Introduction to Machine Learning
NPFL 054

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

Barbora Hladká **Martin Holub**

{Hladka | Holub}@ufal.mff.cuni.cz

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

Outline — Lab session #2

- **Remarks on entropy**
  - Tutorial on probability distributions and entropy in R
- **Remarks on Decision Trees and evaluation**
  - Recap of the Decision Trees learning algorithm
  - Fundamental principles of evaluation
- **Exercises in R**
- **Homework**

Conditional probability Pr(SENSE | A19)

Distribution of the A19 feature

```
> table(examples$A19)

 line lined lines
 2392     2  1130
```

Contingency table

```
> table(examples$A19, examples$SENSE)

       cord division formation phone product text
 line   226      250       189   201    1319  207
 lined    0        0         2     0       0    0
 lines  110       72       105   179     519  145
```

Sum of rows

```
> margin.table(table(examples$A19, examples$SENSE), 1)

 line lined lines
 2392     2  1130
>
```

Conditional probability Pr(SENSE | A19)

Conditional probability – percentage

```
> for(i in 1:3) {
+   print(round(
+     table(examples$A19, examples$SENSE)[i,] /
+     margin.table(table(examples$A19, examples$SENSE), 1)[i] * 100,
+     digits=2
+   ))
+ }
     cord  division formation     phone   product      text
     9.45     10.45      7.90      8.40     55.14      8.65
     cord  division formation     phone   product      text
        0         0       100         0         0         0
     cord  division formation     phone   product      text
     9.73      6.37      9.29     15.84     45.93     12.83
>
```

Computing conditional entropy in R

You can use contingency table

```
##### computing H( C | A4 )

> examples <- read.table("wsd.development.csv", header=T)

> p.A4 <-
    table(examples$A4)/nrow(examples)

> p.joint.sense.A4 <-
    as.vector(table(examples$A4, examples$SENSE))/nrow(examples)

> p.cond.sense.given.A4 <-
    p.joint.sense.A4 / rep(p.A4, 6)

> H.cond.sense.given.A4 <-
    - sum( p.joint.sense.A4[p.joint.sense.A4 > 0]
           * log2( p.cond.sense.given.A4[p.cond.sense.given.A4 > 0] ) )

> H.cond.sense.given.A4
[1] 1.95903
>
```

Explanation of the code

The formula for conditional entropy

$$\mathrm{H}(\texttt{SENSE} \mid \texttt{A4}) \quad = \quad - \sum_{y \in \texttt{SENSE}, x \in \texttt{A4}} \Pr(y, x) \cdot \log_2 \frac{\Pr(y, x)}{\Pr(x)}$$

Vectors in the code

- `p.joint.sense.A4` stands for $\Pr(y, x)$
- `p.cond.sense.given.A4` stands for $\dfrac{\Pr(y, x)}{\Pr(x)}$
- then `H.cond.sense.given.A4` computes the sum

Conditional entropy in R – `entropy.cond()`

```
entropy <- function(x){
  # computes H(x)
  # expects a factor

  p <- table(x) / NROW(x)
  return( -sum(p * log2(p)) )
}

entropy.cond <- function(x, y){
  # computes H(x|y)
  # expects two factors of *the same length*

  N       <- NROW(x)                                  # should be equal to NROW(y)!
  p.y     <- table(y) / N                             # p(y)
  p.joint <- as.vector(table(y, x)) / N               # p(y,x)
  p.cond  <- p.joint / rep(p.y, NROW(table(x)))       # p(x|y) = p(y,x) / p(y)
  H.cond  <- - sum( p.joint[p.joint > 0] * log2(p.cond[p.cond > 0]) )  # H(x|y)
  return( H.cond )
}
```

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

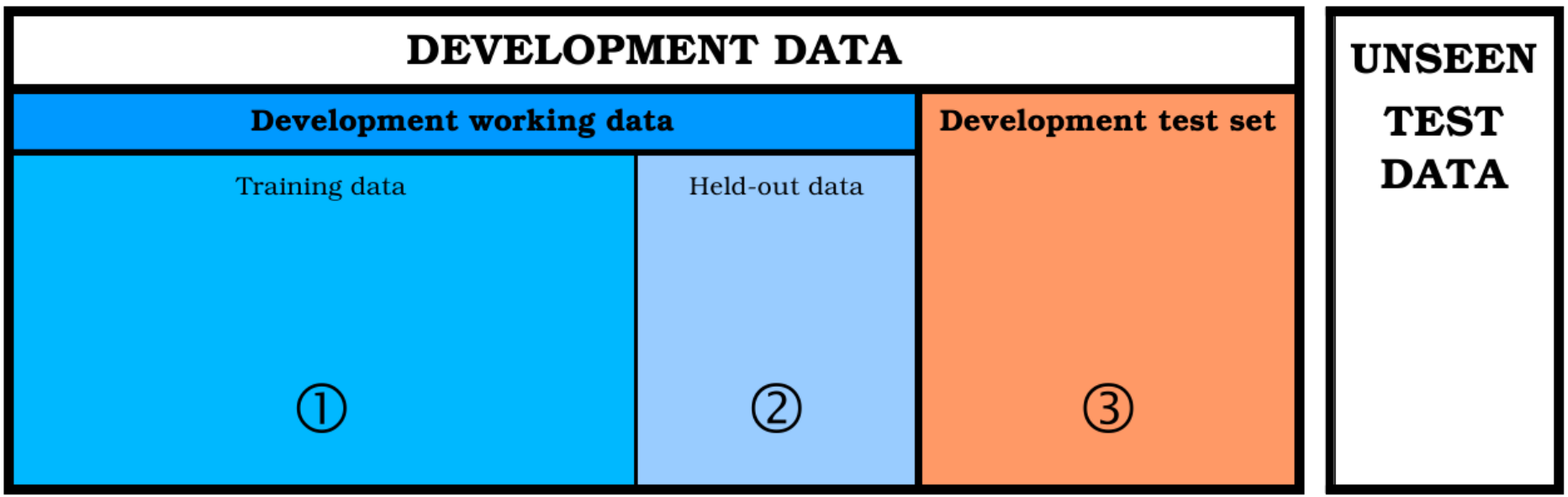
You need *good* performance

not only on *your* data,

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

Working with data

Working with data



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.

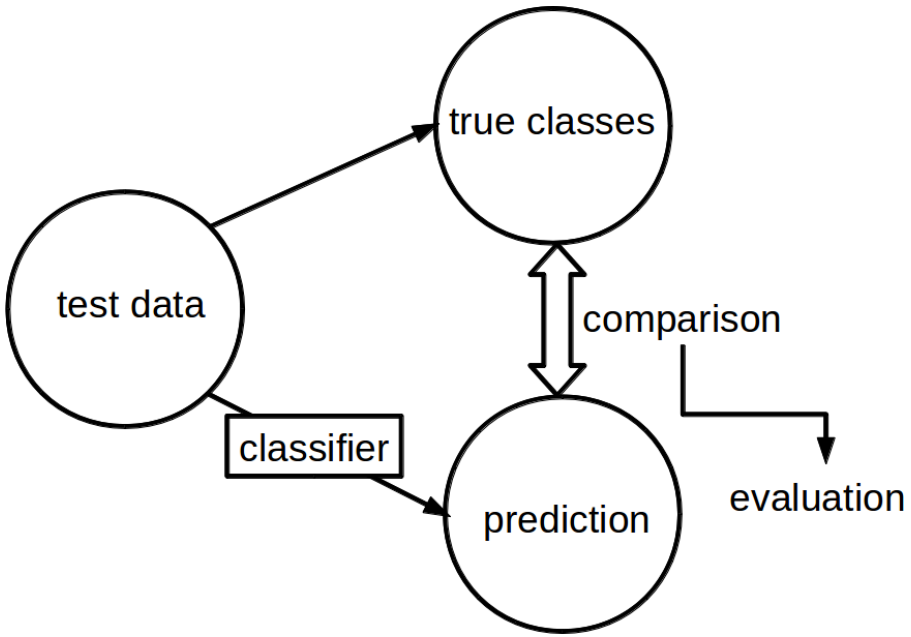
2. When you have your classifier tuned, keep the best parameters. Then use all "development working" portion as training data to make the best model.

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

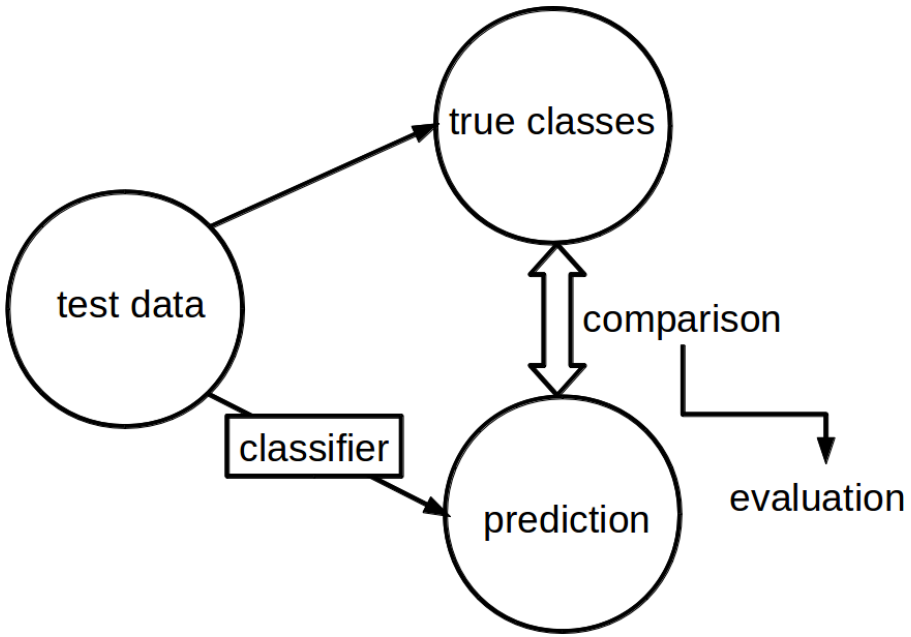
Evaluation process



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

The need of data for the evaluation process

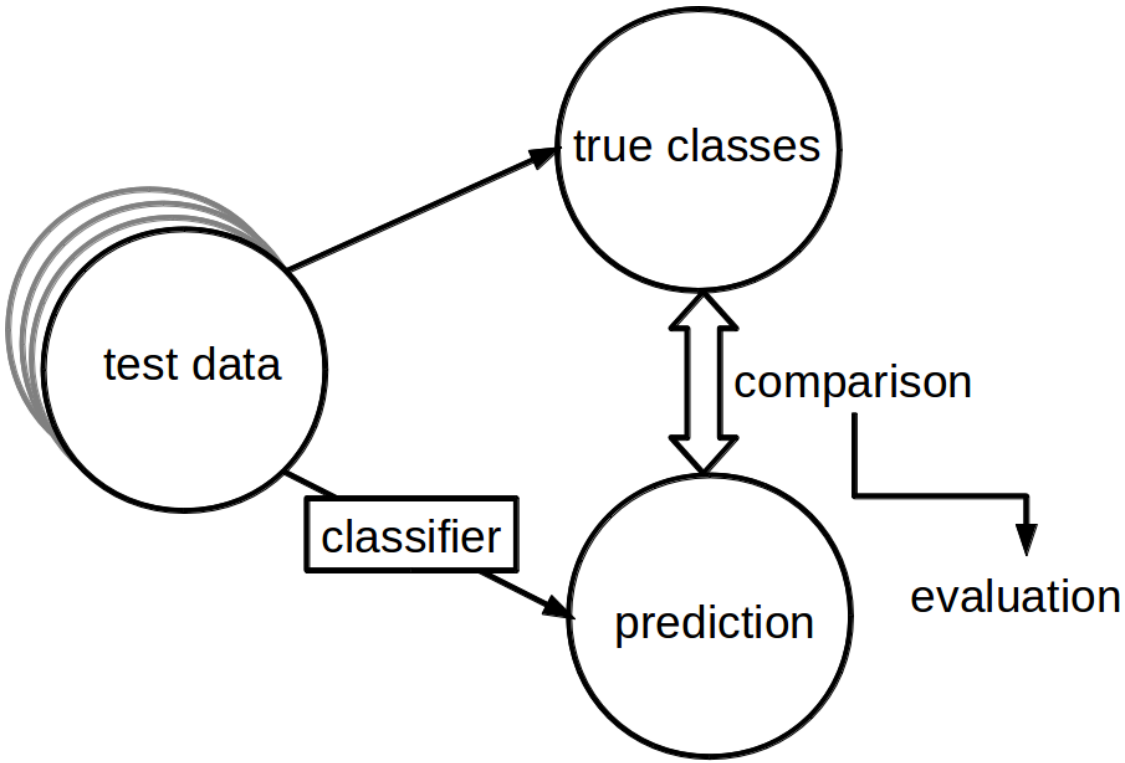
Evaluation process



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 . . .

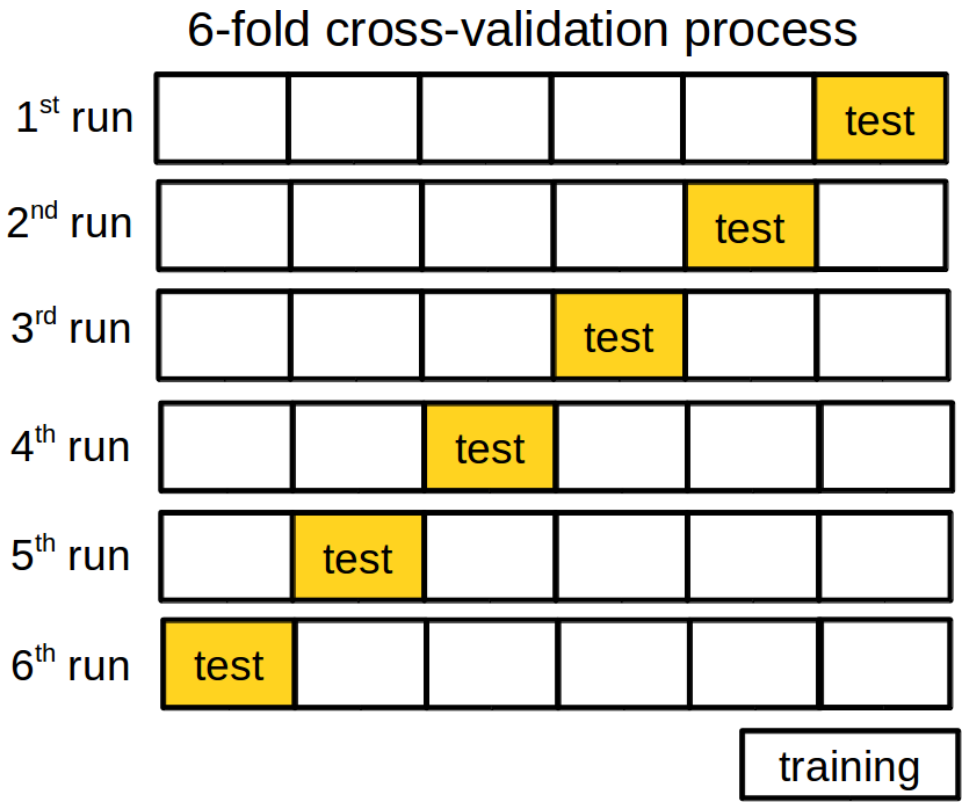


K-fold cross-validation process

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



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.

Accuracy and error rate

Accuracy

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

Error rate

is equal to **1** - **accuracy**

Homework

- Go through all details in the posted *Tutorial on distributions and entropy* – and make sure that you are able to use R and do the exercises

- Run and study example code `cp-and-pruning.Forbes.R`