**2**

This chapter

Sentiment analysis (also Opinion Mining)

* Text Mining in general
  + Information Retrieval as first step OR database
  + Linguistic analysis, NLP (POS-tagging)
* TM process
  + Structure data
  + find patterns
  + evaluate results
* TM topics
  + text categorization
  + text summarization
  + entity relation modeling
  + sentiment analysis
  + …
* TM concepts?
  + TFIDF

Plutchik

8 created emotions 🡪 Buch6

resources at WordNet

Machine learning

supervised/unsupervised/…

regression/classification

Multiclass Classification

Multiclass classification is a problem in the field of supervised learning. The task is the prediction of one class out of a finite set of three or more possible classes for each instance. As can be seen in figure~\ref{fig:problem\_hierarchy\_supervised\_learning}, it can be distinguished from other problems in supervised learning. First, as the name implies, it is a classification problem because the range of values is finite rather than continuous, as is the case with a regression problem. Furthermore, each instance should be associated with exactly one class, which distinguishes it from the similar sounding multi-label classification, since each instance can be associated with any number of classes (= labels). Finally, the model to be learned has to choose from a set of $k$ classes with $k>2$, i.e. ${1,2,\dots,k}$. This makes the problem slightly more difficult than binary classification with $k=2$. Ordinal classification is a specialization of multiclass classification and will be discussed in detail in subsection~\ref{subsec:ordinal\_classification}.

Most algorithms applied to classification problems are explained by their application to binary classification problems, which cover a large part of the real-world use cases. However, these approaches cannot be adopted to multiclass problems without further ado. \citeauthor{Aly2005} describes with the decomposition into binary classification, hierarchical classification and the extension of algorithms the three groups of approaches to cope with the difficulties of multiclass classification \autocite{Aly2005}:

\begin{itemize}

\item Decomposition into binary classification

A frequently chosen approach is the transformation of the multiclass classification problem into several binary classification subproblems, for which there are three different possibilities. In any case the results of subproblems have to be joined to be able to make a final prediction which is similar to ensemble learning. \\\\

The first option called one-against-all splits the multiclass classification problem into $k$ binary classification subproblems. The classifier $f\_{i}$ treats the instances of class $i$ as positive and instances of the $k-1$ other classes as negative. An example of one-against-all using an SVM-classifier is shown in figure~\ref{fig:one\_against\_all\_or\_one} on the left. For an unseen instance $x$ the class $k$ with the highest confidence score will be predicted:

\[y(x)=\underset{i\in\{1 \dots k\}}{\operatorname{argmax}} f\_{i}(x)\]

However, there is no guarantee that the real-valued quantities $f\_{i}(x)$ for different classifiers will have appropriate scales. Furthermore, especially for a high $k$ the ratio between positive and negative instances is low, which complicates the creation of a model. This problem can be addressed by giving greater weight to the positive instances \autocite[Subsection~7.1.3]{Bishop2006}. \\\\

The second option called one-against-one splits the multiclass classification problem into even more binary classification subproblems, in fact $\frac{k(k-1)}{2}$. This number results from the fact that now in each subproblem two classes are pairwise distinguished from each other. Since in the binary classifier between the classes $i$ and $j$ all instances of the classes except $i$ and $j$ can be discarded (see figure~\ref{fig:one\_against\_all\_or\_one} on the right), there are smaller subproblems than in one-against-all, which usually also create simpler models. However, the number of subproblems is $\frac{k}{2}$-times as high, which usually results in significantly more training time \autocite[Subsection~7.1.3]{Bishop2006}. \\

The final decision is made by voting. All pairwise comparisons are summed up and the winner with the most partial victories is the output class. However, using the binary outputs of the subproblems, i.e. only the preferred class, often leads to several winners. There are different ways to handle these ambiguities. The first idea is weighting the subproblems differently, for example by modifying the output from $1$ to the accuracy and from $0$ to $1-$accuracy. Another idea called pairwise coupling can be used which includes probabilities for every subproblem in the calculation. \citeauthor{Wu2004} present a few pairwise coupling techniques that are more stable than using the simple voting approach \autocite{Wu2004}. \\\\

The third option is the usage of error-correcting output codes. Each class is assigned a unique binary string of length $n$; we will refer to these strings as “codewords”. During training for an example from class $i$, the desired outputs of these $n$ binary functions are specified by the codeword for class $i$. With artificial neural networks, these $n$ functions can be implemented by the $n$ output units of a single network \autocite[Section~1]{Dietterich1994}. Though, it is also possible to train all $n$ binary classifiers individually with an arbitrary algorithm. A test instance is assigned to the class that is closest to the code determined by the $n$ functions. \\

Now the question arises, how the codewords respectively the binary functions are chosen most effectively. Out of all $2^{k}$ binary combinations for a $k$-class problem, after removing $2^{k-1}$ complements (e.g. all starting with $1$) and the uniform classifier there are $2^{k-1}-1$ possible classifiers, from which to be selected. A good error-correcting output code for a $k$-class problem should satisfy two properties \autocite[Subsection~2.3]{Dietterich1994}:

\begin{itemize}

\item Row separation

If in a code converting an arbitrary codeword into another arbitrary codeword needs $d$ changes of bits, the code has a so-called Hamming distance of $d$. The higher the Hamming distance, the better the row separation. A code with Hamming distance $d$ can detect up to $d-1$ and correct up to $\left\lfloor\frac{d-1}{2}\right\rfloor$ bits.

\item Column separation

Each bit-position function $f\_{i}$ should be uncorrelated with the functions to be learned for the other bit positions $f\_{j}$; $j\neq i$. If two columns $i$ and $j$ (or the complement of $j$) are similar or identical, then when a deterministic learning algorithm such as C4.5 is applied to learn $f\_{i}$ and $f\_{j}$, it will make similar (correlated) mistakes.

\end{itemize}

The classifier sets in the one-against-all and one-against-one approaches are feasible solutions. The one-against-all approach guarantees Hamming distances of $2$ for both row and column separation, the one-against-one approach even $2\times(k-2)$ for the row and $4$ (if $k>7$) for the column separation. The optimal row distance is achieved by the union of all $2^{k-1}-1$ binary combinations and called exhaustive code. Until $k=5$ the exhaustive code is just the union of all classifiers of the one-against-all and one-against-one approaches (or their inversions, see table~\ref{tab:exhaustive\_ecoc}), but for $k>6$ the number of additional classifiers increases quickly, wherefore instead of an exhaustive code also a suitable subset of those classifiers can be sufficient and improve the column separation.

table{exhaustive\_ecoc}

The exhaustive error-correcting code for $k=5$ is shown in table~\ref{tab:exhaustive\_ecoc}. The hamming-distance of the code is $d=8$. Therefore, the code can detect up to $d-1=7$ and correct up to $\left\lfloor\frac{d-1}{2}\right\rfloor=3$ bits. Error-correcting output codes are a robust alternative to one-against-all and one-against-one approaches which outperforms them in most cases but also entail the disadvantages of a more complex training and model \autocite[Section~4]{Dietterich1994}.

\item Hierarchical classification

Hierarchical classification, also known as nested dichotomies, is strictly speaking also a type of decomposition of a multiclass classification problem into binary classification problems. However, in this approach the output of the classifiers is not always used for the prediction but as the input for the next classifier. In this way, a hierarchy of classifiers and their subproblems is created. The first classifier is trained on the whole input space; the total set of all classes is suitably divided into two subsets between which the classifier distinguishes. This procedure is then continued recursively with the two subsets until all subsets only contain instances of a single class. This procedure creates a tree structure whose leaves are the set of instances of a class. As a result, this tree contains $k$ leaves and $k-1$ inner nodes representing the binary classifiers \autocite{Dong2005}. Two possibly outcomes of a multiclass classification problem with $5$ classes are shown in figure~\ref{fig:nested\_dichotomies}.

The probability of a class can be calculated by multiplying all probability estimates of the classifier along the path from the root to the class. This value differs for disparate nested dichotomies, wherefore usually various trees are computed and their probabilities are averaged. Though, for high $k$ there are too many different possibilities to build a nested dichotomy. Balanced nested dichotomies prove to be particularly suitable, since they require the shortest running time without affecting the accuracy negatively \autocite{Dong2005}. \citeauthor{Dong2005} developed methods to balance the nested dichotomies either by class or by data amount.

\item Extension of algorithms

The third idea is the inversion of the first two approaches. Instead of modifying the problem and using the same classifier as for the binary classification, we now modify the classifier such that we can leave the classifier unchanged. Typical algorithms in machine learning are decision trees, pattern- or rule-based classifiers, probabilistic classifiers, SVM classifiers, neural network classifiers and proximity-based classifiers. A short description for every algorithm can be found in subsection~\ref{subsec:analysis}. Decision trees, pattern- or rule-based classifiers, probabilistic classifiers as Naïve Bayes and proximity-based classifiers as $k$-nearest-neighbor can be easily adapted to multiclass problems. For the other two algorithms some preliminary considerations are necessary. \\\\

SVM classifiers use hyperplanes of dimension $D-1$ described by linear functions to separate the whole input space of dimension $D$ into two subspaces which ideally only contain a single class. Obviously, a single hyperplane is not sufficient for separating more than two classes. Approaches like one-against-all and one-against-one lead to ambiguous subspaces where either more than one class or no class is assigned to. We can avoid these difficulties by considering a single $k$-class discriminant comprising $k$ linear functions of the form

\[y\_{i}(\mathbf{x})=\mathbf{w}\_{i}^{\mathrm{T}} \mathbf{x}+w\_{i 0}\]

and then assigning a point $\mathbf{x}$ to class $i$ if $y\_{i}(\mathbf{x})>y\_{j}(\mathbf{x})$ for all $j\neq i$. The decision boundary between class $i$ and class $j$ is therefore given by $y\_{i}(\mathbf{x})=y\_{j}(\mathbf{x})$ and hence corresponds to a $(D-1)$-dimensional hyperplane defined by

\[\left(\mathbf{w}\_{i}-\mathbf{w}\_{j}\right)^{\mathrm{T}} \mathbf{x}+\left(w\_{i 0}-w\_{j 0}\right)=0.\]

The decision regions of such a discriminant are always singly connected and convex \autocite[Subsection~4.1.2]{Bishop2006}. \\\\

A neural network classifier has in general $n$ neurons in the output layer, each of them with a binary output. For binary classification problems, a single neuron in the output layer, i.e. $n=1$, is sufficient since it only needs to distinguish between two classes, represented by the two outputs $0$ and $1$. For multiclass classification problems, $n=1$ is not sufficient. The straightforward solution would be to define $n=k$ neurons in the output layer, one for each of the $k$ classes. But already $\left\lfloor\log\_{2}k\right\rfloor$ output neurons are enough if the classes are encoded as the first $k$ numbers in binary representation. By using error-correcting output codes containing more than $\left\lfloor\log\_{2}k\right\rfloor$ output neurons, the binary code can also be made more resistant to errors \autocite[Subsection~2.1]{Aly2005}.

\end{itemize}

As mentioned in the beginning of this subsection, ordinal classification problems are a subgroup of multiclass classification problems. The following subsection describes the special properties of this type of problems and other approaches that are even better adapted to these properties than those presented here.

Ordinal Classification

Ordinal classification problems are a specialization of multiclass classification problems. They refer to an important category of real-world problems, in which the classes the class values exhibit a natural order. Thus, the class attribute is not nominal, but ordinal. However, the possibility of calculating the difference (essential for an interval class attribute) or even the quotient (essential for a ratio class attribute) is usually not given \autocite[Section~1]{Frank2001}. Ordinal classification may be viewed as a bridging problem between the two standard machine-learning tasks of classification and regression, because the set of classes has the property of a finite range given in classification problems as well as the property of a natural order given in regression problems \autocite[Section~1]{Kotsiantis2004}. \\\\

As a specialization of a multiclass classification problem, ordinal classification problems can be approached with the same strategies as described above in subsection~\ref{subsec:multiclass\_classification}. However, the additional information of the linear order of the classes is ignored. Two strategies tailored to ordinal classification problems taking this information into account are ordinal regression and an adapted transformation to binary classification problems.

\begin{itemize}

\item Ordinal regression

In regression tasks, the output is a numeric value in a continuous range. The idea of ordinal regression is to treat the ordinal output as a numeric value. In this way the same approaches as for regression tasks can be used only with the extension that the predicted values still have to be mapped to the fixed classes. Nevertheless, most regression methods require some metric assumptions like building differences or ratios within the ordinal scale \autocite[Section~1]{Ruan2014}. Hence the result of the model depends on the choice of numeric values and their distances from each other. This leads to problems in many real-world applications, for example in the quantification of ordinal scales in opinion surveys. \\\\

Ordinal logistic models avoid this issue by only considering the ranking order of the classes \autocite[Chapter~13]{Harrell2015}. The most commonly used ordinal logistic model is the proportional odds model. It uses the logarithms of odds that the predicted class $y$ is one of the first $i$ classes together with a linear function, whereby the probability after forming is given by

\[\operatorname{P}(y\leq i|\mathbf{x})=\frac{1}{1+\exp\left[-\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}+w\_{i 0}\right)\right]}\]

where $\mathbf{x}$ is the test instance and $\mathbf{w}$ and $w\_{i 0}$ the parameters of the linear function. There are many possible extensions to this ordinal model, among others replacing the cumulative probabilities with conditional probabilities or using a hazards model \autocite[Chapter~13]{Harrell2015}.

\item Transformation to binary classification problems

A simpler approach using a transformation to binary classification problems is presented by \citeauthor{Frank2001}. In contrast to the binary classification approaches in multiclass classification it takes advantage of the order of the classes. The ordinal class attribute with ordered values $v\_{1}<v\_{2}<\dots< v\_{k}$ is converted into $k-1$ binary attributes. The $i$-th binary attribute represents the test if the class attribute exceeds the value of the $i$-th class: $x>v\_{i}$. This construction of binary attributes requires exactly two classifiers and their probabilities to determine whether a test instance $x$ belongs to a class $i$:

\[\operatorname{P}\left(v\_{i}\right)=\operatorname{P}\left(x>v\_{i-1}\right)-\operatorname{P}\left(x>v\_{i}\right)\]

For the highest class, the first probability is $1$ and for the lowest, the second is $0$, which further simplifies the calculation. Finally, the class with the highest probability is predicted. \citeauthor{Frank2001} showed that for ordinal classification problems this approach outperforms the “naïve” one-against-all algorithm, which treats the class values as an unordered set \autocite{Frank2001}.

\end{itemize}

Cost-sensitive learning

In many classification problems the goal is the optimization of some performance measurement, most often the accuracy. However, the best solution in respect to the accuracy does not have to be the one with minimized costs, which is also often a goal to be achieved. Costs can not only arise from misclassifications, but also from additional tests or time for the creation of a better model. \citeauthor{Turney2002} divides the reasons for costs that a machine learning problem can entail into nine groups \autocite{Turney2002}:

\begin{itemize}[noitemsep]

\item misclassification errors

\item tests

\item teacher

\item intervention

\item unwanted achievements

\item computation

\item cases

\item human-computer interaction

\item instability

\end{itemize}

In each of the nine groups, we can distinguish between constant and conditional costs. In the case of test costs, we speak of constant costs if a certain test has a fixed cost value for its execution. Note that different test may have different costs if they still remain constant for all instances in all circumstances. On the contrary, conditional test costs depend on a criterion. For a medical test this criterion might be the test result, the age of a patient or the result of prior tests. In the remaining subsection, of all cost types only the misclassification costs are discussed and therefore abbreviated to costs.

The standard approach to evaluate the performance of a classifier is the calculation of the accuracy, i.e. the number of correct classified instances in relation to the total number of instances. The error costs result directly from the number of misclassified instances, which only has to be multiplied by a constant cost factor. Thus, \citeauthor{Turney2002} calls this value constant misclassification error cost. However, there are various problems in machine learning where such a simplification would lead to a miscalculation of the total costs because the conditional error costs are ignored.

In many cases, the equal treatment of all error types is the cause of the incorrect cost estimate. For this we first consider the binary classification with only two classes and therefore only two possible error types. A false positive (FP) occurs when the outcome is incorrectly predicted as yes (or positive) when it is actually no (or negative). The reverse case is called false negative (FN). False positives and false negatives have rarely equal costs in real-world applications. In spam-classification, a no-spam e-mail classified as spam is worse than a spam e-mail which have not been detected correctly. An accepted customer which is not capable of paying back a loan has bigger costs than a customer wrongly classified as insolvent. A sick patient who is classified as healthy is more problematic than a healthy patient who is classified as sick. In order to map this, a cost value is determined for each of the two errors FP and FN or the cost ratio is determined.

If we transfer the cost calculation to the multiclass classification, there are more error types and therefore more cost factors to be defined: Each instance of class $i$, misclassified as class $j$ will be multiplied with the correspondent cost factor $c\_{ij}$. For a multiclass classification problem with $n$ classes we get a $n\times n$ cost matrix similar to the confusion matrix, which has $n(n-1)$ possibly different cost factors and whose main diagonal contains zeros.

A special case arises for classification problems with ordinal classes. The value of $c\_{ij}$ should reflect the extent of the difference between $i$ and $j$. Thus, it is common to assume that $c\_{ij}=0$ when $i=j$. In addition, the cost $c\_{ij}$ is assumed to be larger when $i$ is further away from $j$. Two common functions satisfy the requirements and have been widely used in practice \autocite[Section~3.1]{Ruan2014}:

\begin{itemize}

\item \makebox[4cm][l]{Absolute cost vectors} $c\_{ij}=\left|i-j\right|$

\item \makebox[4cm][l]{Squared cost vectors} $c\_{ij}=\left(i-j\right)^{2}$

\end{itemize}

Note that the squared cost charges more than the absolute cost when $i$ is further away from $j$. The cost matrix of a multiclass classification problem with $n$ classes using absolute cost vectors looks like this \autocite[Section~3]{Kotsiantis2004}:

\[\left[ \begin{array}{ccccc}

{0} & {1} & {2} & {\dots} & {n-1} \\

{1} & {0} & {1} & {\dots} & {n-2} \\

{\dots} & {\dots} & {\dots} & {\dots} & {\dots} \\

{n-2} & {\dots} & {1} & {0} & {1} \\

{n-1} & {\dots} & {2} & {1} & {0}

\end{array}\right]\]

By squaring each element of the matrix, one obtains the matrix of squared cost vectors. In both cases it is a symmetrical matrix, i.e. $c\_{ij}=c\_{ji}\forall i,j\in\{1, \dots, m\}$. In some cases, it may be useful to weight the over- and underestimation of classes differently. If the overestimation of a class is to be penalized more severely than the underestimation, all elements to the right of the main diagonal are multiplied by a constant value $\lambda>1,\lambda\in\mathbb{R}$. In a comparable way all values of the matrix can be adjusted as required to the problem which has to be solved.

Given such a cost matrix, you can calculate the cost of a particular learned model on a given test set just by summing the relevant elements of the cost matrix for the model’s prediction for each test instance. Here, the costs are ignored when making predictions, but taken into account when evaluating them \autocite[Chapter~5.7]{Witten2005}. In this way the classifier obviously does not have to deliver the cost-optimal result. However, there are different possibilities to include costs in advance \autocite[Chapter~2.3]{Qin2010}:

\begin{itemize}

\item Change of the class distribution of the training data

This approach incorporates the misclassification cost into the data pre-processing step by re-sampling or re-weighting the training data in proportion of their misclassification cost.

\item Modification of the learning algorithm

This approach modifies the error-based classifiers directly to handle misclassification cost, but each classifier needs to be modified separately.

\item Boosting approach

This approach generates a set of different weak classifiers in sequential trial and then constructs a composite classifier by voting them. The advantage of this approach is that it is applicable to any kind of error-based classifiers.

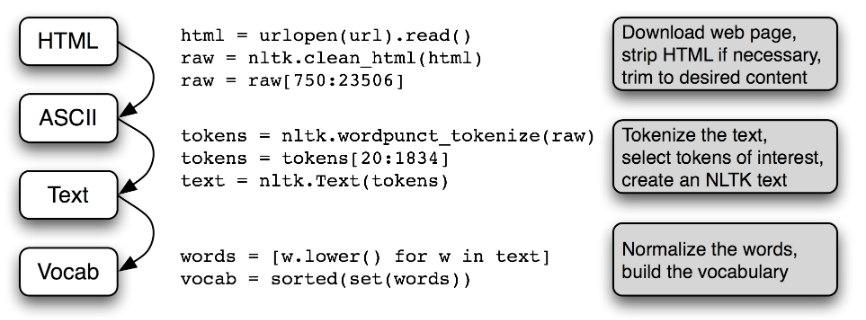
\item Conditional probability estimates / Direct cost-sensitive learning

This approach incorporates the misclassification cost into the data post-processing step by using the probability estimation generated by error-based classifiers and the cost function to directly compute the optimal class for each test example. This approach is easy to implement, but needs good calibration methods to generate accurate probability estimation.

\end{itemize}

Note that the use of the costs as in one of the approaches presented here does not preclude a subsequent evaluation of the costs. In this way it can be checked whether a classifier who incorporates the costs into the learning process is actually superior to a classifier without knowledge of the costs.

NLP & NLTK



further reading chapter 7 (POS-tagging, chunking)

Word2Vec / Word Embedding

neuronal network with two layers

idea: word vector, the more similar words, the more similar the vectors

adding of vectors

comparison of vectors based on angle (0°-90°(?))

CBOW (context 🡪 word) vs skip-gram (word 🡪 context)

glove