) draws the true positive rate (TPR) against the false positive rate (FPR) illustrating the performance of a binary classifier.

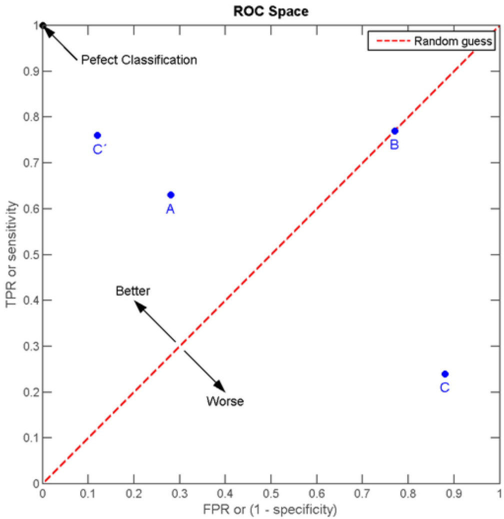
- A ROC space is defined by FPR and TPR as x and y axes respectively, which depicts relative trade-offs between TP (benefits) and FP (costs).



ROC curves

- The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space representing 0% FPR (no false negatives) and 100% TPR (no false positives).
- The (0,1) point is also called a perfect classification.
- A completely random guess would give a point along a diagonal line (the so-called line of no-discrimination) from the left bottom to the top right corners. An intuitive example of random guessing is a decision by flipping coins (head or tail).

ROC curves


ROC Space
Random guess
Pefect Classification
C′
A
B
C
Better
Worse
TPR or sensitivity
FPR or (1 - specificity)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

ROC curves

- The area under curve (AUC), i.e. the area under the ROC curve, is an indicator how well the classifier solves the problem. The larger the area, the better the solution for the classification problem.
- traditionally used in medicine for comparing diagnostic tests
- increasingly popular in data mining / machine learning
- compares two classifiers over all possible thresholds simultaneuosly
- invariant to relative class distributions
- useful when the proper misclassification costs are unknown

Area under ROC curve

$$\frac{1}{|X_+||X_-|}\sum_{x_i \in X_+}\sum_{x_j \in X_-} H(h(x_i) - h(x_j)),$$

$$H(a) = \begin{cases} 1, & \text{if } a > 0 \\ 1/2, & \text{if } a = 0 \\ 0, & \text{if } a < 0 \end{cases}.$$

- AUC: probability, that randomly chosen positive example ranked higher, than randomly chosen negative one
- similar to ranking based correlation measures considered earlier

Notation

h	classifier function
x_i	i:th training instance
X_+	set of positive instances
X_-	set of negative instances

Area under ROC curve

- AUC produces a value that is between 0 and 1
- 1.0 perfect ranking, all instances of positive class receive higher predicted values than those of negative class
- 0.5 random performance, you can get the same AUC by flipping a coin or predicting a constant value always
- sanity check for binary classifier: is AUC (much) higher than 0.5? If not, you have not captured any meaningful pattern.
- (on most real problems AUCs very close to 1.0 also suspicous, usually 'too good to be true', sign that you have messed up somewhere in your evaluation protocol, e.g. overfitting and then testing on training data...)

Advantages of AUC

- ”stupid” random or constant prediction baselines always have 0.5 AUC
- does not require specifying decision threshold or class specific error costs
- simple interpretation: probability of being able to distinguish positive instance from negative
- invariant to relative class distributions
  - other things being equal, we would expect same AUC for imbalanced test set with 90% positives and 10% negatives as for balanced 50%-50% class distribution
  - most performance measures don’t behave like this!

Practical advice

- AUC is computed from decision function values or predicted probabilities
- if method implements both AUC results should be same for both, since they should provide same ordering of data
- binary classifier with scikit-learn:
  - `learner.predict(X)` returns 0/1 or -1/1 values (do not use these values)
  - `decision_function(X)` real value
  - `predict_proba(X)` class probabilities
- classical mistake: use discrete class predictions for computing AUC $\rightarrow$ code will not crash, but results wrong (lower AUC)