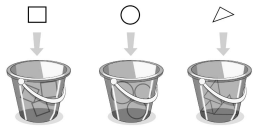


Classification Algorithms
Text Analysis and Natural Language Processing

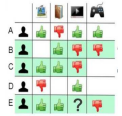
Miłosz Kadziński

Institute of Computing Science
Poznan University of Technology, Poland

www.cs.put.poznan.pl/mkadzinski/iai

[1] Hello everyone. In the previous lecture, we talked about two aspects. First, we discussed what artificial intelligence is all about. We tried to define the field and concluded that it is science and a set of computational technologies inspired by the way people use their bodies and nervous systems to sense, learn, reason, and take action. We also discussed the main application areas of AI, indicating that even if in the previous century it was mostly an academic field of study, in the last twenty years, it started to deliver technologies that are improving human wealth, health, safety and productivity. To prove this, we showed some examples of applications in transportation, healthcare, education, or entertainment. Moreover, we discussed some recent trends in AI and named some hot topics prevailing in the last decade. I promised you that at this course, we would be mostly talking about various data-driven techniques and problem-solving approaches. Then, we already started to enter the world of AI by discussing the problem of clustering. In particular, we presented two approaches representing various streams of clustering methods. You already know how K-means and agglomerative hierarchical clustering work. Today we will continue with data mining techniques. That is why I would like to start by elaborating a bit on the general aim of data mining.

Problems Considered in Data Mining



Data mining: process of ***discovering patterns*** in data sets

- *given a set of **data points** described in terms of a set of **attributes**, the **task** may be*:

Prediction / regression
Predict *a value* of some attribute based on the values of other attributes (e.g., sales amount of a new product or a rating for a moving in a recommender system)

Association
Discover *dependencies* predicting occurrence of an item based on occurrences of other items (e.g., identify items bought together by sufficiently many customers)

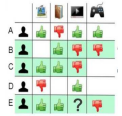
Clustering
Find *groups* such that data points in one group are more similar to one another and data points in separate groups are less similar to one another

TODAY

Classification
Find a *model* for predicting *class* attribute as a function of the values of other attributes

[2] In general, data mining concerns the exploitation of points contained in the data set. The points are described in terms of a set of attributes, characteristics, or features. This exploitation should result in discovering some patterns which are possibly present in the data sets. Depending on the aim of the analysis, we can distinguish four basic tasks. Let me first talk about prediction or regression. Imagine you are the owner of a company introducing a new product on the market and you wish to estimate its sales amount. Alternatively, think of yourself as a user of the movie rating system. You and other users fill marks for different movies based on which it would be possible to know the chances you would like some other movies you have not seen yet. In both these scenarios, we wish to predict some attributes' value based on other attributes' values. The second task is called association. Imagine you are running an online shop, and you offer a variety of products. You may wish to identify which items are bought together by many customers. Such associations can be subsequently used to predict the occurrence of some item based on the occurrence of other items. This is frequently done in both online shops where you are suggested products you may be interested in based on other products you have bought or viewed, and in traditional shops, which may adjust their offers based on the previously discovered associations.

Problems Considered in Data Mining



Data mining: process of ***discovering patterns*** in data sets

- *given a set of **data points** described in terms of a set of **attributes**, the **task** may be*:

Prediction / regression

Predict *a value* of some attribute based on the values of other attributes (e.g., sales amount of a new product or a rating for a moving in a recommender system)

Association

Discover *dependencies* predicting occurrence of an item based on occurrences of other items (e.g., identify items bought together by sufficiently many customers)

Clustering

Find *groups* such that data points in one group are more similar to one another and data points in separate groups are less similar to one another

TODAY

Classification

Find a *model* for predicting *class* attribute as a function of the values of other attributes

[2a] You already know the third type of task considered in data mining, which is clustering. It consists of finding groups of items such that those contained in a single group are similar to one another, but those contained in separate groups are less similar to one another. In the last lecture, we discussed plenty of realistic applications, in which clustering was very useful, just to mention the identification of users with similar interests or access patterns, documents clustered in a web searcher, genes with similar expression patterns, or houses of the same type. Today we will talk about classification, which can be explained in reference to prediction and clustering. On the one hand, in classification, we wish to predict the value of some attribute based on the values of other attributes. On the other hand, the attribute whose value we wish to predict has a special role. It is called a class attribute, and in this way, it decides upon the assignment to a group of objects which are similar to some extend. However, the classes to which the objects can be assigned need be defined in advance, and for this reason, they need to have some concrete interpretation.

Examples of Text Classification (1)

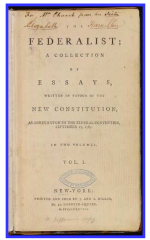
Subject: Important notice!
Date: November 25, 2019 12:00 PM
From: John Odonkor (john.odonkor@123.com)

Is this spam?

Dearest one,
I was a servant at the presidential palace in Burkina Faso until when the president was overthrown. Presently we are in Mali and the president is in hiding here. He asked me to seek your assistance to help us move some money abroad. If you feel you can help us, kindly reply. If however you are not willing to do so, please do not expose what I have revealed to you. I look forward to your quick response.

Who wrote which Federalist papers?

- 1787-8: anonymous essays try to **convince NY to ratify U.S Constitution**: Jay, Madison, Hamilton
- Authorship of 12 of the letters in dispute
- Solved by Mosteller and Wallace using Bayesian methods in 1963



[3] We will be talking about classification for two lectures. However, today we want to talk a bit about natural language processing. That is why we will limit the context of possible applications to the classification of text documents. I would like to start by discussing some examples of classification contexts that are characteristic for this domain. Consider an e-mail presented in the slide that I got from a servant at the presidential palace in Burkina Faso. This guy is now together with the president in Mali, and they ask me for some help in transferring the money. I wonder if you could advise me if I should help them or not? In any case, from the text classification perspective, this e-mail, as well as any other one that reaches my e-mail box, could be a subject of a binary classification problem with classes named as spam or no-spam. The other example refers to the history of the United States and, more specifically, to so-called federalist papers. It is a collection of 85 articles and essays published in different journals and newspapers to promote the ratification of the US constitution. They were written by John Jay, Alexander Hamilton, and James Madison. However, at the time of publication, the authors attempted to hide their identities because they were taking an active part in the convention on the ratification of the constitution. Many years later, Hamilton claimed that he wrote 63 papers, but Madison did not confirm it. Only in the second half of the 20th century, the problem of authorship attribution was resolved using some Bayesian classifier that you will get to know during this lecture.

Examples of Text Classification (2)

Positive or negative movie review?

- *"Unbelievably disappointing"*
- *"Full of richly applied satire, and some great plot twists"*
- *"This is the greatest screwball comedy ever filmed"*
- *"It was pathetic. The worst part about it was the boxing scenes."*

research paper

?

***Journal categories*:**

Artificial intelligence
Data science
Decision analysis
Decision support systems
Machine learning
Operational research
....

[4] Now, as a third example, think of reviews and, more specifically, about movie reviews. Based on the character of words used in the review, it is possible to predict whether it is positive or negative. For example, expressions such as > unbelievably disappointing < or > the worst part < suggest a negative attitude, whereas words such as > great plot twists < or > the greatest comedy < suggest a positive attitude. This kind of classification is formally called sentiment analysis. In the more scientific world I live in, we - as scientists - are expected to publish papers on a specific research topic. However, the journals publishing the papers are usually not very focused, and you can find there articles raising different, though interrelated subjects. To identify what the main topic for a given paper is can be seen as a classification task. It simply helps to improve the presentation of the papers published in a given journal in a more structured way. In the slide, you have one of my papers and a set of categories that can be considered within such a task.

Examples of Text Classification (3)

Given an occurrence of a word, decide which sense, or meaning, was intended: example for "**run**"

Categories: use word sense labels (run_1, run_2, etc.) to name them

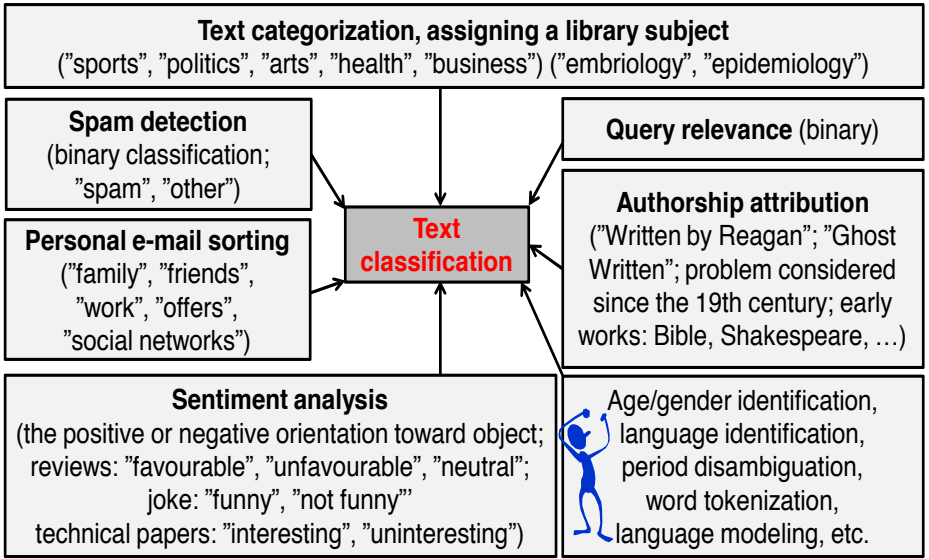
- **run_1**: move swiftly (I **ran** to the store.)
- **run_2**: operate (I **run** a store.)
- **run_3**: flow (Water **runs** from the spring.)
- **run_4**: length of torn stitches (Her stockings had a **run**.)

Features: describe the *context* of the word we want to disambiguate

- ***near(w)***: is the given word near an occurrence of word w?
- ***pos***: the word's part of speech
- ***left(w)***: is the word immediately preceded by the word w?

[5] I would also like to mention an interesting example that we will not consider today because it does not deal with the classification of documents, but it is closely related as it refers to the classification of words. More precisely, the common problem is to consider the so-called word disambiguation. The task here is to determine a sense or a meaning for a given word. Sometimes it is not easy, because if you considered words such as > run <, they have many different meanings such as moving swiftly, operating, running, or even as a noun related to stockings. Such a problem can be usually solved based on the analysis of its part of speech and words, which appear near an occurrence of a given word.

Examples of Text Classification (4)



[6] To sum up this part, even if you limit the classification task to texts or documents, a variety of problems that you may encounter in practice is just enormous. We have already talked about text categorization or assigning a library subject, which is a kind of task in which the interpretation of classes is the most straightforward. We have mentioned spam detection, authorship attribution, and sentiment analysis, where these classes are already more specific. For example, in sentiment analysis, they refer to an orientation toward an object, which can be a movie, a product, a joke, or a paper. But think about your mailboxes, where messages are assigned to different categories. Think of using a web query system, where you ask some query, and each document needs to be classified as either relevant or non-relevant. We can list even more such problems, just to mention age, gender, or language identification.

How to Perform Classification? (1)

Manual classification

- Accurate when job is done by **experts**
- Consistent when the problem size and team is **small**
- Difficult and expensive to scale
- **Yahoo!** (~200 people for manual labeling of web pages using a hierarchy of 500,000 categories)
- **MEDLINE** - bibliographic database of life sciences/biomedical information: $2 million/year for manual indexing of journal papers (18,000 categories)

Hand-coded rule-based classifiers

- **Humans encode knowledge** of what constitutes membership in category
- Widely used in government and enterprise, building and maintaining the rules is expensive, difficult, and raises some consistency issues (size)
- Accuracy can be high if **rules** are carefully refined over time by experts
- Example for spam: *black-list-address* OR ("*dollars*" AND "*president*")

[7] A relevant question is: How to perform classification? I will first provide two answers that will not be my final ones. First, we can do it manually, provided that we are experts in a given domain. However, the problems are that it is very expensive and difficult. It is not easy to ensure consistency between various experts' judgments, and it works well when the problem size is relatively small. Still, such a manual classification is performed by some companies. On the one hand, Yahoo had a large team of experts classifying web pages and images, and it was well known for maintaining a hierarchy of a few hundred thousand categories. On the other hand, Medline, a bibliographic database of biomedical information, spends millions of dollars on manually indexing journal papers into several thousand categories. The second way of classification also involves experts, but instead of classifying the text manually, the experts are expected to define a model on what decides on membership in the category. For example, in spam detection, we may refer to the address of an e-mail and some words as dollars or president that are frequently encountered in spam, and rarely in regular e-mails. This way of proceeding is often implemented by governments or enterprises, where the models have the form of rules carefully refined over time by experts. Similarly to the previous solution, this task is expensive, difficult, and raises some consistency issues.

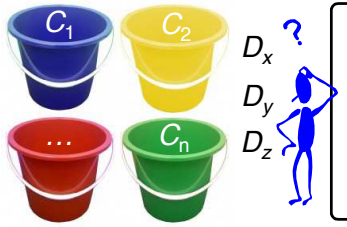
How to Perform Classification? (2)

Input

- A **fixed set of class** or category labels $C = \{C_1, C_2, \ldots, C_n\}$
- A training set of m hand-labeled documents $\{(D_1, C_2), \ldots, (D_m, C_j)\}$
- A document $D_x \in X$ *to be classified*

Output

A **learned classifier** predicts class $c(D_x)$ for D_x, where $c(D_x) \in C$ *and* c is a function whose domain is X and whose range is C



- Create classifier (model) and use it to classify new data (i.e., predict a discrete, nominal value (category))
- *There can be many documents to be classified in set X*

Classification algorithms Bayesian, Distance-based, Decision trees, Support Vector Machines, Neural networks, Association-based, Genetic

[8] As my final solution, I would like to propose an idea of combining the previous two solutions. Let us first gather a so-called training set of a limited number of hand-labeled documents. That is, for each document in this set, we would collect information on its assignment to one of n pre-defined classes or categories. Then, to classify a document for which such a label is unknown, we would use a classification model. But instead of asking the experts to define it directly, we would learn it from the collected data. Once we have it for multiple documents, we could use to predict a nominal, discrete value, called category. In the literature on data mining and machine learning, there exist different algorithms for constructing such models, and we will get to know plenty of them during this course. For example, the next lecture will be fully devoted to decision trees. However, today we will start with a pair of the most famous classifiers. One of them will exploit similarities or distances between documents, whereas the other will be based on the Bayes theorem, so we will compute some probabilities.

Classification - Three Steps

Model construction (learning)

- Each instance (example) is assumed to belong to a pre-defined class, as determined by one of the attributes called a **target attribute**
- The values of the target attribute are the **class labels**
- The set of all instances used for learning the model is called **training set**

Model evaluation (*predictive accuracy*) – *closer to the end of this lecture*

Model use (classification)

- The model is used to **classify unseen instances** (i.e., to predict the class labels for new unclassified instances)
- Predict the value of an actual attribute

[9] When talking about classification, we will distinguish three tasks: model construction, evaluation, and use. During the construction or equivalently learning phase, we will use a set of examples assumed to belong to some pre-defined classes. Formally, we will talk about the target attribute whose values are class labels, and we will treat a set of instances for which the classification is known as a training set. Then, we will evaluate the model we have learned. I will show how to perform such an evaluation and what measures can be used for this purpose in the context of classification. Finally, we will use the model for performing classification of previously unseen objects or instances. That is, for the unclassified instances, we will try to predict their class labels so that these instances can be subsequently treated adequately.

Representation of Text Documents (1)

Given a **description of an instance** $D_x \in X$ in terms of features or attributes in space X *and* a **fixed set of class** or category labels $C = \{C_1, C_2, \ldots, C_n\}$, a **classification task** is to determine the class of D_x

Need for **features/attributes representating** document

D_x: *I was a servant at the presidential palace in Burkina Faso until when the president was overthrown. Presently we are in Mali and the president is in hiding here. He asked me to seek your assistance to help us move some money abroad. If you feel you can help us, kindly reply. If however you are not willing to do so, please do not expose what I have revealed to you.*

VECTOR SPACE MODEL

terms (keywords) used to represent the document

	t_1	t_2	t_3	t_4	t_5
document D_x	…	…	…	…	…

values indicating the relevance of a given term for the document

[10] If we came back to what classification is about, we would notice that the task was formulated as follows: given a description of an object in terms of features or attributes and a fixed set of classes, determine the object's class. Today, we limit our interest to objects, which are documents, but there is no doubt that we need some attributes, features, or characteristics that can be used to describe the documents and subsequently referred to within the classification task. If you looked at the e-mail I received from this guy from Burkina Faso, you should intuitively feel that it would be challenging to use such a raw representation within the classification algorithms. We have words with different inflectional endings, some of these words are clearly irrelevant for the classification; they also form sentences which, apart from the words, contain punctuation. Instead of analyzing a raw text, we will use the so-called vector space model. So our objects will be documents, and they will be described in terms of a finite set of terms or keywords. In this regard, each document will be represented as a vector of numbers, and the values in the vector will indicate the relevance of a given term for the document.

Processing of a “Raw" Text Document

Tokenization**:* chopping it up into pieces, called ***tokens, at the same time throwing away certain characters/words, such as punctuation

- I, was, a, servant, at, the, presidental, palace, ….

Stopwords**:* removing from the vocabulary ***ubiquitous words which are of little value in helping the purpose of analysis

- I, ~~was~~, ~~a~~, servant, ~~at~~, ~~the~~, presidental, palace, ….

Text normalization**:* transforming text into a single ***canonical form that it might not have had before (text transformations vs. relations)

- anti-democratic vs. antidemocratic; money vs. cash

Stemming**:* identify the ***stem of a word and use it in lieu of the word itself

Lemmatization*:** return the base or dictionary form of a word, which is known as the ***lemma by means of the morphological analysis of words, typically aiming to remove *inflectional* endings

- president vs. presidenti~~al~~

[11] To transform a raw text document to such a vector, several stages need to be conducted. I will discuss them briefly, and you will get to know more in the fifth and sixth semesters. First, we need to chop the text into pieces, called tokens, and get away of certain characters such as dots or commas. This stage is called tokenization. Second, we need to remove some words of little value in helping the purpose of classification. These include conjunctions such as and, articles such as a or the, some pronouns, or verbs commonly used in every second sentence. All such words are called stopwords. Third, we normalize the text so that tokens in different forms but denoting the same concept are mapped into the same canonical form. There are two ways of performing such a normalization. You may automatically transform the text by means of some simple rule, and you have an example of the word anti-democratic, which, with or without a hyphen, means exactly the same. Or you may maintain some relations between words so that words meaning exactly the same, but not having anything in common in a textual form, like money and cash, car and automobile, are still mapped on the same token. Finally, we wish to discover the basic form of each token, which is called a stem or a lemma, to get rid of different inflectional endings or differences between various parts of speech such as nouns, adjectives, or verbs. For this, we may use some heuristics approaches, which are called stemmers or formal morphological analysis, using a dictionary.

Representation of Text Documents (2)

Tranformation of a raw text by means of: tokenization, elimination of stopwords, normalization, and lematization/stemming

D_x: *I XXX ~~a~~ XXXXXXXXXX presidential XXXXXXXXXXXXXXX ~~until when~~ XXX president XXXXXXXXXX~~.~~ XXXX we XXXXXXXXXXXX president XXXXXXXX~~.~~ XXXXXXXX ~~asked~~ me XXXXXXXXXXX help us XXXXXXXXXX money XXXX~~.~~ XXXXXXXXXX help XXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX~~.~~*

DICTIONARY = A SET OF KEYWORDS

$\mathbf{T} = \{t_1 = \text{president}, t_2 = \text{ask}, t_3 = \text{bank}, t_4 = \text{help}, t_5 = \text{money}\}$

$|\text{DICT}| = |\mathbf{T}| = 5$

BINARY

if t_j occured, *then* 1; *otherwise*, 0

	t_1	t_2	t_3	t_4	t_5
D_x	1	1	0	1	1

BAG-OF-WORDS (BOW)

number of occurences of t_j

	t_1	t_2	t_3	t_4	t_5
D_x	3	1	0	2	1

TERM-FREQUENCY (TF)

normalized BOW *(e.g., divided through the maximal number of occurences of any term)*

	t_1	t_2	t_3	t_4	t_5
D_x	3/3	1/3	0/3	2/3	1/3

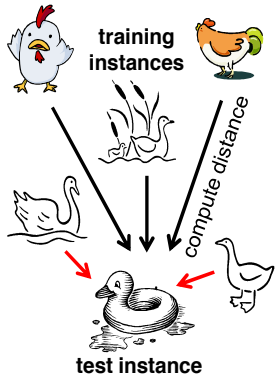
[12] What remains after all these stages is very much different from the initial raw document. Also, we cannot afford to represent each document in terms of a few hundred thousand terms. So we need to select their subset called dictionary, and each keyword in this subset will correspond to a single component in a vector. Usually, such terms are the ones that appear most frequently but only in some documents. In this way, they can be treated as good discriminators. Now, I would like to discuss three different representations, which indicate how to fill the vectors' numbers. In the binary representation, we have only 1s and 0s, 1s in case a given term appears in a doc, and 0 otherwise. My example doc would have 1s on the features related to the words such as president or money, and 0 on the attribute bank. In the second representation, called bag of words, we store the number of occurrences of each keyword. That is, where we had 0s in the binary representation, we would still have 0s, but instead of 1s we would just have the exact numbers of occurrences such as 3 for president and 2 for help. Finally, in the term-frequency representation, we normalize the bag of words vectors so that different docs become comparable. The most common normalization is to divide the bag of words numbers through the maximal number of occurrences of any term in a given doc. Here, we would have three in the denominator. These values represent the relative importance of each term in a given document. Please note that it is not possible to reconstruct the original text based on all these representations. We simply lose information on how the original sentences looked like and how the words co-occurred. But at least we have some internal representation on which we can run the classification algorithms.

Distance-based Classification

- Classify new instances based on their similarity (distance) from instances we have seen before

Basic idea

- Save all previously encountered instances
- Given a new instance, find those instances that are *most similar* to the new one
- Assign new instance to the same class as these “nearest neighbors”



Lazy classifiers

- The approach defers all of the real work until new instance is obtained; no attempt is made to learn a generalized model from the training set
- Less data preprocessing and model evaluation, but more work has to be done at classification time

[13] The first group of classifiers I would like to discuss are distance-based. The general idea is to classify a new instance based on its similarity or distance from instances we have seen before. You may think that the construction of a classification model needs to be very complex. Not in here. In turn, we just store our learning examples for which the classification is already known and retain all of the real work until a new instance, to be classified, is available. Only then, we would compute its similarity to the learning objects and identify these, which are the most similar to the new one. This is very intuitive as such nearest neighbors are the best examples to learn from. For this reason, we would assign a new instance to the same class as those nearest neighbors.

Popular Similarity Measures for Text Documents

JACCARD SIMILARITY $Jac(x,y) = \frac{|T(x) \cap T(y)|}{|T(x) \cup T(y)|}$

	t_1	t_2	t_3	t_4	t_5
D_x	1	1	0	1	1
D_y	1	0	1	1	1
D_z	1	0	0	0	1

how many terms are joint (in the product)?

$Jac(D_x,D_y) = \frac{3}{5}$ $Jac(D_x,D_z) = \frac{2}{4}$

how many terms overall (in the sum of D_x and D_y)?

COSINE SIMILARITY $cos(x,y) = \frac{\sum_j x_j \cdot y_j}{|x| \cdot |y|}$

	t_1	t_2	t_3	t_4	t_5
D_x	1	1/3	0	2/3	1/3
D_y	1/2	1/2	0	1	1/2
D_z	0	1/2	1	0	1

dot product

$cos(D_x,D_y) = \frac{1.5}{1.29 \cdot 1.32} = 0.88$

product of vector lengths

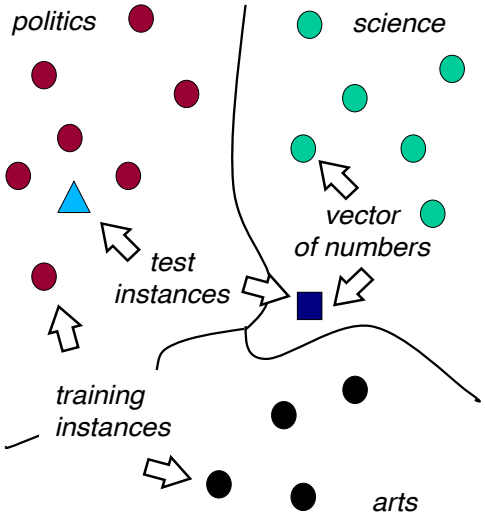
more similar lesser angle

$cos(D_x,D_z) = \frac{0.5}{1.29 \cdot 1.50} = 0.26$

OTHER DISTANCE/SIMILARITY MEASURES CAN BE USED

[14] Before moving to the algorithms, I would like to mention a pair of similarity metrics, which are most frequently used in comparing documents. In case you analyze the binary vectors, it is common to use the so-called Jaccard coefficient. It indicates how many out of terms appearing in a pair of compared docs are joint for them. So consider Dx and Dy; five terms are appearing in them, but they share only three of these terms. In the same spirit, for Dx and Dz - four terms can be found in both docs, but only two in both Dx and Dz. Clearly, the closer this measure to 1, the more similar the objects, and the closer it is to 0, the less similar they are. The same interpretation holds for the other measure, which is used in the context of other representations. It is a cosine measure that we already discussed in the previous lecture. Note that cosine quantifies an angle between a pair of vectors. The lesser this angle is, the more similar are the documents as this means they involve the same terms, and these terms are similarly important for these docs. Obviously, other distance or similarity measures can be used, including Euclidean, Chebyshev, or Manhattan distances that you already know.

Rocchio Classification

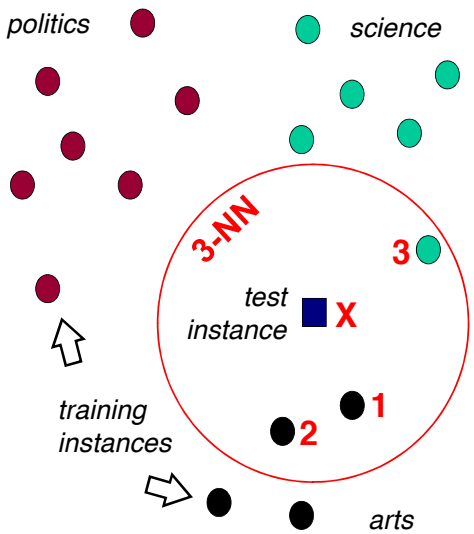


Basic Idea

- How to find separators?
- Separators equally distanced from the class centroids
- Assign instance to the class of training sample whose centroid is closest to it
- **Nearest centroid classifier**
- No guarantee to reproduce classification for training instances

[15] Coming to the classification algorithms, consider this example, where different symbols represent documents, that we already know are just vectors of numbers. We have docs from three classes: politics, science, and the arts. Several of them constitute our learning set, but the square and the triangle are docs for which the class is unknown. The first, very simple distance-based classifier I would like to discuss is called the Rocchio classifier. Its underlying idea consists in drawing the boundaries between classes. Specifically, it assumes that those separators are put in equal distances from the centroids of different classes. You already know that a centroid is an average representation of objects in a given class. So assume our boundaries are represented with solid lines in the slide We would assign a given document to the class to which area it would belong. Formally, this means that we would assign it to a class whose centroid is closest to it. This algorithm has two significant drawbacks. First, it does not guarantee that the classification of our learning objects would be reproduced. I have a simple example in the figure where it did not happen, but the reality is more complex, and the docs from a given class are usually not so well-separated. Second, on a more intuitive level, if you looked into the classification of a triangle as politics, it seems OK, but to judge a square as science is already controversial because it is far away from the science documents and much closer to the arts docs. This last observation suggests a new solution.

k-Nearest Neighbour Classifier



Model

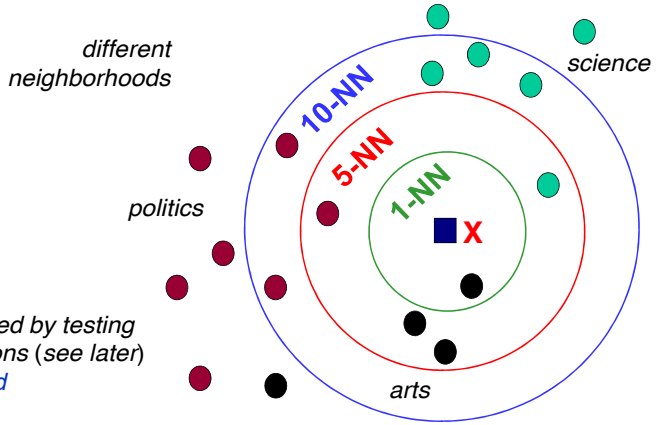
- Given instance X find its ***k* nearest neighbors**
- Variety of distance or similarity measures can be used
- Requires comparison of X with all training instances

Classification

- Find the class label for each of the *k* neighbors
- Use a voting approach to determine the majority class among the neighbors
- Assign the majority class to X

[16] Perhaps instead of analyzing distances from the centroids, we should compute, for a given object, its distances from all learning or training objects. Then, we should identify the closest or the most similar ones. Let us say that we would analyze a limited number of such nearest neighbors, and denote this number by k. In the slide, k is equal to three. Subsequently, we should analyze the class labels for the nearest neighbors and use some voting approach to determine the class, which has the greatest support. Such a majority class should be suggested as a category for a test instance. This idea underlies the k-nearest neighbors algorithm or, in short, k-NN with k meaning the size of the neighborhood analyzed to predict a class.

How to Select *k*?



value of k
may be learned by testing
different options (*see later*)
or pre-defined

using **k=1**, *subject to errors* (*single atypical example, noise in the category*)
with ***large k***, *you loose the underlying idea* (*prone to bias*)
k is often taken to be odd in order to avoid ties (*works in case of two classes*)
if there are more than two classes, take k to be the number of classes plus one

[17] An important question is how to select an appropriate value of k? Well, there are two approaches. We may try to learn it, and we will talk about it later on. Or we may use some pre-defined value of k. In this regard, there are a few suggestions that may help in its setting. You should rather not use k equal to 1, because our learning data is usually not perfect. If it contains some noise or atypical example, this error would be propagated on the test instances. If you opted for larger k, such as ten used in the figure, where the number of objects is rather small, you would lose the essence of the algorithm and start to predict based on the objects which are not so similar to the analyzed one. As a rule of thumb, k is often taken to be an odd small number such as three or five, in case two classes are considered. This allows for avoiding ties in the voting. When more classes are involved, k is often set to their number increased by one, which means that for our example with three classes we would use the neighborhood size equal to four.

Voting Functions

Once the nearest neighbors are identified, the "**votes**" of these neighbors must be **combined to generate a prediction**

- Regular voting (democracy): each neighbor has a single vote

  A: 1 (D1) + 1 (D2) = **2** **B**: 1 (D3) + 1 (D4) + 1 (D5) = **3** ⇨ conf. level **B (60%)**

Use 5-NN	**D1**	**D2**	**D3**	**D4**	**D5**	D6	D7	D8	D9	D10		**X**
Class	**A**	**A**	**B**	**B**	**B**	A	B	B	A	A		**?**
Similarity with X	**0.9**	**0.8**	**0.6**	**0.5**	**0.5**	0.4	0.4	0.2	0.1	0.0		**1.0**

- Closer neighbors get higher weights (proportional to the similarity)
- The weighted sum for each class gives the comibined score for that class

  A: 0.9 (D1) + 0.8 (D2) = **1.7** **B**: 0.6 (D3) + 0.5 (D4) + 0.5 (D5) = **1.6** ⇨ **A**
- Introduces enough variation to prevent ties in most cases

[18] Once you identify the nearest neighbors and collect their respective classes, there is still an issue on how to perform voting. There are two basic approaches for doing this. In one of them, each of the nearest neighbors is attributed the same voting power of one. So assume we have ten learning examples, X is a doc to be classified, and we use 5-NN. Docs from D1 to D5 are identified as the nearest neighbors. Two of them vote for class A and the remaining three for class B. Naturally, B would be suggested for X as the majority class. However, you may note that D1 and D2, which are learning examples from class A are more similar to X that D3, D4, and D5. In the other approach, instead of assuming the voting powers to be the same, they are equal to similarity measures so that closer neighbors get higher weights. In our example, the classification for X would change because the support given to class A would be greater than for class B. From a practical viewpoint, this approach is also nice because it prevents ties in most cases as it is more difficult to obtain the same support levels for different classes.

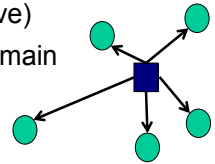
Main Characteristics of *k*-Nearest Neighbour Classifier

Lazy classifiers

- Does not compute anything beoynd storing the examples
- For each document, you need to compare its similarity to all training instances (scanning through all feature values)

Instance- or case-based learning (simple and effective)

- Each training instance is a case from the problem domain
- Raw training instances are used to make prediction



Memory-based learning

- ☺ Works well with a small number of features/attributes
- ☹ In high dimensions, points that may be similar may have very large distances
- ☹ Need to set *k*, a similarity/distance measure, and voting scheme
- ☹ Output not good in explaining why a category is relevant

[19] K-NN is the most famous representative of the so-called lazy classifiers. Indeed, it does nothing until the need to be classified appears. No model is created, and more work needs to be done at the classification stage. K-NN also represents two other groups of algorithms called instance-based and memory-based. It is so because it directly employs the training instances to make a prediction and just stores them in the memory to reuse later. This algorithm is reported to work well when the number of features is relatively small. In the case of working with a large number of features, the distances are a bit harder to predict, and you may think that some objects are similar, which is not confirmed by raw numbers. Also, you need to parameterize this approach so that it works well. This involves the value of k, similarity measure, and voting scheme, and it is no surprise that its performance strongly depends on what options are selected.

Bayesian Learning

Bayes's theorem plays a critical role in probabilistic learning and classification

- **Uses prior probability of each class** given no information about an item
- Classification produces **a posterior probability distribution over the possible classes given a description of an item**

- A *data instance X* with an unknown class label
- Hypothesis *H* that *X belongs to a specific class C*
- The *conditional probability* of hypothesis *H* given *observation X,* P*(H|X),* follows the **Bayes's theorem**:

$$P(H|X) = \frac{P(X|H) \cdot P(H)}{P(X)}$$

- Practical difficulty: requires initial knowledge of many probabilities

[20] Now, I would like to pass to the second classifier. It would be representative of the large group of probabilistic algorithms in machine learning, which implement the scheme of Bayesian learning. As the name suggests, it will be based on the famous Bayes theorem, so we will estimate a probability distribution over the possible classes given a description of some item to be classified. Consider some instance X with an unknown class label. Our task is to compute the probability of a hypothesis that X belongs to a specific class. Formally, it will be a conditional probability of hypothesis H given item X. Since it cannot be computed directly, we would take advantage of the Bayes theorem, which says that such a probability is equal to the product of the conditional probability of item X given hypothesis H and probability of the hypothesis divided by the probability of item X. Even if the theorem helps to obtain some results, you see that still many probabilities need to be known or calculated.

Bayesian Classification (1)

- Let set of classes be $\{C_1, C_2, \ldots, C_n\}$
- Let X be description of an instance (e.g., vector representation)
- Determine class of X by computing for each class C_i

$$P(C_i|X) = \frac{P(X|C_i) \cdot P(C_i)}{P(X)}$$

- **Select C_i for which $P(C_i|X)$ is maximal**

- P(X) can be determined since classes are complete and disjoint:

$$\sum_{i=1,\ldots,n} P(C_i|X) = \sum_{i=1,\ldots,n} \frac{P(X|C_i) \cdot P(C_i)}{\boldsymbol{P(X)}} = 1$$

$$\mathbf{P(X)} = \sum_{i=1,\ldots,n} P(X|C_i) \cdot P(C_i)$$

- In practice, P(X) can be neglected (the same for $P(C_i|X)$, i=1,…,n):

$$P(C_1|X) = \frac{P(X|C_1) \cdot P(C_1)}{P(X)} \quad \cdots \quad P(C_n|X) = \frac{P(X|C_n) \cdot P(C_n)}{P(X)}$$

[21] Let me try to formalize things a bit. We wish to consider objects which are vectors of numbers and a classification task. In this perspective, we would need to verify the conditional probability for each of the n classes given a description of an object to be classified. Since we will apply a probabilistic classifier, it would select the class for which such a respective conditional probability is the greatest. Yet computing such probabilities is not an easy task because we need to know the probability of X given some class and the probabilities of X and a class. It sounds complicated, but we will try to approach it step by step. First, let me focus on the probability of an instance X, which is in the denominator. In general, since the set of classes is complete and they are disjoint, it would be possible to compute the probability of X. However, please note that it is not needed. Our task consists in computing the conditional probabilities for all classes, and P of X appears in the denominator of all such probabilities. As a result, it would not change the relative comparison between the probabilities for different classes. Since we wish to identify which is the highest, we can simply neglect the denominator.

Bayesian Classification (2)

$$P(C_i|X) \approx P(X|C_i) \cdot P(C_i)$$

- $P(C_i)$ are easily estimated from data
- N_i of the N training examples are in C_i

$\Rightarrow$ $P(C_i) = N_i / N$

Problem: too many possible combinations (exponential in *m* being the number of features/atttributes) to estimate all $P(X|C_i)$
Assume instance is a **conjunction of binary features/attributes**:

	t_1	t_2	…	t_m
X	1	1	…	0

$\Rightarrow$ $X = t_1 \wedge t_2 \wedge \ldots \wedge t_m$

$X = (t_1=1) \wedge (t_2=1) \wedge \ldots \wedge (t_m=0)$

If we assume features/attributes of an instance are **independent** given the class (C_i) (*conditionally independent*)

$$P(X|C_i) = P(t_1 \wedge t_2 \wedge \ldots \wedge t_m | C_i) = \prod_{j=1,\ldots,m} P(t_j|C_i)$$

- We need to know $P(t_j|C_i)$ for each feature and category
- Estimated based on observed frequencies in training data
- If N_{ij} of N_i examples in class C_i contain attribute t_j: $P(t_j|C_i) = N_{ij} / N_i$

[22] and agree that we will not compute exact probabilities, but just estimate their values. So that was easy; we got rid of one component. To compute the probabilities of a class is also easy. You just need to analyze an entire training set and verify how many instances belong to a given class. As simple as that. Unfortunately, things are not that straightforward with the last type of probability. To compute the probability of a given instance X provided some class Ci would require that X is present in exactly this form in this class and perhaps to get various results for different classes, some distinct numbers of X should be present in each class. It does not sound realistic, which means that we need to simplify it in a naïve way to approach the problem. Let us first assume that an instance is a conjunction of binary attributes. If we assumed that these features are independent, it would be possible to decompose this conditional probability to a product of conditional probabilities of all attributes separately appearing in exactly this form as in the test instance in the objects from class Ci. So in the context of the document visible in the slide, for t1 and t2 we would be interested in the probability of their occurrence in the training docs from class Ci, but for tm we would account for the probability of its non-occurrence exactly as specified by the test instance. We need to know these probabilities for each feature and class, but this is relatively easy. It is enough to analyze the ratio between the number of docs from class Ci containing the feature in precisely this form as in instance X and the number of all docs from class Ci. In this way, we would know how typical it would be judged for the docs in this class.

Bayesian Classification (3) - Spam Example

	t_1	t_2	t_3	t_4	SPAM
D1	1	1	1	0	no
D2	0	1	0	0	no
D3	1	1	1	0	no
D4	1	0	0	1	no
D5	1	0	1	1	yes
D6	0	1	1	1	yes

Training data

$P(no) = 4/6 \quad P(yes) = 2/6$

Probabilities

	P(t|no)	P(t|yes)
t_1	3/4	1/2
t_2	3/4	1/2
t_3	2/4	2/2
t_4	1/4	2/2

New e-mail containing t_3 and t_4

	t_1	t_2	t_3	t_4
X	0	0	1	1

Should it be classified as spam = "yes" or spam = "no"?

$P(C_i|X) \approx P(C_i) \cdot P(t_1=0|C_i) \cdot P(t_2=0|C_i) \cdot P(t_3=1|C_i) \cdot P(t_4=1|C_i)$

$\boldsymbol{P(no|X)} \approx 4/6 \cdot (1-3/4) \cdot (1-3/4) \cdot 2/4 \cdot 1/4 = \mathbf{0.0052}$

$\boldsymbol{P(yes|X)} \approx 2/6 \cdot (1-1/2) \cdot (1-1/2) \cdot 2/2 \cdot 2/2 = \mathbf{0.0833}$

$P(yes|X) > P(no|X) \Rightarrow$ *spam*

[23] Let me consider some example so that things become clear. We have a set of six training docs, which are represented by means of four terms. For example, D1 contains terms t1, t2, and t3, and does not contain term t4. We construct a spam filter, so our task is to detect some e-mails that may potentially be classified as spam. This task should now be carried out for instance X, which does not contain t1 and t2, but it contains t3 and t4. Our final aim is to compute the conditional probabilities for classes no-spam and yes-spam, given the description of X. For this, we need the probabilities of these classes and conditional probabilities of all terms in the form in which they appear in X in the e-mails of different classes. With the probabilities of classes, it is straightforward. We have 4 out of 6 training messages classified as no-spam, and the remaining 2 out of 6 as yes-spam. In this perspective, the chances that a random message would not be spam are twice as high. Now, let us talk about the features of X as they are represented in docs of the two classes. I gathered all conditional probabilities for different terms in the slide. For example, what is the probability of term 1 appearing in class no-spam? Well, there are 4 docs in this class, but only three of them contain t1. In the same, spirit the conditional probability of t3 appearing in class yes-spam is equal to 1, because 2 out of 2 docs in this class contain this term. To get the final results, we need to multiply many probabilities, that is, the probability of a given class and the conditional probabilities of t1 not appearing, t2 not appearing, t3 and t4 appearing in a given class. When analyzing the final numbers, we see that the conditional probability for class yes-spam is slightly higher, so X is more probable to be spam than not.

Bayesian Classification (4) - Smoothing

Estimating probabilities from small training sets is error-prone:

- If due only to chance, a rare feature, t_k, is always false in the training data, $\forall C_i : P(t_k|C_i) = 0$
- If t_k then occurs in a test example, X, the result is that $\forall C_i: P(X|C_i) = 0$ and $\forall C_i : P(C_i|X) = 0$

- To account for estimation from small samples, probability estimates are adjusted or smoothed

Laplace smoothing using an m-estimate assumes that each feature is given a prior probability, p, that is assumed to have been previously observed in a "virtual" sample of size m:

$$P(t_j|C_i) = \frac{N_{ij} + m \cdot p}{N_i + m}$$

- For binary features, p is simply assumed to be 0.5

[24] There is one small problem related to the use of such a naïve Bayes classifier. If you multiplied so may probabilities, it would be enough that one of them was equal to zero so that the resulting probability does not make much sense. Imagine that some feature is not present in the learning docs for some class and an example to be classified has that specific, rare feature. As a result, the conditional probability for such a class would ne be nullified. We wish to avoid it, and for this reason, we apply a technique called adjusting or smoothing. In the simplest case, instead of having zero, you would assume sth very small, like one hundredth. However, to do it more professionally, we would apply the so-called Laplace smoothing. Here, a derived probability is aggregating what we know from data with what is assumed to be a virtual sample, where each feature has some pre-defined probability of occurrence. For example, for a binary feature, the size of a sample m is equal to one, and the probability of either 0 or 1 is equal to 0.5. As a result, you add something positive to both the nominator and denominator, preventing this ratio from being equal to 0. Note that what you add to the nominator is lesser than what you add to the denominator.

Bayesian Classification (5) - Smoothing

	Doc	Content	SPAM
training	D1	business proposal	yes
training	D2	gold gold	yes
training	D3	gold bank business	yes
training	D4	bank transfer	no
training	D5	business business bank	no
test	**D6**	business business gold	?

$N_{yes\text{-}term} = 7$, $N_{no\text{-}term} = 5$

the number of occurences of t_j *in docs belonging to* class C_i

$$P(t_j|C_i) = \frac{N_{ij}}{N_{i\text{-}term}}$$

the number of all terms in docs belonging to class C_i

create mega-documents for each clas by concatenating all docs in a given class

P(yes) = 3/5

P(businesslyes) = 2/7

P(goldlyes) = 3/7

P(yesID6) ≈ 3/5·(2/7)²·3/7 = 0.0210

yes

P(no) = 2/5

P(businesslno) = 2/5

P(goldlno) = 0/5

P(noID6) ≈ 2/5·(2/5)²·0/5 = 0

[25] Now I would like to show you that naïve Bayes is just a general framework, which needs to be adapted to a given setting. So imagine we will use a bag of words representation, and we will not apply smoothing. We have a training set composed of 5 docs, 3 of them in class yes-spam, and 2 in class no-spam. We consider a test doc: business business gold, for which the class should be predicted. The probabilities of classes yes and no are estimated in the same way. However, since we have switched to bag of words, we compute the conditional probabilities of each term given some class in a different way. Specifically, for each class, we create a mega document by combining all learning docs in this class. Then, for each term appearing in the test doc, we divide the number of its occurrences in such a mega doc by the number of all terms in it. For example, you have 7 terms in the docs of class yes-spam and two occurrences of business, or five terms in the docs of class no-spam and 0 occurrences of gold. Then, you multiply all probabilities with the proviso that if some term occurs more than once in the test doc, the related probabilities are considered so many times as the number of its occurrence. Just look for the term business, which appears twice, and gold, which appears only once. Now, you see the final conditional probability for yes-spam is greater. Yet the problem is that results for class no-spam are hardly interpretable because of some zero, which nullified the result. So let us check how they would look in case of applying smoothing.

Bayesian Classification (6) - Smoothing

	Doc	Content	SPAM
training	D1	business proposal	yes
training	D2	gold gold	yes
training	D3	gold bank business	yes
training	D4	bank transfer	no
training	D5	business business bank	no
test	**D6**	business business gold	?

$|DICT| = 5$, $N_{yes\text{-}term} = 7$, $N_{no\text{-}term} = 5$

the number of occurences of t_j *in docs belonging to* class C_i

$$P(t_j|C_i) = \frac{N_{ij} + 1}{N_{i\text{-}term} + |DICT|}$$

"add one"

the number of all terms in docs belonging to class C_i

dictionary size

ADD ONE SMOOTHING

P(yes) = 3/5

P(business|yes) = (2+1)/(7+5) = 3/12

P(gold|yes) = (3+1)/(7+5) = 4/12

P(yes|D6) ≈ 3/5·(3/12)²·4/12 = 0.0125

yes

P(no) = 2/5

P(business|no) = (2+1)/(5+5) = 3/10

P(gold|no) = (0+1)/(5+5) = 1/10

P(no|D6) ≈ 2/5·(3/10)²·1/10 = 0.0036

[26] When dealing with bag of words, a standard way of smoothing the probabilities is called add one. That is, you assume that each term occurred more than one than it did. To model this, you add one to the nominator, but at the same time, you add the size of the dictionary to the denominator. This size is equal to the number of all distinct terms in the learning docs, in our case five. As a result, no conditional probability is equal to 0. Just look for gold in class no. The relative outcomes have not changed. Still, class yes would be suggested, but the numbers just got more interpretable.

Bayesian Classification (7) - Characteristics

☹ Huge drawback = independence assumption

Naive Bayes is not so naive

☺ Very **fast**, low storage requirements

☺ Robust to **irrelevant features** (cancel each other without influencing results)

☺ Very good in domains with many equally important features

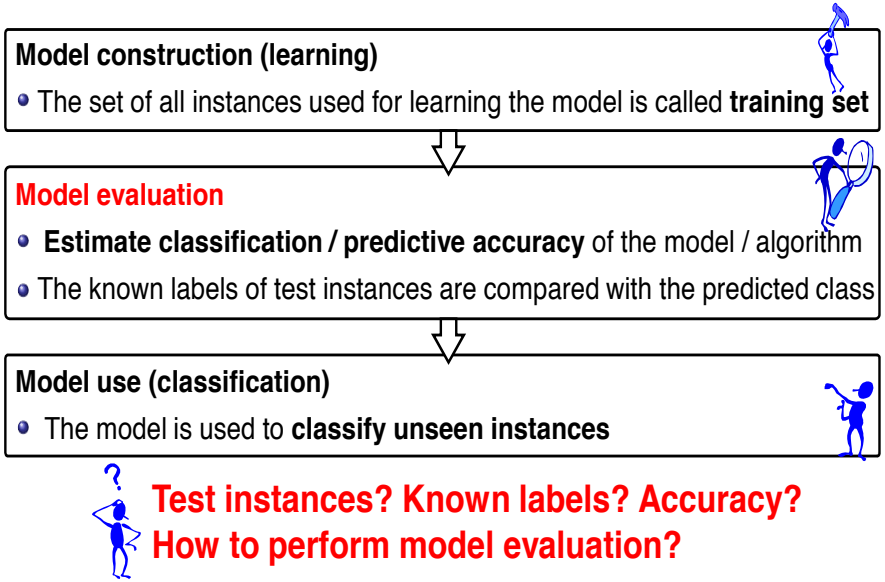
☺ The models are **incremental** (each training example can incrementally increase or decrease the probability that a hypothesis is correct)

☺ **Prior knowledg**e can be combined with observed data

☺ Optimal if the **independence** assumptions hold (if assumed independence is correct, then it is the Bayes Optimal Classifier for the problem)

[27] So let me still talk for a while about Naïve Bayes. It has one huge drawback. It incorporates this independence assumption, which is often hard to justify in practice. At the same time, it is the only way we can approach the problem. In the contexts where this assumption holds, it is not called naïve Bayes but rather Bayes optimal classifier. But it has even more advantages. The basic ones come from its speed and being robust to irrelevant features. Also, when new training examples appear, you can incrementally learn this classifier, just by updating the respective probabilities. What is more, you can combine the knowledge discovered from data with some expert knowledge that could be integrated into the probabilities of classes or features.

How to Evaluate the Classification Model?



[28] Until now, we have talked quite extensively about model learning and model use, but we have neglected the phase of model evaluation. This phase should answer how good is the constructed classifier, which formally boils down to estimating its classification or predictive accuracy. Such a process is based on the analysis of test instances for which the class labels are known, and they are compared with what is suggested by the model. Now, you may think: hey, what are test instances, where the known labels come from, what exactly is accuracy, and in general, how to perform model evaluation? We will try to answer these questions in the next minutes.

Training vs.Test Data

Train data

Using training set for model evaluation

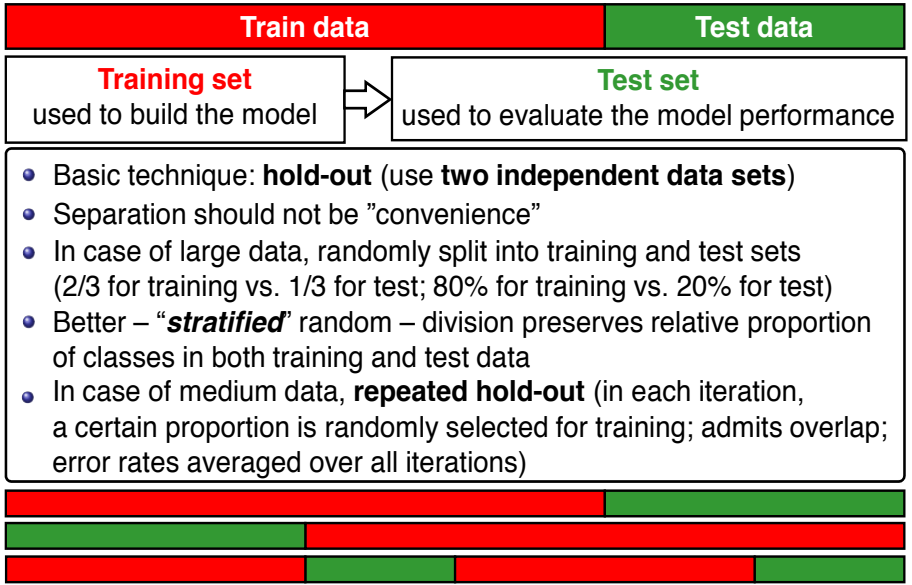
- **Very bad idea!**
- Not represenative and cheating – seen all test instances in training
- 1-NN would always be the best technique
- New data will probably not be exactly the same as training data
- *Overfitting* – fitting the training data too precisely – usually leads to poor results on new data (*we will come back to this next week*)

Using test set for model evaluation = good idea!

- The general paradigm: **train and test**
- Test data cannot be used in any way to create the model

[29] The only set you know until now is a training set. To use it for the evaluation of the model is a terrible idea for several reasons. First, the training instances have already been seen by the algorithm, so if we used them again for evaluation, it could simply be regarded as cheating. Moreover, the new data on which the classifier will be finally applied would probably not be the same as training data. The models have tendencies to fit the learning data too precisely, which is formally called overfitting. We will come back to this topic in the next lecture, but you should already understand that if the model performs very well on the training data, the chances that it will be equally good on other data are low. Finally, some classifiers, such as 1-NN, would always perform perfectly on the learning data. So using the training set for testing purposes is not the best idea. A much better one would be to have an independent testing set for conducting model evaluation. This set should not be used in any way to create the model. But where to take it from?

Test And Train - Hold-Out



- Basic technique: **hold-out** (use **two independent data sets**)
- Separation should not be ”convenience”
- In case of large data, randomly split into training and test sets (2/3 for training vs. 1/3 for test; 80% for training vs. 20% for test)
- Better – “***stratified***” random – division preserves relative proportion of classes in both training and test data
- In case of medium data, **repeated hold-out** (in each iteration, a certain proportion is randomly selected for training; admits overlap; error rates averaged over all iterations)

[30] Let me remind you that we have some set of examples for which the classification is known. Previously, we treated it as a learning set. Now, the idea is to hold out some part of it and employ it as a testing set. The default proportions between training and test are 2 to 1 or 4 to 1, and usually, the division is not fully random. In turn, it is made in a way, which preserves the relative proportion of classes in both training and test data. If your set is pretty large, you can just divide it once. If its size is medium, you can apply a repeated hold-out technique. This means that you repeat the division many times; in each iteration, different objects can be used as training or test data, and you just average the results over multiple runs. This technique admits overlapping between different iterations, which means that the same object can be used to learn or test the classifier depending on the random division made in different runs.

Classification Accuracy

Test set	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10		*acc*
Actual class	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_3	C_3	C_3		-
Predicted class (model M1)	C_1	C_2	C_1	C_2	C_2	C_3	C_3	C_2	C_3	C_3		6/10
Predicted class (model M2)	C_2	C_1	C_1	C_2	C_2	C_2	C_2	C_1	C_3	C_3		8/10
		loss=0					loss=1					

loss/cost function is a function that maps an event or values of one or more variables onto a real number ⇨ **0-1 loss function**

- N_t = number of testing examples
- N_c = number of correctly classified testing examples

$N_t = 10$
$N_c = 6$ for M1;
$N_c = 8$ for M2

classification accuracy $acc = \frac{N_c}{N_t}$

mislassification error $err = \frac{N_t - N_c}{N_t} = 1 - acc$

[31] The question is how to quantify such a performance of the model. In what follows, I will discuss a few measures which are used in the context of classification. The basic measure is called classification accuracy. It quantifies the proportion of objects for which the model correctly predicted their classification. It derives from the analysis of the so-called 0-1 loss. The objects for which the classification is predicted correctly generate no loss, whereas those for which a difference is observed imply a loss of one. Overall, this measure can be used to compare different models. You can see that in the slide, model M1 performs slightly better than M2. A counterpart of the classification accuracy, which is focused on badly classified alternatives, is called misclassification error. It is defined as 1 minus accuracy.

Confusion Matrix

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
Actual class	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_3	C_3	C_3
Predicted class (model M1)	C_1	C_2	C_1	C_2	C_2	C_3	C_3	C_2	C_3	C_3

confusion matrix		predicted class		
		C_1	C_2	C_3
original (actual) class	C_1	2	1	0
	C_2	0	2	2
	C_3	0	1	2

- N_t = number of testing examples
- N_c = number of correctly classified testing examples

classification accuracy

$$acc = \frac{N_c}{N_t} = \frac{6}{10}$$

$N_{i,j}$ = number of testing examples in class C_i that were labelled by the classifier as class C_j

$$acc = \frac{\sum_i N_{ii}}{\sum_i \sum_j N_{i,j}} = \frac{6}{10}$$

[32] The original and predicted classes can be compared using the so-called confusion matrix. In the rows of such a matrix, you have the original classes and in the columns the predicted ones. The matrix values indicate the number of testing examples in class from the row that were labeled by the classifier as a class from the column. For example, for our model, 2 objects from class C1 were correctly predicted as belonging to this class, whereas one object from C1 was assigned by the model to C2. Such a matrix is more detailed than classification accuracy or misclassification error because you can identify objects from which class are the most troublesome. Clearly, based on this matrix, you can compute the classification accuracy by referring to the numbers on the main diagonal, whereas the misclassification error is built upon the numbers outside the main diagonal.

Confusion Matrix - Cost Sensitive Analysis

- **cost sensitive analysis**
- costs ($cost_{i,j}$) assigned to different types of errors (may be enequal)
- sentiment analysis: favourable, unfavour., neutral

$$cost = \sum_i \sum_j cost_{i,j} \cdot N_{i,j}$$

could be different than 1

confusion matrix		predicted class		
		C_1	C_2	C_3
original (actual) class	C_1	2	1	0
	C_2	0	2	2
	C_3	0	1	2

cost matrix (defualt)		predicted class		
		C_1	C_2	C_3
original (actual) class	C_1	0	1	1
	C_2	1	0	1
	C_3	1	1	0

no cost related to the correct classification

$N_{i,j}$ = number of testing examples in class C_i that were labeled by the classifier as class C_j

[33] Such a matrix can be further used to perform the so-called cost-sensitive analysis. For this, you need to define a cost matrix, which defines the cost of a mistake for a particular pair of classes. Normally, these errors have equal costs, and making no mistake generates no cost. Think, however, of sentiment analysis with classes favorable, neutral, and unfavorable. A mistake between favorable and unfavorable should have a greater cost than between favorable and neutral. Such a comprehensive cost function is just a sum of products of confusion made and respective costs. This is useful when comparing multiple models, as you may favor the one with the least cost.

Recall, Precision, and F-measure

confusion matrix		predicted class C_1	C_2	C_3
original (actual) class	C_1	2	1	0
	C_2	0	2	2
	C_3	0	1	2

Recall for class C_i

Fraction of testing examples in class C_i classified correctly

$$recall_i = R_i = \frac{N_{ii}}{\sum_j N_{i,j}}$$

$recall_1 = 2/3$
$recall_2 = 2/4$
$recall_3 = 2/3$

- fraction of results which are relevant (*imagine looking for lectures before the exam from Intro to AI*)

Precision for class C_i

Fraction of testing examples assigned to class C_i that are actually from/about C_i

$$precision_i = P_i = \frac{N_{ii}}{\sum_i N_{i,j}}$$

$precision_1 = 2/2$
$precision_2 = 2/4$
$precision_3 = 2/4$

- fraction of the relevant results which are correctly classified (*consider your spam folder*)

$N_{i,j}$ = number of testing examples in class C_i that were labelled by the classifier as class C_j

Measure F

= weighted harmonic average

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

[34] The measures discussed so far were comprehensive and somehow accounted for all classes at once. Now, I would like to discuss a few class-specific measures. They are called recall, precision, and F-measure. For class Ci recall indicates a fraction of testing examples from this class, which were classified correctly. To compute it, you divide a number on the main diagonal by the sum of numbers from the row. For example, for C1 it is 2/3. In turn, for class Ci, precision is a fraction of testing examples assigned to class Ci that are actually from this class. So in the denominator, you refer to the sum of numbers from the column. In this perspective, precision for C1 is perfect. Note that these measures are important in different contexts. Imagine you are preparing for the exam from this course, and you are looking for the pdf with lectures. If you missed one, you would be unfortunate, because this may decide upon your failure. But if your searcher finds all of them and some more irrelevant, you would be fine because, in such a context, recall is essential. On the contrary, think of your spam folder. All messages classified as spam should truly be spam, which means that in this context, you optimize precision, and you are fine if some spam messages are not identified.

Confusion Matrix for Binary Classification

confusion matrix		predicted class: *positive*	predicted class: *negative*
actual class	***positive***	TP = true positive	FN = false negative
	negative	FP = false positive	TN = true negative

$$\textit{recall} = R = \frac{TP}{TP + FN}$$

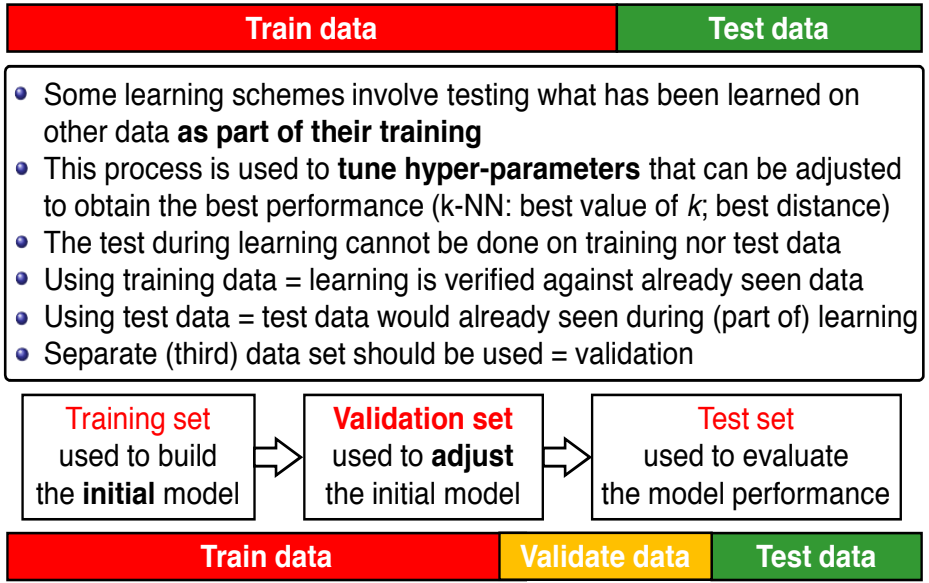
$$\textit{precision} = P = \frac{TP}{TP + FP}$$

* positive = no spam, negative = spam; positive = relevant, negative = non-relevant

- **TP** = outcome where the model correctly predicts the positive class
- **FP** = outcome where the model incorrectly predicts the positive class
- **TN** = outcome where the model correctly predicts the negative class
- **FN** = outcome where the model incorrectly predicts the negative class

[35] When a classification problem involves only two classes, which are generally called positive and negative, the confusion matrix cells have some special names. An example of the two-class problems refers to the identification of spam or docs which are relevant for a given query. However, also if there are more classes, you may just focus on a single selected class and consider the remaining ones jointly as negative. Coming to those special names, for example, objects for which the model correctly predicts the positive class are called true positives, whereas objects for which the model incorrectly predicts such a positive class are called false positives. This notation and concepts are commonly used in the machine learning community, so it is good to be familiar with them. Clearly, referring to those special names, you can define recall and precision in a standard way.

Validation – Testing as Part of the Training



[36] So far, we have talked about training and test data. It is interesting to note that some learning schemes involve testing what has been learned on other data as part of their training. For example, consider some hyper-parameters in the k-NN algorithm. You need to select a value of k; you need to choose a distance metric or a voting procedure. You cannot verify different possibilities on the training set, because you would verify learning against already seen data. It would also be stupid to use test data for this purpose, as this would imply that test data was already seen during learning. The correct solution consists of introducing a third set, called a validation set, that can be used to adjust the initial model. When the data set is large, you just divide your set of objects for which the class labels are known into three independent parts. You learn different models with various hyper-parameters on the training data, and validate their performance on the validation set. The model attaining the best results is verified against test data.

k-fold Cross-Validation

4-fold cross validation

Data set	Fold 1	Fold 2	Fold 3	Fold 4
1	**Test**	Train	Train	Train
2	Train	**Test**	Train	Train
3	Train	Train	**Test**	Train
4	Train	Train	Train	**Test**

** often used: 2-fold, 10-fold, leave one out*

_k_-fold cross validation

- Randomly divide data into *k* folds
- Use *k*-1 fold as training data and 1 fold as validation / test data
- Repeat *k* times
- Average results to yield an overall error estimate

- Extensive experiments have show that ***10-fold cross-validation* is the best choice** to get an accurate error estimate
- ***leave one out*** for very small size data (each fold composed of a single example; train on all but one example; repeat until tested on all examples)

[37] The question is what to do in case we cannot afford extensive training and validation sets. The answer is given by k-fold cross-validation. In this procedure, you divide your data into k folds, use k-1 as training data and one as validation set. This procedure is repeated k times so that each subset plays the role of a test set. The results are averaged over different runs. In the slide, you have an example of 4-fold cross-validation, but in practice, the most recommended division is into 10 folds. Many experiments are proving that this choice leads to very accurate estimates of a classification error. In case the data set if very small, the recommended technique is called leave-one-out. Here, each fold is composed of a single object, and you train the model on all but one example. The entire procedure is repeated until each example is used as the test set. The framework of k-fold cross-validation can be used in the context of using learning and validation sets, but also it is just commonly applied when only training and test sets are used, and you wish to avoid overlap between test sets used in various runs.

Summary (1)

Given the following documents **D1-D3** and a set of terms **T**:

- **D1** = {artificial intelligence artificial intelligence}
- **D2** = {artificial artificial introduction}
- **D3** = {artificial intelligence introduction artificial}
- **T** = {t_1=artificial, t_2=intelligence, t_3=introduction}

I) Represent D1-D3 using the binary setting, BOW, and TF

II) What is the Jaccard coefficient for D1 and D2 or D1 and D3?

III) What is the cosine similarity for D1 and D2 when using BOW or for D1 and D3 when using TF?

	t_1	t_2	t_3
D1			
D2			
D3			

Jac(D1,D2)=
Jac(D1,D3)=

cos(D1,D2)=
cos(D1,D3)=

Summary (2)

Given the following similarities of Y with D1-D10 and classes A, B, and C:

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10		**Y**
Class	A	C	B	C	B	A	A	C	A	B		**?**
Similarity with X	0.9	0.8	0.6	0.5	0.5	0.4	0.4	0.2	0.1	0.0		**1.0**

use 4-NN to provide a classification for Y :

I) … using either majority rule voting or weighted voting?

II) Would the results be different in case of using either k=3 or k=5?

III) What should be the minimal k so that A is recommended?

Summary (3)

Given the following collection of documents and assuming they are represented with the binary vector of terms, use the Naive Bayes classifier to classify D6:

	Doc	Content	Class
training	D1	T1 T2	A
training	D2	T3 T3	A
training	D3	T1 T3	A
training	D4	T1 T2 T2	B
training	D5	T1	B
test	**D6**	T1 T3 T3	?

I) What is the binary represenation of D5?

II) What is P(A)?

III) What is P(T3=1|A)? If 0, assume 0.1 as a simple smoothing technique.

IV) What class would be recommended for Y and why (compare P(A|D6) and P(B|D6))?

Doc	T1	T2	T3	Class
D1	1	1	0	A
D2	0	0	1	A
D3	1	0	1	A
D4	1	1	0	B
D5	?	?	?	B
D6	1	0	1	?

P(A) = ?

P(T1=1|A) = 2/3

P(T2=0|A) = 2/3

P(T3=1|A) = ?

P(A|D6) ≈ ?

P(B) = 2/5

P(T1=1|B) = 1

P(T2=0|B) = 1/2

P(T3=1|B) = 0 ≈ **0.1**

P(B|D6) ≈ 0.02

Summary (4)

Given the following collection of documents and assuming they are represented with the number of terms T1-T3, use the Naive Bayes classifier to classify D6:

	Doc	Content	Class
training	D1	T1 T2	A
training	D2	T3 T3	A
training	D3	T1 T3	A
training	D4	T1 T2 T2	B
training	D5	T1	B
test	**D6**	T1 T3 T3	?

I) What is the size of the dictionary?

II) What is P(B)?

III) What is P(T3|B) while using add-one smooting technique?

IV) What class would be recommended for Y and why (compare P(A|D6) and P(B|D6))?

P(A) = 3/5

P(T1|A) = (2+1)/(6+3) = 3/9

P(T3|A) = (3+1)/(6+3) = 4/9

$P(A|D6) \approx 3/5 \cdot 3/9 \cdot (4/9)^2 = 0.039$

P(B) = ?

P(T1|B) = (2+1)/(4+3) = 3/7

P(T3|B) = ?

P(B|D6) ≈ ?

Summary (5)

Given the following confusion matrix for the classification problem involving four classes C_1 – C_4 compute and 100 documents:

confusion matrix		predicted class			
		C_1	C_2	C_3	C_4
original (actual) class	C_1	20	0	0	0
	C_2	5	15	0	5
	C_3	0	10	15	0
	C_4	5	5	0	20

VI) Misclassification cost using the cost matrix to the right

I) Classification accuracy

II) Misclassification error

III) Recall for C_1 and C_4

IV) Precision for C_2 and C_3

V) Can you immediately say which class has the highest recall / precision?

cost	C_1	C_2	C_3	C_4
C_1	0	1	1	2
C_2	1	0	3	3
C_3	1	3	0	1
C_4	2	5	1	0