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

Lecture #3
Part I — Fundamentals of Classifier Evaluation

Outline

- **Basics of classifier evaluation**
  - why we need evaluation
  - working with data
  - cross-validation process
  - baseline classifier
  - confusion matrix, accuracy, error rate
  - evaluation metrics for the binary case
  - sample error and generalisation error
  - overfitting

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

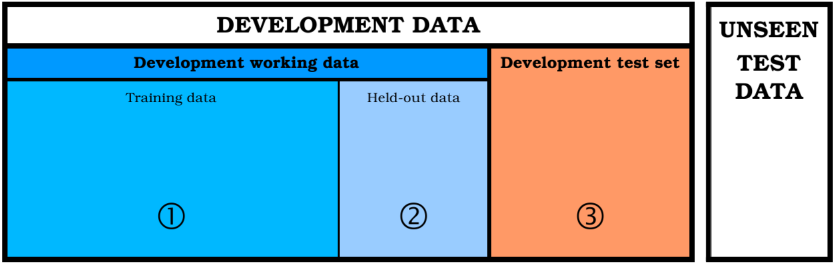
Model robustness

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



DEVELOPMENT DATA

Development working data

Training data

①

Held-out data

②

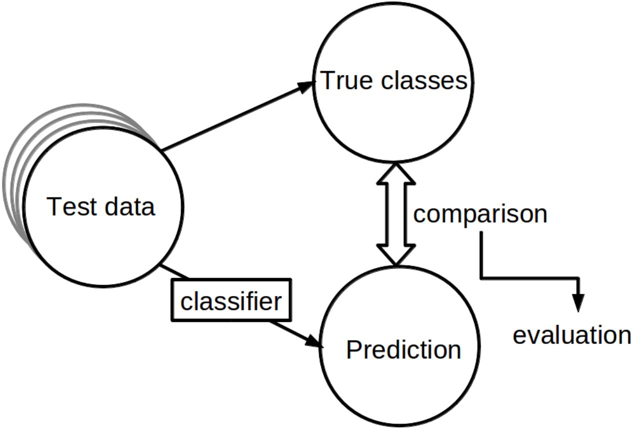
Development test set

③

UNSEEN TEST DATA

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.

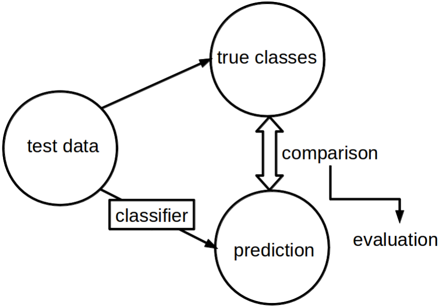
2. When you have your classifier tuned, keep the best parameters. Then use all "development working" portion as training data to make the best model.

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!).

The need of data for the evaluation process

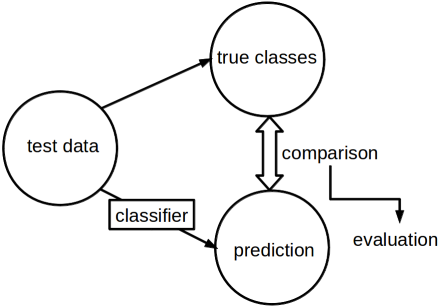
Evaluation process



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

The need of data for the evaluation process

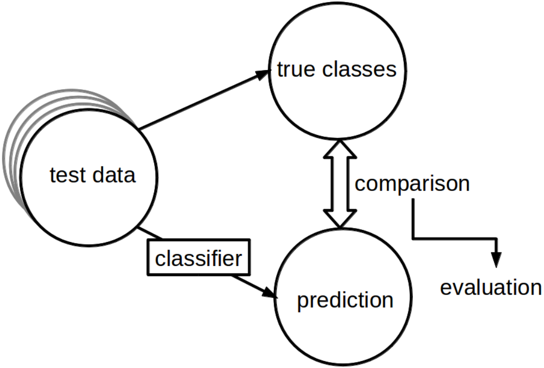
Evaluation process



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 . . .

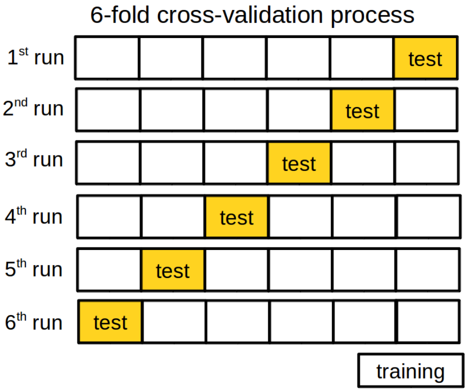


K-fold cross-validation process

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



6-fold cross-validation process

1st run						test
2nd run					test	
3rd run				test		
4th run			test			
5th run		test				
6th run	test					

training

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!

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".

Confusion matrix

Confusion matrix is a square matrix indexed by all possible target class values.

```
** Comparing the predicted values with the true senses -- M3 **

           Prediction
Truth       cord division formation phone product text
  cord       268        3        10     7       9    6
  division     3      280         1     2       5    3
  formation   13        3       225     4      19    4
  phone       25        5         2   293      12   10
  product     51       10        39    32    1442   72
  text        12        1         7     4      28  262
```

Correctly predicted examples are displayed on the diagonal.

Sample accuracy and sample error rate

To measure the performance of classification tasks we often use (sample) *accuracy* and (sample) *error rate*

Sample accuracy is the number of correctly predicted examples divided by the number of all examples in the predicted set

Sample error rate is equal to **1 - accuracy**

Sample accuracy and sample error rate

To measure the performance of classification tasks we often use (sample) *accuracy* and (sample) *error rate*

Sample accuracy is the number of correctly predicted examples divided by the number of all examples in the predicted set

Sample error rate is equal to **1 - accuracy**

Training error rate is the sample error rate measured on the training data set

Test error rate is the sample error rate measured on the test data set

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

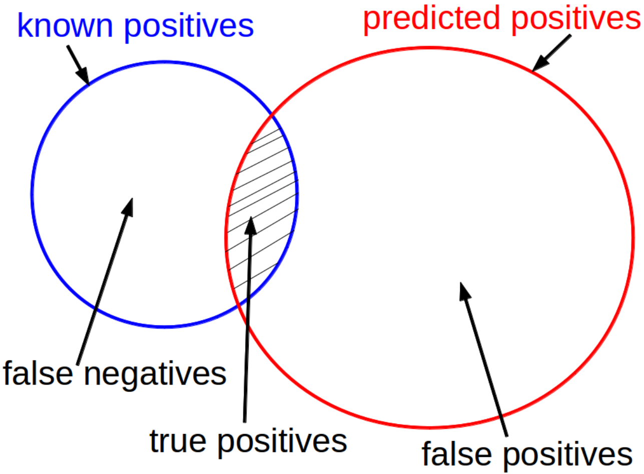
		Predicted class	
		Positive	**Negative**
True class	**Positive**	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

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**

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

Sample error and generalization error

Sample error of a hypothesis h with respect to a data sample S of the size n is usually measured as follows

- for **regression**: **mean squared error** $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
- for **classification**: **classification error** $= \frac{1}{n}\sum_{i=1}^{n}\mathrm{I}(\hat{y}_i \neq y_i)$

Sample error and generalization error

Sample error of a hypothesis h with respect to a data sample S of the size n is usually measured as follows

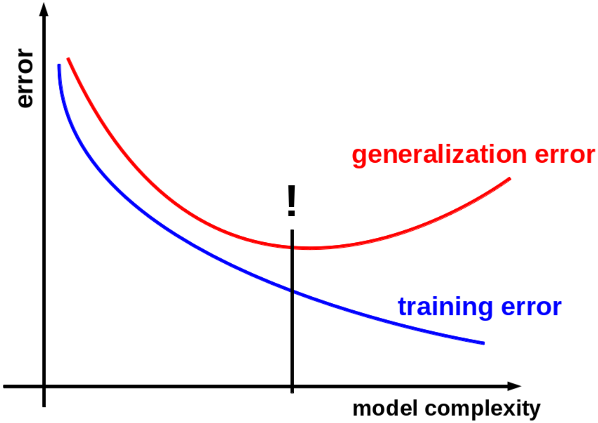
- for **regression**: **mean squared error** $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
- for **classification**: **classification error** $= \frac{1}{n}\sum_{i=1}^{n}\mathrm{I}(\hat{y}_i \neq y_i)$

Generalization error (aka "true error" or "expected error") measures how well a hypothesis h generalizes beyond the used training data set, to unseen data with distribution $\mathcal{D}$. Usually it is defined as follows

- for **regression**: $\mathrm{error}_{\mathcal{D}}(h) = \mathsf{E}\,(\hat{y}_i - y_i)^2$
- for **classification**: $\mathrm{error}_{\mathcal{D}}(h) = \mathsf{Pr}\,(\hat{y}_i \neq y_i)$

Minimizing generalization error vs. overfitting

Finding a model that minimizes generalization error
... is one of central goals of the machine learning process


error
generalization error
!
training error
model complexity