Introduction to Machine Learning
NPFL 054

http://ufal.mff.cuni.cz/course/npfl054

Barbora Hladká
hladka@ufal.mff.cuni.cz

Martin Holub
holub@ufal.mff.cuni.cz

Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

Outline

- **Basics of classifier evaluation**
  - why we need evaluation
  - working with data
  - cross-validation process
  - leave-one-out method
  - a bootstrap heuristic
  - baseline classifier
  - evaluation metrics for the binary case

Fundamentals of classifier evaluation

You need thorough evaluation to

1. **get a reliable estimate of the classifier performance**
   - i.e. how it will perform on new – so far unseen – data instances
   - possibly even in the future

2. **compare your different classifiers** that you have developed
   - to decide which one is "the best"

= Model assessment and selection

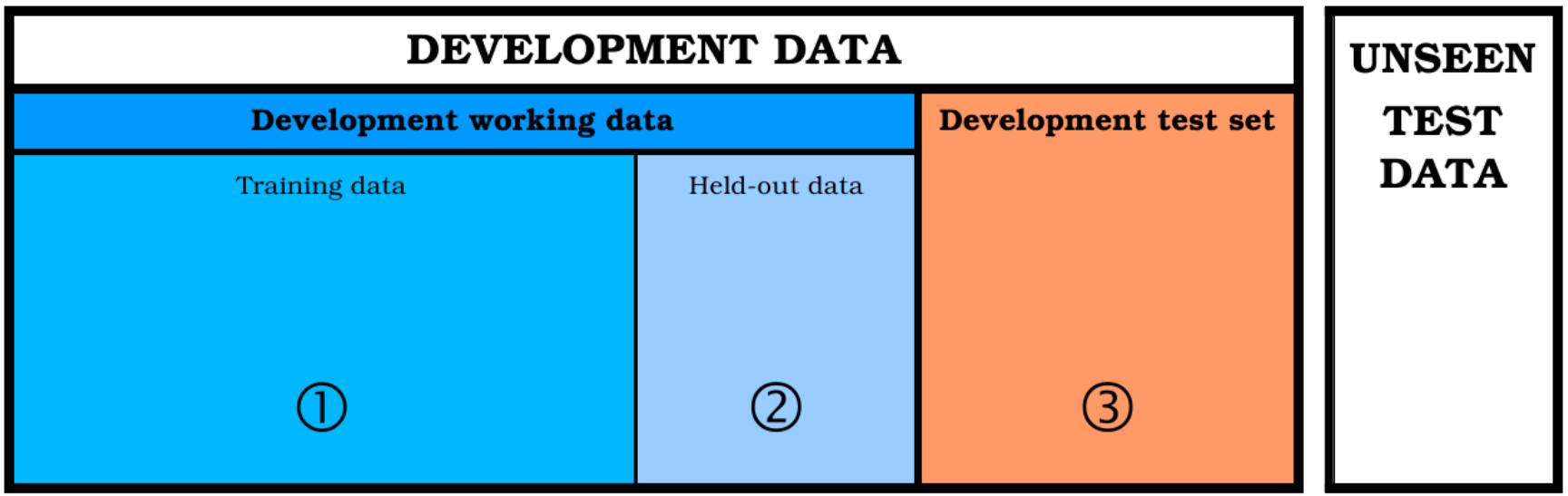
Model robustness

You need *good* performance

not only on *your* data,

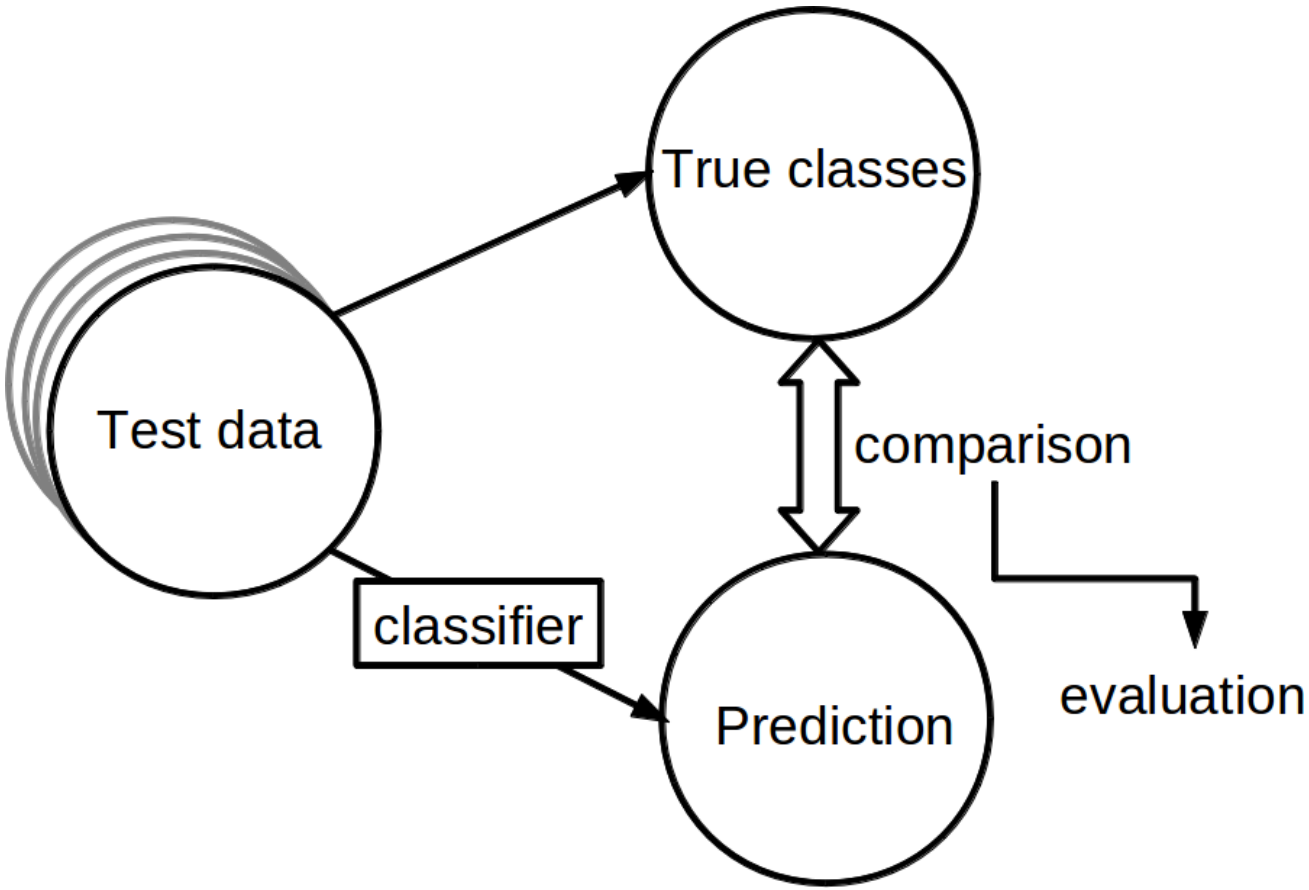
but also on any data that can be *expected*!

Working with data
Development data and its division



All subsets should be selected randomly in order to represent the characteristic distribution of both feature values and target values in the available set of examples.

Evaluation – basic scheme



Development data – the working portion

Development working data

Is used both for training your classifier and for evaluation when you tune the learning parameters.

- **Training data**
  is used for **training** your classifier with a particular learning parameter settings when you tune your classifier

- **Held-out data**
  is used for **evaluating** your classifier with a particular learning parameter settings when you tune your classifier

Development data – the test portion

Development test set

- the purpose is to simulate the "real" test data
- should be used only for your final development evaluation when your classifier has already been tuned and your learning parameters are finally set
- using it you get an estimate of your classifier's performance at the end of the development
- is also used for model selection

Using bigger training sets

Generally, whenever you extend your training data, you should get a better classifier!

Using bigger training sets

Generally, whenever you extend your training data, you should get a better classifier!

If not, there may be a problem

- either with your data
  - e.g. noise data or not representative data
  - distortion of statistical characteristics
- or with your method/model
  - e.g. bad settings of learning parameters

Using bigger training sets

Generally, whenever you extend your training data, you should get a better classifier!

If not, there may be a problem

- either with your data
  - e.g. noise data or not representative data
  - distortion of statistical characteristics
- or with your method/model
  - e.g. bad settings of learning parameters

– **Sometimes,** you cannot get better results because the performance is already stable/maximal. However, even in this case using more training data should imply better robustness.

Using different training sets

1. When you tune your classifier you split your development working set and use only the "training portion" to train your classifier. You always hold out some data for classifier evaluation.

   In this phase you can do cross-validation, bootstrapping, or any other tricks. – Will be discussed later.

2. When you have your classifier tuned, keep the best parameters. Then use all "development working" portion as training data to make the best model.

3. Finally – after model selection – use all your development data as a training set to train the best model you are able to develop.

   This model can be later evaluated on the "unseen test" data (which is NOT a developer's job!).

Sample error measuring

Definition (Empirical and sample error)

Given a sample set S, the **empirical error** (aka *observed error*) of classifier $\hat{f}$ is the observed number of errors that $\hat{f}$ does on S.

The **sample error** of hypothesis $\hat{f}$ with respect to target function f and data sample S is the proportion of examples that $\hat{f}$ misclassifies

$$\text{error}_S = \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq \hat{f}(x))$$

where

- $n = |S|$ is the sample size
- $f(x)$ is the true classification of example x
- $\hat{f}(x)$ is the predicted class of example x
- $\delta(f(x) \neq \hat{f}(x))$ is 1 if $f(x) \neq \hat{f}(x)$, and 0 otherwise.

Estimating "true error"

Definition (Generalization error)

The **generalization error** (aka *true error*) of hypothesis $\hat{f}$ with respect to target function f and distribution $\mathcal{D}$ is the probability that $\hat{f}$ will misclassify an instance drawn randomly according to $\mathcal{D}$.

$$\text{error}_{\mathcal{D}} = \Pr_{x \in \mathcal{D}} \left\{ \delta(f(x) \neq \hat{f}(x) \right\}$$

Estimating "true error"

Definition (Generalization error)

The **generalization error** (aka *true error*) of hypothesis $\hat{f}$ with respect to target function f and distribution $\mathcal{D}$ is the probability that $\hat{f}$ will misclassify an instance drawn randomly according to $\mathcal{D}$.

$$\text{error}_{\mathcal{D}} = \Pr_{x \in \mathcal{D}} \left\{ \delta(f(x) \neq \hat{f}(x) \right\}$$

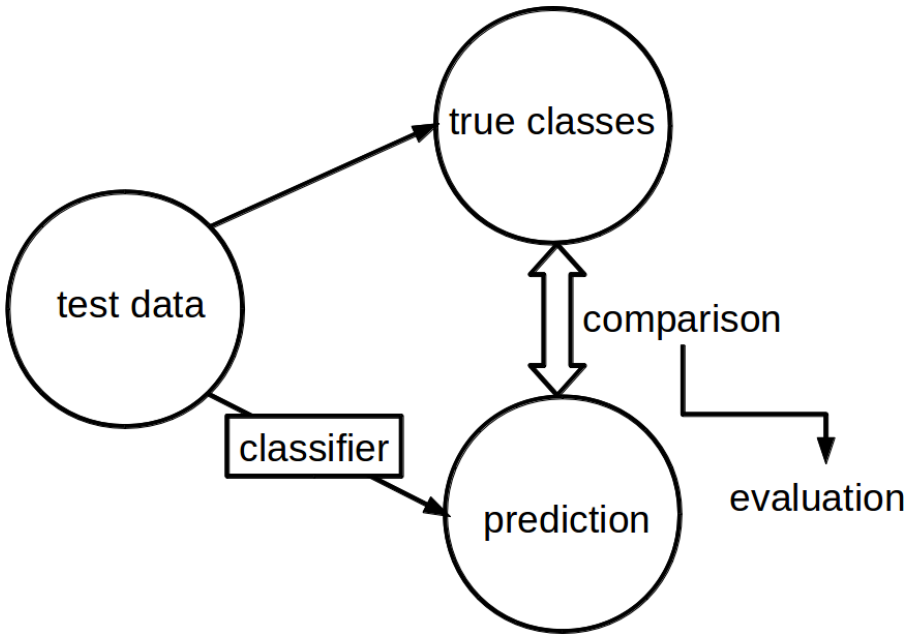
Generalization error – how to estimate?
Typically, the generalization error is not an observable quantity because the distribution $\mathcal{D}$ is usually unknown.

$\longrightarrow$ **The question is**

How well does error_S estimate $\text{error}_{\mathcal{D}}$?

The need of data for the evaluation process

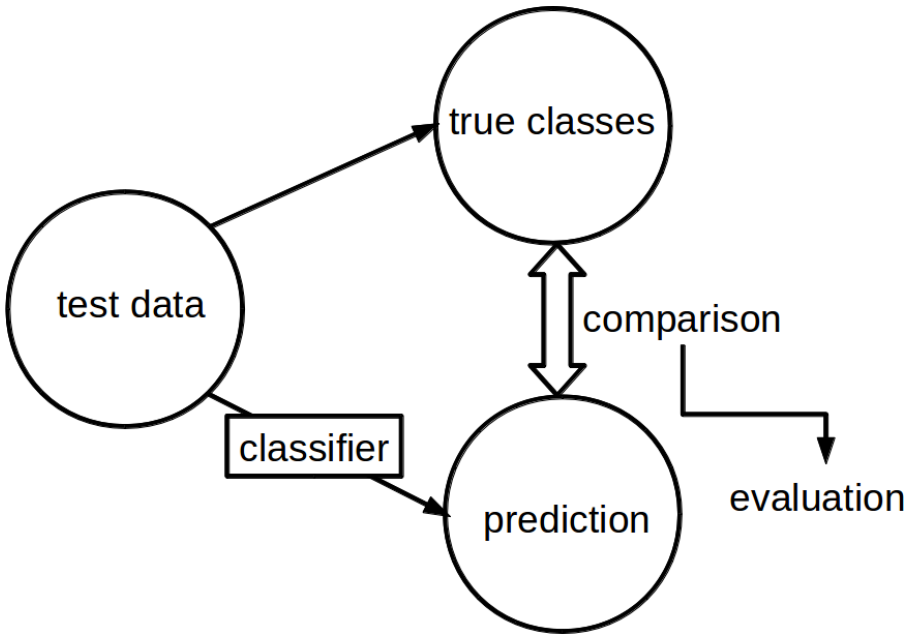
Evaluation process



Is it enough to test your classifier on one test set?

The need of data for the evaluation process

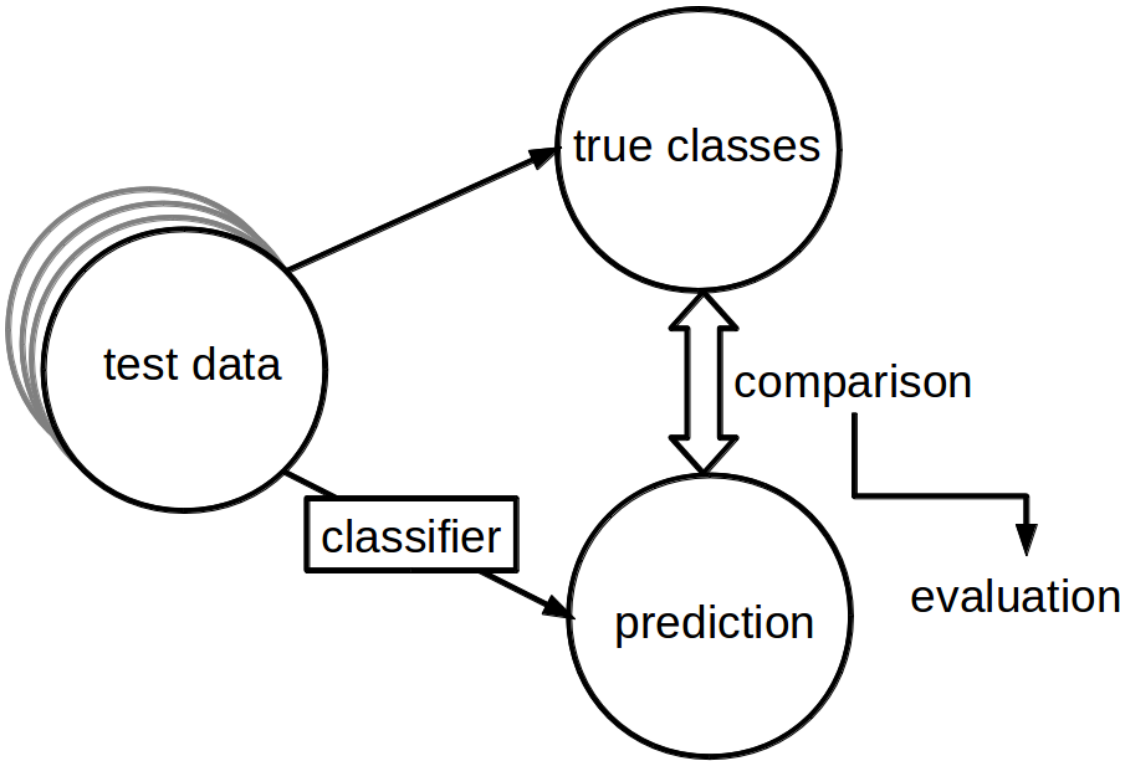
Evaluation process



Is it enough to test your classifier on one test set?
You can get a good/bad result by chance!

The ideal evaluation

The more test data, the more confident evaluation . . .



Using a test set

- **Purpose** – How well will your classifier perform on novel data?
  – We can **estimate** the performance of the classifier using a test data set.
  And we do NOT have any better chance to get reliable estimate!

- Performance on the training data is not a good indicator of performance on future data.
  – You would easily **overestimate**!

- **Important!** – You should NOT have any **look** at your test data during the development phase!
  – Test set = independent instances that have NOT been used
  in **any way** to create the classifier.

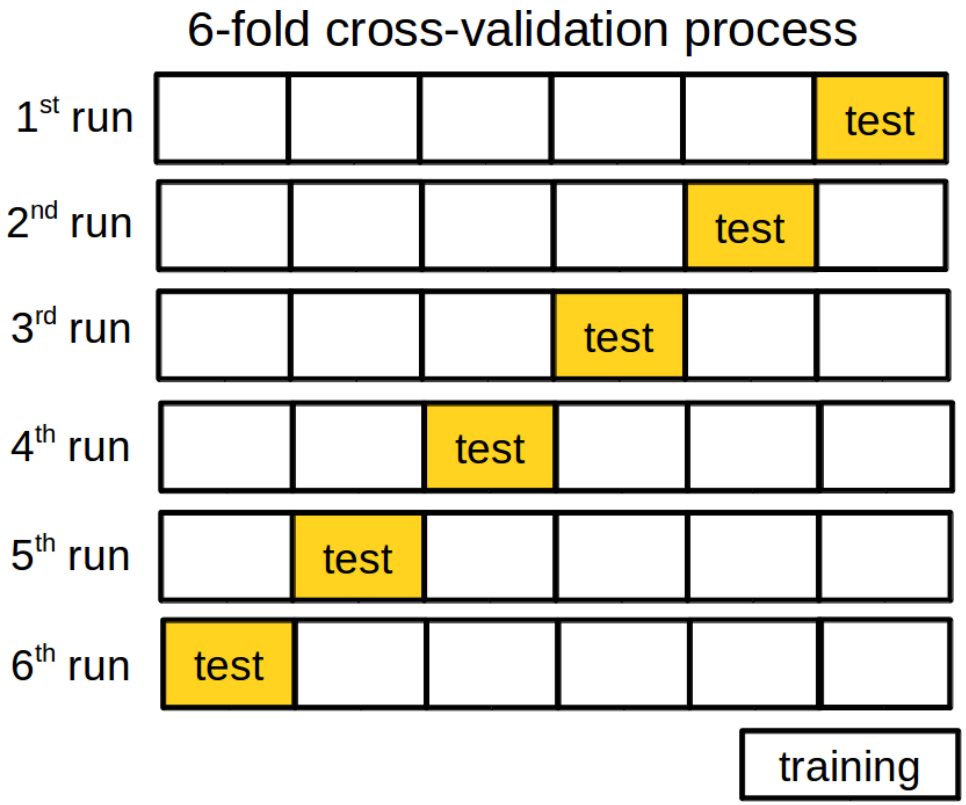
- **Assumption** – Both training data and test data are representative samples of the underlying problem!

K-fold cross-validation process

Development working data is partitioned into k **subsets** of equal size. Then you do k iterations.

In the i-th step of the iteration, the i-th subset is used as a test set, while the remaining parts form the training set.

Example



Using k-fold cross-validation: Which k is the best?

The goal: get a good estimate of generalization error

$\Rightarrow$ low bias: not to underestimate, nor overestimate

$\Rightarrow$ low variance: low sensitivity to the data sample

Small k (k close to 2)

$\longrightarrow$ small training sets, error rate tends to be overestimated

Large k (up to the data set size)

$\longrightarrow$ could be computationally demanding = main practical problem

$\longrightarrow$ small test sets, training sets are almost identical

$\longrightarrow$ low bias, but high variance

Heuristic recommendation: cca $5 \leq k \leq 10$

$\longrightarrow$ moderate bias, moderate variance, moderate computational cost

$\longrightarrow$ has been empirically shown to yield good error rate estimates

Stratified cross-validation – each class should be represented in roughly the same proportion as in the entire data set

$\longrightarrow$ if data sets are small, the risk of purely ramdom split should be avoided

Leave-one-out cross-validation (LOOCV)

Extreme case: $k = n$, where n is the size of the development data set
$\longrightarrow$ leave-one-out method (LOOCV)

Advantages

- using maximum training sets $\rightarrow$ low bias
- no randomness in data splitting

Disadvantages

- training sets are almost identical(!) $\rightarrow$ high variance of the estimate
  – variance is high, because LOOCV averages the outputs of n models that are highly positively correlated with each other
  – high variance is the reason why k-fold CV with moderate k often gives more accurate estimates of the test error than does LOOCV
- may be (typically) too time-consuming
- similar class distribution in training and test data is not guaranteed
  – The extreme case: 50 % class A, 50 % class B. Then the trivial MFC classifier has true error 50 %, BUT the LOOCV error estimate is 100 % (!).

Recommended evaluation procedure

Typically, use k-fold cross-validation for $k = 5$ or $k = 10$ for estimating the performance (accuracy, etc.)

Then compute

- the mean value of performance estimate
- standard deviation
- confidence intervals

Report mean values of performance estimates and their standard deviations, or (better) 95 % confidence intervals around the mean.

A simple sampling method

Motivation: When the total number of examples is very small (≤ 50), even the leave-one-out method becomes unreliable.

- repeat 2-fold cross-validation (e.g. 100 times)
- it has been shown that the average quality of estimation is better than the leave-one-out method

Bootstrapping principle

Bootstrap sampling – generating different training subsets

- New data sets $D_1, \ldots, D_K$ are drawn from an original data set D *with replacement*, each of the same size as the original $|D| = n$.

- Then in the i-th step of the iteration, D_i is used as a training set, while all the other examples $\mathbf{x} \in D \setminus D_i$ form the actual test set.

Bootstrapping principle

Bootstrap sampling – generating different training subsets

- New data sets $D_1, \ldots, D_K$ are drawn from an original data set D *with replacement*, each of the same size as the original $|D| = n$.

- Then in the i-th step of the iteration, D_i is used as a training set, while all the other examples $\mathbf{x} \in D \setminus D_i$ form the actual test set.

How many examples will appear in the bootstrap samples?

Bootstrapping principle

Bootstrap sampling – generating different training subsets

- New data sets $D_1, \ldots, D_K$ are drawn from an original data set D *with replacement*, each of the same size as the original $|D| = n$.

- Then in the i-th step of the iteration, D_i is used as a training set, while all the other examples $\mathbf{x} \in D \setminus D_i$ form the actual test set.

How many examples will appear in the bootstrap samples?

- The probability that we pick an instance is $1/n$, and the probability that we do not pick an instance is $1 - 1/n$. The probability that we do not pick an instance after n draws is $(1 - 1/n)^n \approx e^{-1} \doteq 0.368$.

- It means that for training the system will not use 36.8 % of the data, and thus the error estimate will be rather pessimistic.

A simple bootstrap heuristic

- Suppose a development data set of n examples
- An optimistic error rate e_ℓ of the model is obtained by building and testing on all available examples
  - Train a model using all n examples

  – Get training error $=$ optimistic estimate e_ℓ.

A simple bootstrap heuristic

- Suppose a development data set of n examples
- An optimistic error rate e_ℓ of the model is obtained by building and testing on all available examples
  - Train a model using all n examples

  – Get training error = optimistic estimate e_ℓ.
- A pesimistic error rate e_0 is obtained by making 200 bootstrap samples
  - Randomly select n examples with replacement and train a model
    – on average, it will be 63.2 % of the original set
  - Test the model on the examples not used in the training
    – on average, it will be 36.8 % of the original set
  - Get the test error

  – Get mean test error as an average quality = pesimistic estimate e_0.

A simple bootstrap heuristic

- Suppose a development data set of n examples
- An optimistic error rate e_ℓ of the model is obtained by building and testing on all available examples
  - Train a model using all n examples

  – Get training error = optimistic estimate e_ℓ.
- A pesimistic error rate e_0 is obtained by making 200 bootstrap samples
  - Randomly select n examples with replacement and train a model
    – on average, it will be 63.2 % of the original set
  - Test the model on the examples not used in the training
    – on average, it will be 36.8 % of the original set
  - Get the test error

  – Get mean test error as an average quality = pesimistic estimate e_0.
- Finally, the error ".632 estimator" is defined as a linear combination

$$e = 0.368 \cdot e_\ell + 0.632 \cdot e_0$$

Notes

The .632 estimator can break down in overfitting situations (when e_ℓ is close to 0).

The error estimation obtained by a hundred 2-fold CV runs may be used instead of e_0.

Baseline classifier

The most trivial baseline classifier is the classifier that always gives the most frequent class (sometimes called the MFC classifier).

Your classifier should never be worse than that baseline :–)

Baseline classifier

The most trivial baseline classifier is the classifier that always gives the most frequent class (sometimes called the MFC classifier).

Your classifier should never be worse than that baseline :–)

More practical/realistic baseline

The trivial MFC baseline should always be considered. However, usually another (better) simple classifier (e.g. with a default settings of learning parameters) is considered to be a baseline.

– Then you compare your developed classifiers to that "real baseline".

Evaluation of binary classifiers
Binary confusion matrix

Binary classification $\overset{aka}{=}$ **2-class classification** $\overset{aka}{=}$ **0/1 classification**

In binary classification tasks, examples are divided into two disjoint subsets:

- **positive examples** – "to be retrieved" (ones)
- **negative examples** – "not to be retrieved" (zeros)

```
# Example of confusion matrix for binary classification
> table(cv.test$Class, pred.test)
       prediction
          0   1
 true  0 580  69
       1  37 144
>
```

Evaluation of binary classifiers
Confusion matrix

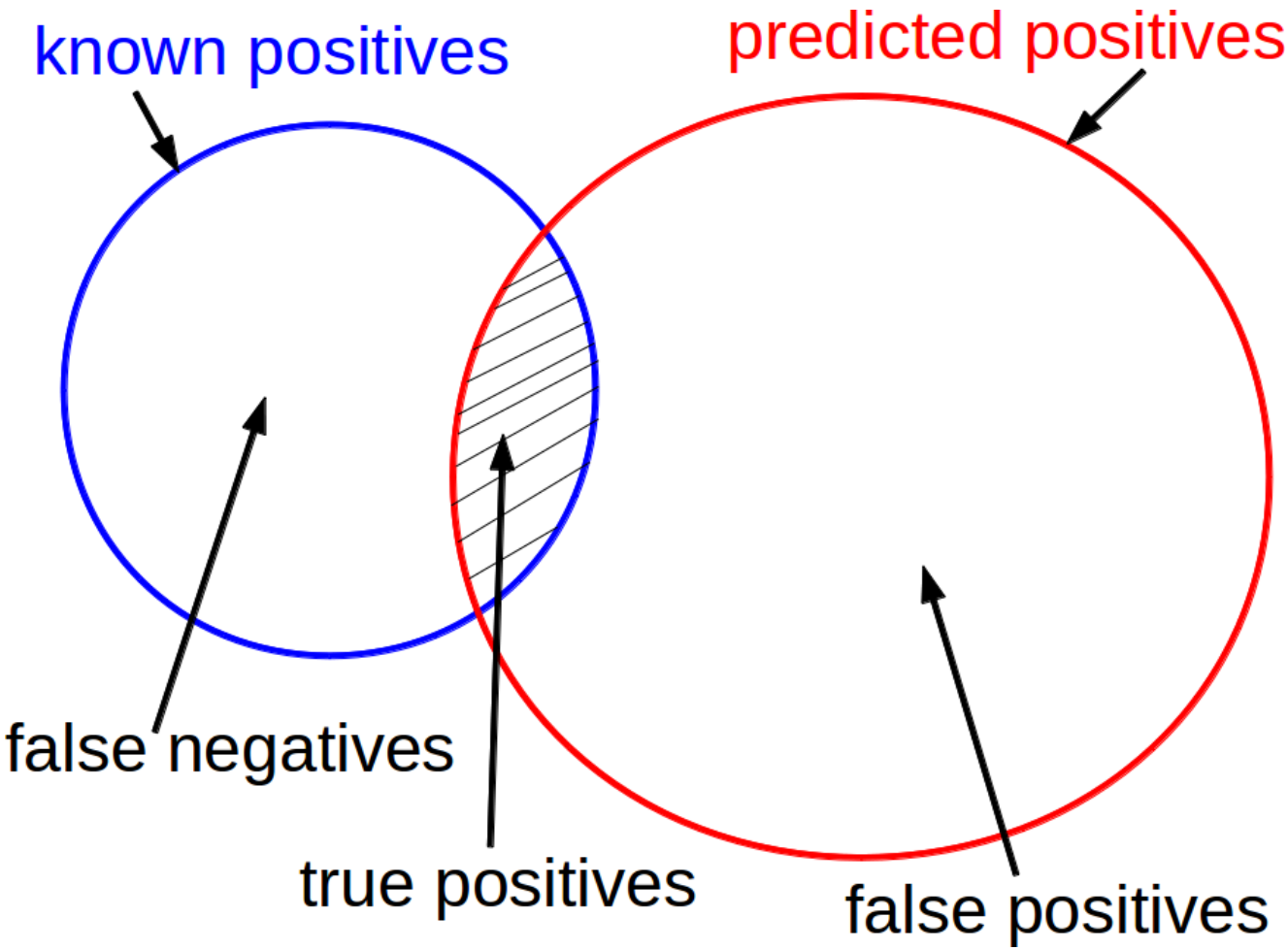
		Predicted class	
		Positive	**Negative**
True class	**Positive**	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Evaluation of binary classifiers
Confusion matrix

		Predicted class	
		Positive	**Negative**
True class	**Positive**	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

- **'Trues'** are examples correctly classified
- **'Falses'** are examples incorrectly classified
- **'Positives'** are examples predicted as positives (correctly or incorrectly)
- **'Negatives'** are examples predicted as negatives (correctly or incorrectly)

Evaluation of binary classifiers
Confusion matrix



Evaluation of binary classifiers
Basic performance measures

Measure	Formula
Precision	TP/(TP+FP)
Recall/Sensitivity	TP/(TP+FN)
Specificity	TN/(TN+FP)
Accuracy	(TP+TN)/(TP+FP+TN+FN)

Very often you need to **combine both good precision and good recall**. Then you usually use **balanced F-score**, so called **F-measure**

$$F = 2\,\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$