A closer look at learning relations from text

Katrenko, S.

Citation for published version (APA):
In this chapter we describe the machine learning methods that we use in our experimental part. We start with the methods that are used as base classifiers and proceed further with the ensembles of classifiers.

3.1 General notions

Let us formulate a learning task (supervised classification) as follows. Let $X$ be an input space and $Y$ be an output space. Pairs $(X, Y) \in X \times Y$ are random variables distributed according to the unknown distribution $D$. The observed data points we denote by $(x_i, y_i)$ and say that they are independently and identically distributed according to $D$. The goal is to construct a hypothesis $h$ such that for any instance from the input space $X$ it predicts its label from the output space $Y$, i.e. $h : X \rightarrow Y$.

If $Y = \{+1, -1\}$, then it is a binary classification task. Let also every example $x_i \in X, i = 1, \ldots, n$ be represented by a fixed number of features, $x_i = (x_{i1}, \ldots, x_{ik})$.

A hypothesis $h$ is an approximation of some unknown target function $t$. Ideally, the hypothesis $h$ should not make any errors on the data points which would mean that the probability $p(h(X) \neq Y)$ is as low as possible. The risk of $h$ is defined in Eq. (3.1) where the fact that a given hypothesis $h$ can err on the data is reflected in a loss function, $l(h(x_i), y_i)$.

$$R(h) = p(h(X) \neq Y) = \mathbb{E}(l(h(X), Y))$$  (3.1)

The risk of the target function $t$ is the minimum over all possible hypotheses $g$ and is called the Bayes risk $R^* = \inf_g R(g)$. Since the underlying distribution is unknown, the quality of $h$ is usually measured by the empirical error in Eq. (3.2)

$$R_n(h) = \frac{1}{n} \sum_{i=1}^{n} l(h(x_i), y_i)$$  (3.2)

Several loss functions have been proposed in the literature so far, the best known of which is the zero-one loss (Eq. 3.3). This loss is a function that outputs 1 any time a method errs on a data point $(h(x_i) \neq y_i)$ and 0 otherwise.

$$l(h(x_i), y_i) = \begin{cases} 1 & \text{if } h(x_i) \neq y_i \\ 0 & \text{otherwise} \end{cases}$$  (3.3)

Expected risk

Empirical risk

Zero-one loss
At first glance the goal of any learning algorithm should be to minimize empirical error $R_n(h)$, which is often referred to as empirical risk minimization. This turns out to be not sufficient as some methods can perform well on the training set but be not as accurate on the new data points. This led to the introduction of structural risk minimization (Eq. 3.4) where not only the empirical error is taken into account but the complexity (capacity) of $h$ as well. In Eq. 3.4 $\text{pen}(h)$ stands for a penalty that reflects complexity of a hypothesis. In other words, if a hypothesis is very complex, it will be penalized more.

Consider for instance a case of fitting $n$ data points by a polynomial function. The higher degree of a polynomial we choose, the better fit on the data points will be achieved. However, if new data points are given, our model based on the high degree polynomial will not be appropriate because it most likely overfits on the new data.

An error can be decomposed into bias and variance. Note that in some cases a true function $t$ may belong to the chosen hypothesis space $\mathcal{H}$ (realizable case) and in others it may not (agnostic case). Of course, the agnostic case is more realistic because a hypothesis space is usually of limited size and there is no reason to assume that the target function $t$ belongs to $\mathcal{H}$. If $h^*$ is the best function in $\mathcal{H}$ with $R(h^*) = \inf_{h \in \mathcal{H}} R(h)$, then the difference $|R(h^*) - R^*|$ is called the approximation error or bias. A quantity that measures how far any hypothesis $h$ in $\mathcal{H}$ is from its best hypothesis $R(h^*)$ is referred to as an estimation error or variance. Given the notation we use here a variance is defined by $|R(h^*) - R_n(h)|$. Consequently, bias does not depend on data used during the training phase whereas variance always does. Variance is equal to zero if predictions of a method do not change and are always the same regardless of the training data. Bias is equal to zero if a classifier outputs the optimal prediction.

3.1.2 Individual classifiers

In this section we discuss the methods that will be used throughout the thesis. We start with one of the most popular approaches, Naive Bayes and proceed with memory-based learning (MBL) and kernel methods.

One of the simplest widely used machine learning methods is the Naive Bayes classifier (Eq. 3.5). Here, a dependency between categories
and data instances is modeled via conditional probability where a given example \( x_i \) is assigned the most probable class.

\[
h(x_i) = \arg \max_j p(y_j|x_i) = \arg \max_j \frac{p(x_i|y_j)p(y_j)}{p(x_i)}
\]  
(3.5)

We can ignore the denominator in Eq. 3.5 because it does not depend on \( y \) and rewrite this equation as in Eq. 3.6. By assuming that all features are conditionally independent, we arrive at Eq. 3.8.

\[
p(y|x_i) = p(x_i|y)p(y)
\]  
(3.6)

\[
= p(x_{i1}, \ldots, x_{ik}|y)p(y)
\]  
(3.7)

\[
= p(y) \prod_j p(x_{ij}|y)
\]  
(3.8)

Despite the independence assumption which is often violated in practice, naive Bayesian classification has been successfully used for the number of tasks. These include document classification \([116]\), spam filtering \([121]\) and others. This method is not computationally expensive. Given \( n \) training examples, each represented by \( k \) features, training time of Naive Bayes equals to \( \Theta(nk + |Y||F|) \) where \( |Y| \) is a number of classes (as defined in previous section) and \( |F| \) is a number of unique feature values.

One of the reasons why this approach provides good performance is that as long as the target class is more probable than the others, exact values of probabilities do not matter. Zhang \([209]\) studied in depth why feature dependencies do not seem to affect performance and concluded that the behaviour of the Naive Bayes classifier is influenced by dependence distribution rather than by feature dependencies. More precisely, if dependencies are distributed evenly between classes or they cancel each other out, Naive Bayes is an optimal classifier. Rish \([153]\) took another approach to explain the efficiency of this method. Instead of using real-world data sets, she employed Monte Carlo simulations to study the behaviour of the method on randomly generated problems. The findings of Rish \([153]\) confirmed that Naive Bayes yields a good performance when the features are independent. In addition, they also revealed that functionally dependent features do not degrade performance which might explain successful applications.

Naive Bayes can be considered as a special case of Bayesian networks where all features are independent given the class value. Bayesian networks belong to the family of probabilistic graphical models and can be formalized as follows. A Bayesian network is represented by a directed acyclic graph whose nodes are variables (or features). For a node \( A \) in the network, any variables that are non-descendent of \( A \) are conditionally independent of \( A \) given the parents of \( A \). Having defined
the Bayesian network in this way, it becomes possible to make complex
inferences using probabilistic dependency relations between random
variables. Bayesian networks have been less popular in the NLP com-
munity but often, if used, they proved to provide good performance.
For instance, Weissenbacher [201] has shown that Bayesian networks
outperform other methods on the subtask of anaphora resolution.

There exist different types of rules that are used in data mining and
classification. Consider a common value-attribute representation where
each example in the training set is represented by a fixed number of fea-
tures (attributes). To detect dependencies between examples’ attributes,
association rules have been proposed [3]. Association rules have been
particularly useful for market analysis where, for instance, one studies
which products humans usually buy together. For classification, the
most popular are variants of decision rules. A goal of classification rule
induction is to find a hypothesis (which is a set of rules) such that it
covers all positive examples in the data and is consistent (does not cover
any negative examples). More generally, classification rule induction is
divided into propositional rule learning and relational rule learning [46].
The former is concerned with induction of ‘if-then’ rules where each
element obtains a class label given that some conditions defined on
attributes hold. Let us consider binary classification where each test
example has to be classified as such that it is an instance of the Content
- Container relation or not. If each example is represented by a number
of attributes one of which is Context, a possible classification rule could
be “if the attribute Context is equal ‘contains’, classify an example as
positive”. Consequently, if there is a test example <apple, box> that
occurs in the sentence “A box contains an apple”, it will be classified as a
positive instance for Content - Container relation. While propositional
rule learning works on attributes, relational rule induction is employed
when data is stored in several tables (e. g. in relational database form).
A widely used form of relational learning is inductive logic program-
ning (ILP) which explores a first-order rule formalism for concept and
hypothesis representation. One of the advantages of ILP can be seen in
the ability to incorporate prior knowledge in the learning process. To
accomplish this, background knowledge is represented by additional
clauses. In NLP, ILP was used to derive grammar rules [36] and for some
other tasks such as part-of-speech tagging.

MBL [37] differs from the approaches discussed above. Firstly, MBL
stores all examples that are found in the training set without inferring
any hypothesis from the data. Given a new, unseen instance, it compares
it against all examples from the training set by using some similarity
measure. Once the most similar instances (the closest neighbours) of
the new instance are detected, it is assigned the class label that the
neighbours have. The term ‘k-neighbour classification’ stems from the
fact that the classification depends on the class values of k neighbours
where \( k \) is selected in advance. MBL is often referred to as lazy learning whereas the approaches based on abstracting from data are called eager learning methods. It has been shown by Hendrickx [68] that it is possible to combine eager and lazy learning into hybrid algorithms. For instance, by combining the strengths of rule induction and storing rules as an instance base. Test instances are then compared against the rules rather than training instances. The experimental findings in this case suggest that a hybrid algorithm outperforms \( k \)-nearest neighbour and a rule induction method on various NLP data sets.

MBL has been successfully applied to a number of NLP problems. These include morpho-phonology, shallow parsing, dependency parsing [22] and others. Daelemans and van den Bosch [37] explain popularity and effectiveness of MBL in NLP by the fact that there are many sub-regularities and exceptions in language which makes generalization very difficult. By storing all examples in memory one always has access to exceptions and can recognize them in test data.

The past decades have witnessed a boost of interest in kernel methods, their theoretical analysis and practical applications in various fields. The idea of having a method that would work with different structures and representations, starting from the simplest representation using a limited number of attributes to such complex structures as trees, seems indeed very attractive.

The key idea of kernel methods lies in the implicit mapping of objects to a high-dimensional space and considering their inner product (similarity) rather than representing them explicitly. Unfortunately, not all functions can be used in kernel methods as it has to be a positive definite one (Def. 6).

**Definition 6 (Valid kernel).** A function \( k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \) is called a positive definite kernel iff it is symmetric (\( k(x, x') = k(x', x) \)) for any two objects \( x, x' \in \mathcal{X} \), and positive definite

\[
\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0
\]

for any \( n > 0 \), any objects \( x_1, \ldots, x_n \in \mathcal{X} \), and any choice of real numbers \( c_1, \ldots, c_n \in \mathbb{R} \)

**Example 1 (Identity matrix).** Any identity matrix \( M \) such that its diagonal elements are equal to 1 and all others to 0, is positive definite. This can be easily checked by using Def. 6. For any \( c_1, \ldots, c_n \in \mathbb{R} \),

\[
\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_i) = \sum_{i=1}^{n} c_i^2 \geq 0
\]
In the literature one may find cases where the kernels are not valid but still performed well on the learning tasks. Their behaviour is however less clear and cannot be analyzed by standard means.

Using the idea of a kernel mapping, support vector machines (SVM) were introduced as a method which seeks the linear separation between two classes of the input points, \( f(x) = w^T x + b \in \mathbb{R}^p, b \in \mathbb{R} \). Here, \( w^T \) stands for the slope of the linear function and \( b \) for its offset. Often, there can exist several functions that separate data well but not all of them are equally good. Intuitively, a hyperplane that separates mapped examples with the largest possible margin would be the best option.

If one pictures binary classification as in Fig. 2 then the goal is to maximize the margin between two halfspaces defined by \( f(x) \geq 1 \) and \( f(x) \leq -1 \), \( h(x) = \text{sign}(f(x)) \). Note that the distance between two halfspaces is determined by the difference in offsets and the slope and is equal \( \rho = 2/\|w\| \) where \( \|w\| \) is the Euclidean norm of \( w \). Alternatively, this can be seen as computing a distance from a point on one hyperplane to the other hyperplane which gives the same result. In Fig. 2 the data points that lie on the lines defining hyperplanes (i.e., on \( w^T x + b = 1 \) and \( w^T x + b = -1 \)) are marked by diamonds. These are called support vectors.

**Figure 2: Illustrating SVM: a separable case**

Maximizing the margin can be reformulated as the following optimization problem:

\[
\text{argmin}_{w, b, \xi_1, \ldots, \xi_l} \left\{ \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{l} \xi_i \right\}
\]

In Eq. [3.11] the first part of the equation corresponds to the margin maximization while the second takes into account the error on the train-
ing set which has to be minimized (where \( C \) is a penalty term). To handle cases where data cannot be perfectly separated by a linear hyperplane, it was proposed to use the hinge loss (Eq. 3.12) instead of the zero-one loss.

\[
l_h(f(x_i), y_i) = \max(0, 1 - y_i f(x_i)) \tag{3.12}
\]

Contrary to the zero-one loss, the hinge loss can handle cases where \( 0 \leq y_i f(x_i) < 1 \) by considering that a data point \( x_i \) was either classified correctly with low confidence or misclassified. For practical reasons, and namely because of non-differentiability of the hinge loss function, the minimization problem is rewritten by using so-called slack variables \( \xi_i \geq l_h(f(x_i), y_i) \) and by solving thus obtained quadratic programming problem via Lagrange multipliers (more details can be found in Vert et al. [197]). In a nutshell, the hyperplane that is found corresponds to a non-linear boundary in the original input space. There exists a list of standard kernel techniques such as linear kernel, Gaussian kernel and others. Still, often information about the data or problem can lead to introducing a new kernel. It has been shown by [66] that a complex kernel (referred to as a convolution kernel) can be defined using simpler kernels (Def. 7).

**Definition 7 (Convolution kernel).** The convolution kernel of two kernels \( k_1 \) and \( k_2 \) such that they are functions \( X \times X \rightarrow \mathbb{R} \) is the following function \( k_1 \ast k_2 \):

\[
k_1 \ast k_2(x, x') = \sum_{x_1, x_2 = x, x'_1 = x'} k_1(x_1, x'_1)k_2(x_2, x'_2) \tag{3.13}
\]

Since their introduction, convolution kernels have been widely used in a number of fields, including natural language processing [208, 16]. Kernels that have been proposed for relation learning are reviewed in Chapter 6.

### 3.1.3 Ensemble methods

The fact that different machine learning methods may err on different data instances and be complementary to each other led to the development of various ensemble methods. Ensemble of classifiers are known to provide better performance if the individual (base) classifiers are diverse and accurate. A classifier is said to be accurate when its error rate is better than random guess [41]. Accuracy of the individual classifiers would not suffice if they erred on exactly same data points. Hence, diversity has become an additional requirement to the base classifiers.

Dietterich [41] explained the efficiency of ensemble methods by considering three issues, statistical, representational and computational. Firstly, if the data sets are small but the hypothesis space is large, it may happen
that individual classifiers yield the same accuracy but explore different parts of the hypothesis space. If combined, these classifiers would most likely contribute to a higher accuracy. Secondly, since many of the classifiers rely on the (usually computationally expensive) search in the hypothesis space, combining classifiers that perform local search becomes an attractive option. Thirdly, ensembles of classifiers allow to expand the hypothesis space which is particularly important when the true function does not belong to the initial hypothesis space.

Bagging and AdaBoost present ensemble methods manipulating the training examples. The main idea behind such methods lies in generating multiple hypotheses. In the case of bagging, a different subset from the training data is sampled every time a learning algorithm is applied. Stacking belongs to the method of combining different classification models.

**Bagging**

Aggregated bootstrap (or bagging) was introduced by Breiman [13]. Given \( n \) training examples one can create \( m \) new data sets \( (m \leq n) \) by sampling examples uniformly with replacement. When sampling with replacement is used, it is likely that some examples from the initial training set will be repeated. A newly created data set is referred to as a ‘bootstrap sample’. Further, one classifier is built for each of \( m \) bootstrap samples. Finally, predictions of these classifiers are combined via voting (an aggregation is taking place).

It has been noted that bagging works particularly well for the unstable learning methods such as decision trees or neural nets. Experiments with the k-nearest neighbours method (which is known to be stable) supported this claim by showing that the overall performance when using bagging does not usually change. For decision trees, increase in accuracy comes at the expense of interpretability [13].

**Boosting**

The AdaBoost algorithm was proposed by Freund and Schapire [48] who similarly to bagging focused on manipulating the training instances. But in contrast to bagging, AdaBoost assigns weights to the training examples and in each iteration \( i \) it attempts to minimize a weighted error on the training set. The weighted error is further used to update the weights on instances. The main idea behind boosting is that misclassified examples will be weighted more and in the next iteration an algorithm would focus on classifying them correctly. If we denote a hypothesis that was produced after the \( i \)th iteration by \( h_i \), the final classifier \( h_f \) is constructed by a weighted vote over the individual classifiers 

\[
  h_f(x) = \sum_i w_i h_i(x).
\]

Several authors have shown that AdaBoost performs poorly in noisy settings but it does not overgeneralize in other cases. The initial ex-
planning of the good performance of AdaBoost in terms of a margin was proposed by Schapire et al. [167]. A margin is said to measure the confidence of the final classifier so that eventually, its goal becomes to maximize the margin. However, it has been shown recently [151] that maximizing margins is not sufficient. In fact, the generalization error of the final classifier also depends on the complexity of the base classifiers and on the size of the training set. Reyzin and Schapire [151] have conducted experiments with decision trees by varying their complexity (their depth and the number of leaves). Their empirical findings suggest that even if margins are large, high complexity of the base classifiers (decision trees are deeper) causes increase in the error rate.

Stacking

Stacked generalization (stacking) differs from the two approaches described above. Instead of manipulating the training instances, it focuses on combining different classification methods [206]. More precisely, let $r$ be a number of classifiers that are applied to the same data set. They output $r$ classification models that are called ‘level-$0$ models’. Predictions of these $r$ models on each data point are further used together with its true label as an input for the final classifier (‘level-$1$ model’). Its predictions are the final classification results. Ting and Witten [187] investigated the role of the well-known machine learning methods such as a decision tree algorithm, k-nearest neighbour method and others on stacking at ‘level-$0$’ and at ‘level-$1$’ on 10 different data sets. They concluded that stacking works the best for larger data sets and that the regression method is well suited for the ‘level-$1$’ classifier. An interesting observation was made by comparing stacking against majority vote. It turned out that stacking performed better on eight out of ten data sets (on six of them significantly better).

Most comparative studies on meta-learning were done on bagging and boosting [140]. The majority of them showed the same tendency. Namely, bagging and AdaBoost work well for such algorithms as rule learning or decision tree methods which are generally considered to be unstable. If one compares various meta-learning schemes from the bias/variance point of view, then the bagging goal is to reduce variance while stacking affects bias. In a nutshell, stacking detects biases of the base classifiers and filters them out.

3.2 Evaluation Metrics

In machine learning performance of a given algorithm on a data set is measured by several metrics. These metrics reflect how well the method works. Consider for instance binary classification where each examples has to be classified either as positive or as negative. Positive examples
on which the method errs are referred to as false negatives (FN) and negative examples which it misclassifies are called false positives (FP). Those examples that are classified correctly are either true positives (TP) or true negatives (TN).

Accuracy is defined as the fraction of all examples that were classified correctly (Eq. 3.14). Accuracy is often used when the data set is balanced (i.e., a number of true positives and true negatives is the same).

\[
\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}
\]  

(3.14)

Precision reflects how many examples in the data set that were classified as positive really belong to true positives, Eq. 3.15.

\[
\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}
\]  

(3.15)

Recall shows what fraction of the true positives were found by the method (Eq. 3.16).

\[
\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}
\]  

(3.16)

The \( F_1 \) score is defined as the harmonic mean between precision and recall (Eq. 3.17).

\[
F_1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\]  

(3.17)

3.3 Conclusions

This chapter presented the basic notions that are used in machine learning along with some popular methods that we employ later in the thesis. In particular, rule induction is used in Chapter 4 to determine whether textual entailment takes place. Naive Bayes, bayesian networks, MBL, and meta-learning are studied in Chapter 5 where they are applied to a novel representation derived from syntactic trees. Kernel methods are explored in Chapter 6 where they are combined with prior knowledge to extract domain-dependent and generic relations from text.