A Gentle Introduction
to Machine Learning
in Natural Language Processing using R

ESSLLI ’2013
Düsseldorf, Germany

http://ufal.mff.cuni.cz/mlnlpr13

Barbora Hladká
hladka@ufal.mff.cuni.cz

Martin Holub
holub@ufal.mff.cuni.cz

Charles University in Prague,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics
• 3.1 Formal foundations of ML
• 3.2 Naive Bayes learning – Theory
• 3.3 Naive Bayes learning – Practice
• 3.4 Evaluation of a classifier
• Summary
Machine learning process – five basic steps

1. Formulating the task
2. Getting both training and test data
3. Building classifier using training data
4. Prediction on test data
5. Evaluation

Classifier to use
Formulating the task

1. **Task description**
   WSD: Assign the correct sense to the target word "line"
   COL: Decide whether the given word pair forms a semantic collocation

2. **Object specification**
   WSD: Sentences containing the target word
   COL: Word pairs

3. **Specification of target class $C$ and its values $y_1, y_2, \ldots, y_k$**
   WSD: $\text{SENSE} = \{\text{CORD, DIVISION, FORMATION, PHONE, PRODUCT, TEXT}\}$
   COL: $\text{Class} = \{\text{YES, NO}\}$
Step 1: Getting feature vectors
Getting both training and test data

**Step 1:** Getting feature vectors

**Notation**

- Features as variables $A_1, \ldots, A_m$
- Feature values $x_1, \ldots, x_m$, $x_i \in A_i$
- Each object represented as feature vector $\mathbf{x} = \langle x_1, \ldots, x_m \rangle$
- Feature vectors are elements in an $m$-dimensional feature space
- Set of instances $X = \{ \mathbf{x} : \mathbf{x} = \langle x_1, \ldots, x_m \rangle, x_i \in A_i \}$. 
Getting both training and test data

Step 1: Getting feature vectors – Example

Outside, a line of customers waited to get in. Please hold the line. She hung the washing on the line. He drew a line on the chart. …
Step 2: Assigning true classification

- Take a number of original objects and assign true classification to each of them.

- Take these objects and their true classification, do preprocessing and feature extraction. It results in $Data = \{ \langle x, y \rangle : x \in X, y \in C \}$. 
Step 3: Selecting training set \( \text{Train} \) and test set \( \text{Test} \)

- \( \text{Train} \subseteq \text{Data} \)
- \( \text{Test} \subseteq \text{Data} \)
- \( \text{Train} \cap \text{Test} = \emptyset \)
- \( \text{Train} \cup \text{Test} = \text{Data} \)
Machine learning process

Where are we now?

1. Formulating the task
2. Getting both training and test data
3. Building classifier using training data
4. Prediction on test data
5. Evaluation
6. Classifier to use
Building classifier

Classifier as a mapping

- real world objects
- feature extraction
- true classification
- feature vectors
- classifier
- target class
Building classifier

Classifier as a mapping

- We look for a **prediction function**, i.e. a classifier $c : X \rightarrow C : c(x) = y$, $x = \langle x_1, x_2, ..., x_m \rangle \in X$, $y \in C$.

- At the beginning we do not know the target prediction function. We need to approximate it using a **hypothesis** $h : X \rightarrow C$.

- Then, we **search** for the best hypothesis $h^*$ that is finally taken as $c$. 
Two types of parameters in machine learning

- Each machine learning method determines a particular form of prediction function.

- The purpose of the learning process is to search for the best parameters of the prediction function.

<table>
<thead>
<tr>
<th>learning parameters</th>
<th>hypothesis parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>= parameters of the learning process</td>
<td>= parameters of the prediction function</td>
</tr>
</tbody>
</table>
Building classifier using training data

Terminological note

- **Model** = method + set of features + learning parameters

- **Classifier** = trained model, i.e. an output of the machine learning process, i.e. a particular method trained on a particular training data.

- **Prediction function** = classifier (used in mathematics). It’s a function calculating a response value using predictor variables.

- **Hypothesis** = prediction function – not necessarily the best one (used in theory of machine learning).
Building classifier using training data

Supervised learning

\[
Data = \{ \langle x, y \rangle : x \in X, y \in C \}\]

Unsupervised learning

\[
Data = \{ x : x \in X \}\]
Building classifier using training data

**Classification:** $C$ is categorical

**Regression:** $C$ is numerical

---

**Classification Diagram:**
- Points are classified into categories A1 and A2.
- Red points represent one category, blue points another.

**Regression Diagram:**
- The target values are plotted against A1.
- Three curves represent linear, quadratic, and cubic regressions.
- The curves fit the data points.

---

ESSLLI ’2013
Hladká & Holub
Day 3, page 16/64
Building classifier using training data

Development cycle

- Selection of ML method
- Learning parameter tuning
- Feature engineering
- Evaluation on development test data

Training data → Building classifier using training data → Classifier $h^*$
Building classifier using development data

Development data is the set of all examples available to developer

Development cycle

- **Input** Development data (e.g., col.development.csv)
  - Splitting the development data into development working set and development test set

- **Iteration**
  - Learning parameters setting and feature set selection
    Then using development working data to train a classifier
  - Prediction on development working and test sets
    Computing **training error** and **generalization error**
  - Evaluation and analysis of the current classifier

- **Output** $h^* = \text{the best classifier, with the lowest generalization error}$
Overfitting

Example 1

Draw decision boundary between classes described by a linear function $h(x)$.
Overfitting

Example 2

Draw decision boundary between classes described by quadratic function $h_2(x)$
Overfitting

Example 3

Draw decision boundary between classes described by complex function $h_3(x)$
Overfitting

Comparing Examples 1–3

• $h(x)$: a straight line – determined by two parameters of the prediction function
  – doesn’t fit two examples

• $h_2(x)$: a parabola – determined by three parameters of the prediction function
  – doesn’t fit one example

• $h_3(x)$: a curve – determined by many parameters of the prediction function
  – perfectly fits all examples
Overfitting

If the generalization error increases while the training error steadily decreases then a situation of overfitting may have occurred.

Generalization error has its global minimum $\Rightarrow$ the best model
How to avoid overfitting

• feature engineering
  • informative features, i.e. useful for classification; control it by training error
  • robust features, i.e. not sensitive to training data; control it by generalization error

• learning parameters tuning
Machine learning process

Where are we now?

- Formulating the task
- Getting both training and test data
- Building classifier using training data
- Prediction on test data
- Evaluation
- Classifier to use
Prediction by $h^*$ on test data

Test data $Test$, unseen during the training (e.g. col.test.csv)

Doing prediction
$\forall x$ such that $\langle x, y \rangle \in Test$: Get $h^*(x)$. 

![Diagram](image)
Evaluation of $h^*$ on test data

Comparing true classification with the predicted classification
\[ \forall x \text{ such that } \langle x, y \rangle \in \text{Test}: \text{Compare } y \text{ and } h^*(x) \]
Machine learning process & Development cycle

1. Formulating the task
2. Getting both training and test data
3. Building classifier using training data
4. Prediction on test data
5. Evaluation

- Selection of ML method
- Learning parameter tuning
- Feature engineering
- Evaluation

Classifier to use
Block 3.2
Naive Bayes learning – Theory

Machine learning process

1. Formulating the task
2. Getting both training and test data
3. Building Naive Bayes classifier using training data
4. Prediction on test data
5. Evaluation

Naive Bayes classifier to use
### Two types of parameters in machine learning – Examples

<table>
<thead>
<tr>
<th>ML algorithm</th>
<th>learning parameters</th>
<th>hypothesis parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT</td>
<td>minsplit (minimum number of instances in the associated training subset in order for a decision to be attempted), ...</td>
<td>decisions</td>
</tr>
<tr>
<td>NB</td>
<td>–</td>
<td>probabilities</td>
</tr>
</tbody>
</table>
Example The task of word sense disambiguation

- Outside, a line of customers waited to get in.
  - Are you sure of the sense FORMATION? Yes, I’m sure.

- He quoted a few lines from Shakespeare.
  - Are you sure of the sense TEXT? Yes, I’m sure.

- This has been a very popular new line.
  - Are you sure of the sense PRODUCT? No, I’m not sure.
  - Are you sure of the sense CORD? No, I’m not sure.
  - Which sense is more likely?

Probability theory provides a framework for the quantification and manipulation of uncertainty.
What is the sense of a word in a sentence?

Use conditional probabilities.

1. \( P(\text{CORD} | \text{This has been a very popular new line.}) \)
2. \( P(\text{DIVISION} | \text{This has been a very popular new line.}) \)
3. \( P(\text{FORMATION} | \text{This has been a very popular new line.}) \)
4. \( P(\text{PHONE} | \text{This has been a very popular new line.}) \)
5. \( P(\text{PRODUCT} | \text{This has been a very popular new line.}) \)
6. \( P(\text{TEXT} | \text{This has been a very popular new line.}) \)

Output the sense with the highest conditional probability.

Use training data to get conditional probabilities.
Let $\mathbf{x}$ be an instance with feature values $x_1, x_2, \ldots, x_m$ and $C$ is a target class with possible values $\{y_1, y_2, \ldots, y_k\}$.

**Goal:** Classify $\mathbf{x}$ into one of $k$ classes $\{y_1, y_2, \ldots, y_k\}$.

**Output:** Target class value $y^*$ with the highest (maximal) conditional probability $P(y_i|\mathbf{x})$, i.e.

$$y^* = \text{argmax}_{y_i \in C} P(y_i|\mathbf{x})$$

The **argmax** operator will give $y_i$ for which $P(y_i|\mathbf{x})$ is maximal.
Probabilistic inference

\[ P(y_i|x) \textbf{ and } P(x|y_i) \]

**Example:** Assume instance \( x = \langle x_{11}, x_{13}, x_{15} \rangle \).

\[
P(\text{PRODUCT}|\text{TRUE, draw, between}) \quad \text{from definition} \quad \frac{P(\text{PRODUCT, TRUE, draw, between})}{P(\text{TRUE, draw, between})}
\]

\[
P(\text{TRUE, draw, between}|\text{PRODUCT}) \quad \text{from definition} \quad \frac{P(\text{PRODUCT, TRUE, draw, between})}{P(\text{PRODUCT})}
\]
How to calculate $P(y_i|x)$?

Use Bayes theorem

$$P(A|B) \overset{definition}{=} \frac{P(B|A) \cdot P(A)}{P(B)}$$

Then

$$y = \arg\max_{y_i \in C} P(y_i|x) \overset{Bayes\ theorem}{=} \arg\max_{y_i \in C} \frac{P(x|y_i)P(y_i)}{P(x)}$$
Naive Bayes learning

\[ y = \arg \max_{y_i \in C} \frac{P(x | y_i) P(y_i)}{P(x)} \]
\[ x = \langle x_1, ..., x_m \rangle \]
\[ y = \arg \max_{y_i \in C} \frac{P(x_1, ..., x_m | y_i) P(y_i)}{P(x_1, ..., x_m)} \]

- Since \( P(x_1, ..., x_m) \) is not dependent on \( C \), it doesn’t influence \( \arg \max_{y_i \in C} \). Therefore

\[ y = \arg \max_{y_i \in C} P(x_1, ..., x_m | y_i) P(y_i) \]
Naive Bayes learning

assumes that features it uses are conditionally independent of one another given a target class.

**Formal definition of conditional independence**

Two events $A$ and $B$ are conditionally independent given an event $D$ if

$$P(A|B, D) = P(A|D)$$

I.e. knowledge of $B$’s value doesn’t affect our belief in the value of $A$, given a value of $D$. 
How to calculate $P(x|y_i)$ given the assumption of conditional independence of features given a target class $C$?

$$P(x|y_i) = P(x_1, x_2, ..., x_m|y_i)$$

chain rule

$$= P(x_1|x_2, ..., x_m, y_i)P(x_2|x_3, ..., x_m, y_i)...P(x_m|y_i)$$

ass. conditional indp.

$$= \prod_{j=1}^{m} P(x_j|y_i)$$

Then

$$y = \arg\max_{y_i \in C} \prod_{j=1}^{m} P(x_j|y_i)P(y_i)$$
Naive Bayes classifier

How to calculate $P(x_j|y_i)$ and $P(y_i)$?

From training set $Train$ that contains $n$ training examples ($|Train| = n$):

- probabilities of classes

$$P(y_i) = \frac{|\{x : \langle x, y_i \rangle \in Train\}|}{n}$$

- conditional probabilities

$$P(x_j|y_i) = \frac{|\{\hat{x} : \langle \hat{x}_1, \hat{x}_2, \ldots, x_j, \ldots, \hat{x}_m \rangle, y_i \rangle \in Train\}|}{|\{x : \langle x, y_i \rangle \in Train\}|}$$
Naive assumption of feature conditional independence given a target class is rarely true in real world applications.

Nevertheless, Naive Bayes classifier surprisingly often shows good performance in classification.
Task
Assign the correct sense to the target word “line” (“lines”, “lined”)

Objects
Sentences containing the target word (“line”, “lines”, “lined”)

Target class
SENSE = \{CORD, DIVISION, FORMATION, PHONE, PRODUCT, TEXT\}

Features
Binary features \(A_1, A_2, \ldots, A_{11}\)
examples <- read.table("../data/wsd.development.csv", header=T)
examples$A1 <- as.factor(examples$A1)
examples$A2 <- as.factor(examples$A2)
examples$A3 <- as.factor(examples$A3)
... 
examples$A11 <- as.factor(examples$A11)

num.examples <- nrow(examples)
num.train <- round(0.9 * num.examples)
num.test <- num.examples - num.train

set.seed(123); s <- sample(num.examples)

indices.train <- s[1:num.train]
train <- examples[indices.train,]
indices.test <- s[(num.train+1):num.examples]
test <- examples[indices.test,]
First of all, if not installed yet, install the package e1071

```r
# to install the package
> install.packages("e1071")

# to check if the package is installed
> library()

# to load the package
> library(e1071)

# to get help info
> help(naiveBayes)
```
The first model M1 uses only one feature, namely A4

```r
# to create a Naive Bayes model
> M1 <- naiveBayes(SENSE ~ A4, data=train)
>
Prediction on training data

> P1 <- predict(M1, train[5], type="class")
> print(table(P1))
```

```
P1
              cord division formation phone product text
           0     0       0   150   3022    0
```
Comparing the predicted values with the true senses

```r
> table(train$SENSE, P1)

                P1
                cord division formation phone product text
  cord           0       0       0     0     0   303    0
  division       0       0       0     0     0   294    0
  formation      0       0       0     0     0   268    0
  phone          0       0       0     142   205   0    0
  product        0       0       0     0     0  1646   0
  text           0       0       0     8     306   0    0

> round(sum(diag(table(train$SENSE, P1)))/num.train * 100, 2)
[1] 56.37
```

56.37 % of training examples are predicted correctly
NB classifier in R – testing the model M1

Predicted values vs. true senses on the test data

```r
> P1.test <- predict(M1, test[5], type="class")
> table(test$SENSE, P1.test)

<table>
<thead>
<tr>
<th></th>
<th>cord</th>
<th>division</th>
<th>formation</th>
<th>phone</th>
<th>product</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>cord</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>division</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>formation</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>phone</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>product</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>192</td>
<td>0</td>
</tr>
<tr>
<td>text</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>37</td>
<td>0</td>
</tr>
</tbody>
</table>
```

57.95% of test examples are predicted correctly

```r
> round(sum(diag(table(test$SENSE, P1.test)))/num.test * 100, 2)
[1] 57.95
```
More models are described in the attached R-script
Homework 3.1

1. Download the col.development.csv data set

2. Load it both into a spreadsheet and into R and look at the data
   - There are 10 numerical features and 1 categorical feature – the description is given on your handout material

3. Split the data into 90%–10% training and test portions

4. Build your own classifier – you can use both (choose at least one)
   - Decision Tree classifier
   - Naive Bayes classifier
     - You can use any subset of the 11 features

5. Prepare a feedback for us – if you want
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 in the future)

2. **compare your classifiers** that you have developed
   – to decide which one is “the best”

= **Model assessment and selection**
You need good performance
not only on *your* data,
but also on any data that can be expected!
Working with data

DEVELOPMENT DATA

- Development working data
  - Training data
  - Held-out data
  ①

- Development test set
  ②
  ③

UNSEEN TEST DATA
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 is 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. 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!).
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!
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 :-)

Usually a simple classifier (e.g. with a default settings of learning parameters) is considered to be a baseline. Then you compare your developed classifier to that baseline.
**Confusion matrix** is a square matrix indexed by all possible target class values.

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

<table>
<thead>
<tr>
<th>Prediction</th>
<th>cord</th>
<th>division</th>
<th>formation</th>
<th>phone</th>
<th>product</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cord</td>
<td>268</td>
<td>3</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>division</td>
<td>3</td>
<td>280</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>formation</td>
<td>13</td>
<td>3</td>
<td>225</td>
<td>4</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>phone</td>
<td>25</td>
<td>5</td>
<td>2</td>
<td>293</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>product</td>
<td>51</td>
<td>10</td>
<td>39</td>
<td>32</td>
<td>1442</td>
<td>72</td>
</tr>
<tr>
<td>text</td>
<td>12</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>28</td>
<td>262</td>
</tr>
</tbody>
</table>

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**.
The case of binary classification

Binary classification \(\equiv\) 2-class classification \(\equiv\) 0/1 classification

In binary classification tasks examples are sometimes regarded as divided into two disjoint subsets:

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

Confusion matrix for binary classification has only 4 cells

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

<table>
<thead>
<tr>
<th>True class</th>
<th>Predicted class</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
<td>True Positive (TP)</td>
<td>False Negative (FN)</td>
</tr>
<tr>
<td>Positive</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td></td>
<td>False Positive (FP)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>

### Explanation

- **Trues** are examples correctly classified
- **Falses** are examples incorrectly classified
- **Positives** were predicted as positives (correctly or incorrectly)
- **Negatives** were predicted as negatives (correctly or incorrectly)
Proportion of correctly predicted test examples
Basic performance measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>$\frac{TP}{(TP+FP)}$</td>
</tr>
<tr>
<td>Recall/Sensitivity</td>
<td>$\frac{TP}{(TP+FN)}$</td>
</tr>
<tr>
<td>Specificity</td>
<td>$\frac{TN}{(TN+FP)}$</td>
</tr>
<tr>
<td>Accuracy</td>
<td>$\frac{(TP+TN)}{(TP+FP+TN+FN)}$</td>
</tr>
</tbody>
</table>

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

$$F = 2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
Summary of Day 3

1. Formulating the task
2. Getting both training and test data
3. Building classifier using training data
4. Prediction on test data
5. Evaluation
6. Selection of ML method
7. Learning parameter tuning
8. Feature engineering
9. Evaluation
10. Classifier to use