Artificial Intelligence - Knowledge Representation and Planning - Assignment 2

Lorenzo Soligo - 875566. Project done with Alessio Ragazzo and Gianmarco Callegher

Academic Year 2018-2019

1 Requirements

Write a spam filter using discrimitative and generative classifiers. Use the Spambase dataset which already represents spam/ham messages through a bag-of-words representations through a dictionary of 48 highly discriminative words and 6 characters. The first 54 features correspond to word/symbols frequencies; ignore features 55-57; feature 58 is the class label (1 spam/0 ham).

1. Perform SVM classification using linear, polynomial of degree 2, and RBF kernels over the TF/IDF representation. Can you transform the kernels to make use of angular information only (i.e., no length)? Are they still positive definite kernels?
2. Classify the same data also through a Naïve Bayes classifier for continuous inputs, modelling each feature with a Gaussian distribution, resulting in the following model:

$$p(y = k) = \alpha_k$$

$$p(\boldsymbol{x}|y = k) = \prod_{i=1}^{D} \left[(2\pi\sigma_{ki}^2)^{-1/2} exp\left\{ -\frac{1}{2\sigma_{ki}^2}(x_i - \mu_{ki})^2 \right\} \right]$$

where α_k is the frequency of class k, and μ_{ki}, σ_{ki}^2 are the means and variances of feature i given that the data is in class k.

Provide the code, the models on the training set, and the respective performances in 10 way cross validation.

Explain the differences between the two models.

2 The Spambase dataset

2.1 Introduction

The spambase dataset is a collection of spam emails which have been marked as spam from postmasters and individuals. The collection of non-spam emails came from filed work and personal emails.

2.2 Fields

Here are the definitions of the attributes:

- 48 continuous real [0,100] attributes of type `word_freq_WORD = percentage of words in the email that match WORD`, i.e. `100 * (number of times the WORD appears in the email)/ total number of words in email`. A "word" is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
- 6 continuous real [0,100] attributes of type `char_freq_CHAR = percentage of characters in the email that match CHAR`, i.e. `100 * (number of CHAR occurences)/ total characters in email`
- 3 continuous values regarding the presence of capital letters, ignored in our analysis
- 1 nominal {0,1} class attribute of type `spam = denotes whether the email was considered spam (1)or not (0)`, i.e. unsolicited commercial email.
- this is our target Y, i.e. the label we want to predict. Given an email, is it spam or not?

2.3 Cleaning the dataset

First of all, we should notice that the frequencies here are percentages (i.e. max=100), not standard proportions (i.e. max=1.00). We will need to take care of this property when dealing with TF/IDF.

Also, the dataset contains some lines which are only made up of 0's in the first 54 columns, which are the only ones we consider (except for the target spam/ham). Therefore, we remove those lines because they don't give any additional information in that case, and only add noise to the dataset.

Finally, since we have no idea about the way the dataset was coded, we will always shuffle it, in order to try and remove possible biases introduced by the creators.

3 Supervised Learning

In supervised learning, an agent is given `<input, output>` tuples and its objective is to learn a function which maps an input to the relative output.

Formally, given a training set of N input-output pairs $(x_1, y_1), ..., (x_N, y_N)$, the objective of supervised learning is to learn a function h which approximates the unknown ("true") function $y = f(x)$ that maps x's to y's. Sometimes, f might be stochastic, therefore we might need to learn $\boldsymbol{P}(Y|x)$.

Usually, we want h to *generalize* well when we give it some input x that it has never seen before; in order to check whether h generalizes well, we can split the available data into a training set and a test set. We might as well say that we want the "best" possible hypothesis h^* given the data: as a matter of fact, we consider an hypothesis which is consistent (i.e. when applied, classifies correctly) with a sufficiently large set of training examples to be unlikely to be seriously wrong; it must be *Probably Approximately Correct* (PAC).

The main problems faced by supervised learning are the following two: **classification** and **regression**.

3.1 Classification

A **classification problem** is a problem that consists in finding a function ϕ to assign a label to a set of vectors which represent the input data (*feature vectors*).

Classification therefore simply consists in applying a label to an input: for example, our spam filter needs to classify emails to tell us whether they are spam (1) or not (0). Labelling an input implies enforcing its belonging to a determined *class* (in our case: *spam* or *ham*). Classes could be more than 2, but they are always a limited, discrete, "small" number.

3.2 Regression

Regression, on the other hand, usually produces a continuous output in the form of a number. This means that in classification the functions f and h have a discrete co-domain, while in regression they have a continuous one.

Intuitively, classification assigns a label while regression is used to predict continuous quantities.

4 Classifiers

In this assignment, we need to classify emails, therefore we will use two algorithms (Naïve Bayes and SVM) in order to perform classification. We might as well say that we will use a **Naïve Bayes Classifier** and a **Support Vector Classifier**.

4.1 Definition of classifier

Given a population whose members each belong to one of a number of different sets or classes, a classifier (or classification rule) is a procedure by which the elements of the population set are each predicted to belong to one of the classes. A special kind of classification rule is binary classification, for problems in which there are only two classes. The latter is the case we are dealing with in this assignment.

4.2 Some basic background

Before going on, let's define some fundamental background knowledge:

- $p(x)$: probability of observing x as an input, i.e. probability of picking an element x from the set of input data $\mathcal{X}$
- $p(y)$: probability of the class (label) y
- $p(x, y)$: *joint* probability of x and y
- $p(y|x)$: *conditional* probability of y *given* x, i.e. probability that an observed element x belongs to a class y
- $p(x|y)$: *prior* probability, i.e. probability of observing the particular value x when we know that the element belongs to class y.

4.3 Discriminative and Generative algorithms

Supervised learning algorithms can be of two types: *discriminative* and *generative*. Both these models try to predict the probability $p(y|x)$; the difference is in *how* they do it.

Generative models first learn the *joint* probability distribution $p(x, y)$ from the prior probability $p(x|y)$ and the class probability $p(y)$, and then apply Bayes' theorem in order to get to the conditional probability distribution $p(y|x)$. This can be useful to produce possible pairs that might be drawn from the actual distribution. For example, Bayesian Networks and Naïve Bayes are two generative classifiers. On the other hand, discriminative models "only" learn the *conditional* probability distribution $p(y|x)$ directly. Classical examples of discriminative classifiers are Support Vector Machines and Logistic Regression.

5 Understanding performance

5.1 TP, TN, FP, FN

- **True Positive**: correctly predicted positive values which means that the value of actual class is "yes" and the value of predicted class is also "yes"
- **True Negative**: correctly predicted positive values which means that the value of actual class is "no" and the value of predicted class is also "no"
- **False Positive**: actual class is "no" and predicted class is "yes"
- **False Negative**: actual class is "yes" and predicted class is "no"

5.2 Performance measures

Accuracy: $\frac{TP+TN}{TP+FP+FN+TN}$. Ratio of correctly predicted observation to the total observations.

Precision: $\frac{TP}{TP+FP}$. Ratio of correctly predicted positive observations to the total predicted positive observations.

Recall: $\frac{TP}{TP+FN}$. Ratio of correctly predicted positive observations to the all observations in actual class - yes.

F1 score: $\frac{2*(Recall*Precision)}{Recall+Precision}$. Weighted average of Precision and Recall.

6 TFIDF

In information retrieval, tf–idf or TFIDF, short for *term frequency–inverse document frequency*, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

6.1 Term Frequency

In the simplest formulation, we define the *term frequency* $tf(t, d)$ as the number of times that the term t occurs in the document d. Notice that we use the notation $|d|$ to identify the number of words in document d.

$$tf(t, d) = \frac{|\{i \in d : i = t\}|}{|d|}$$

6.2 Inverse Document Frequency

The inverse document frequency is a measure of how much information the word provides, i.e., if it's common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word.

With D = set of documents, and $N = |D|$, we have

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

6.3 Formula for TFIDF

Now that we know what TF and IDF are, we can define TFIDF as

$$TFIDF(t, d, D) = tf(t, d) \cdot idf(t, D)$$

7 Support Vector Machines

7.1 Introduction

Support Vector Machines (SVMs) are one of the most popular approaches for supervised learning, especially when the developer has no specific knowledge about the considered domain. SVMs construct a maximum margin separator; in other words, they construct a decision boundary with the largest possible distance to example points. This helps SVMs generalize well.

The key idea behind SVMs is that some examples are "more important" than others, and paying special attention to them can lead towards better generalization. Therefore, even though (in theory) a SVM could be considered a nonparametric model (it retains the training data), it usually only needs to keep track of only few elements of the training set, therefore moving towards a "more parametric" approach. Also, SVMs provide a good amount of flexibility and resistance towards overfitting.

Support vectors are the points in space that are closer ($d = 0$) to the separator: since they "hold up" the separating hyperplane, they are said to "support" it.

SVMs create a linear separating hyperplane, but they have the ability to embed the data into a higher-dimensional space, using the so-called *kernel trick*. Often, data that are not linearly separable in the original input space are easily separable in the higher-dimensional space. The high-dimensional linear separator is actually nonlinear in the original space. This means the hypothesis space is greatly expanded over methods that use strictly linear representations.

Another great advantage of SVMs is that solving it consists in solving a quadratic optimization problem, which is "easy".

7.2 Definition and separating hyperplane

A SVM classifier can be defined as a function

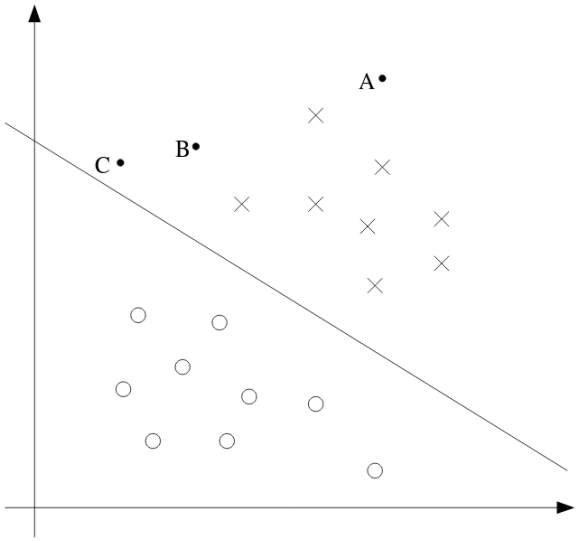
$$h_{w,b}(x) = g(w^T x + b) = \begin{cases} 1 & \text{if } w^T x + b \geq 0 \text{ (spam)} \\ -1 & \text{otherwise (ham)} \end{cases}$$

Here I used -1 instead of 0 be adapt to the standard SVM notation.

We say that w is the weight vector, x is an input vector and b is the bias (which, in the simple case of a line, can also be seen as intercept).

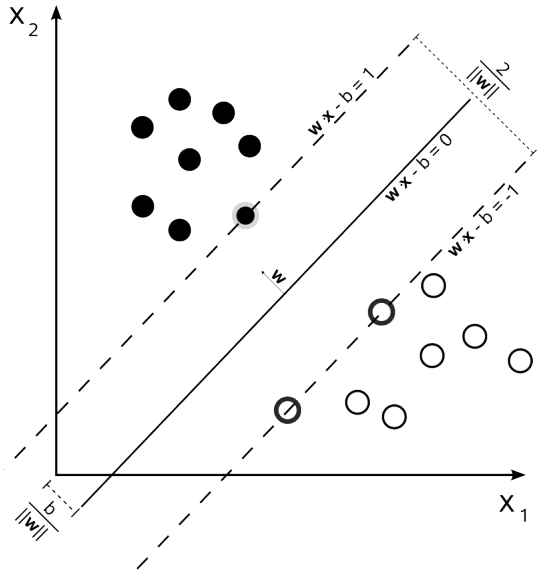
Here $w^T x + b = 0$ is the so-called *separating hyperplane* (decision boundary) between the two classes. g is a function that simply checks the sign of $w^T x + b$ and classifies the vector x according. Intuitively, the more the value $w^T x + b$ is distant from 0, the more confident we will be about the prediction we made.

Of course, many different x's (hence, many different hyperplanes) could be chosen, thus we need to pick the best one. The one we want is, intuitively, the one that provides the **larger possible margin** between the two classes. This concept of choice intuitively explains that SVMs are algorithms to solve constrained optimization problems.



Frist of all, let's appreciate that for each linear separator we choose, there will always be some points closer to it. These points in particular deserve special attention, because they might be the support vectors of our separator or, anyway, influence the equation of the separator. Notice also that if the support vectors are removed from the dataset, the decision boundary might change!

Intuitively, the support vectors are the samples that are most difficult to classify. They are really close to the decision boundary, and therefore it is fundamental to have a close look at them.



7.3 Margins and Constrained Optimization formulation

The SVM algorithm can be seen as a constrained optimization problem: as said before, we need to maximize the distance between the separating hyperplane and the support vectors. To do so, we need the notions of *geometric* and *functional* margin. We can think about the functional margin as a testing function that tells us whether a particular point (vector) is properly classified. The functional margin can be defined as $\hat{\gamma}_i = y_i(w^T x_i + b)$. Notice that if we scale the result, the sign remains the same, and this is the reason geometric margin exists: it tells us the distance in terms of units of $|w|$. Recall that this is not an Eucledian distance because the geometric margin (and also the functional one) can have a negative sign, in the case of misclassification. The geometric margin is simply the functional margin scaled, i.e. divided by the norm-2 of vector w, i.e. $\gamma_i = \frac{\hat{\gamma}_i}{|w|} = \frac{y_i(w^T x_i + b)}{|w|}$.

Given a training set $S = \{(x_i, y_i); i = 1, ..., m\}$ we define the geometric margin of (w, b) with respect to S to be the smallest of the geometric margins on the individual training examples: $\gamma = \min_{i=1,...,m} \gamma_i$.

7.3.1 Linearly separable training set

Assuming the training set is **linearly separable**, we can then select the "best" margin as

$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq \gamma, \quad i = 1, ..., m \\ & ||w|| = 1 \end{aligned}$$

The first constraint enforces a correct classification of all the samples, while the second one is used to set the functional margin to be equal to the geometric one.

We can also write the problem as

$$\begin{aligned} \max_{\gamma,w,b} \quad & \frac{\gamma}{|w|} \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq \gamma, \quad i = 1, ..., m \end{aligned}$$

Finally, if we enforce $\hat{\gamma} = 1$, we can rewrite the problem as a quadratic optimization one

$$\begin{aligned} \max_{\gamma,w,b} \quad & \frac{1}{2}|w|^2 \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq \gamma, \quad i = 1, ..., m \end{aligned}$$

The problem is convex and the constraints are linear, therefore a global optimum exists and it is what we are looking for.

Also, the problem can be formulated in the Lagrangian form, with the advantage that the support vectors will be the ones corresponding to λ_i's greater than 0. The other ones will either be 0 or really close. This formulation corresponds to:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & \sum_{i=1}^{m} \lambda_i y_i = 0, \quad i = 1, ..., m \\ & \lambda_i \geq 0, \quad i = 1, ..., m \end{aligned}$$

7.4 The Kernel trick

In the case the dataset is **not linearly separable**, we have mentioned that we can use the so-called **kernel trick**. A kernel is a function used in the SVM algorithm that maps the data to a higher-dimensional space. This might help because we might find a linear separator in a higher-dimensional space that is not linear in the original one.

In other words, a kernel function is a way of computing the dot product between two vectors x_i and x_j in some high-dimensional feature space. Formally a *valid* kernel is a function $K(x_i, x_j) = \phi(x_i)^T \phi(x_j) \forall x_i, x_j$. A great property about kernel functions is that we can replace the inner product in the SVM with an application of the kernel function to move to a different feature space. Being this a feature mapping, we usually define it using the greek letter ϕ, as above. We don't need to know ϕ: we just need it to verify the aforementioned property.

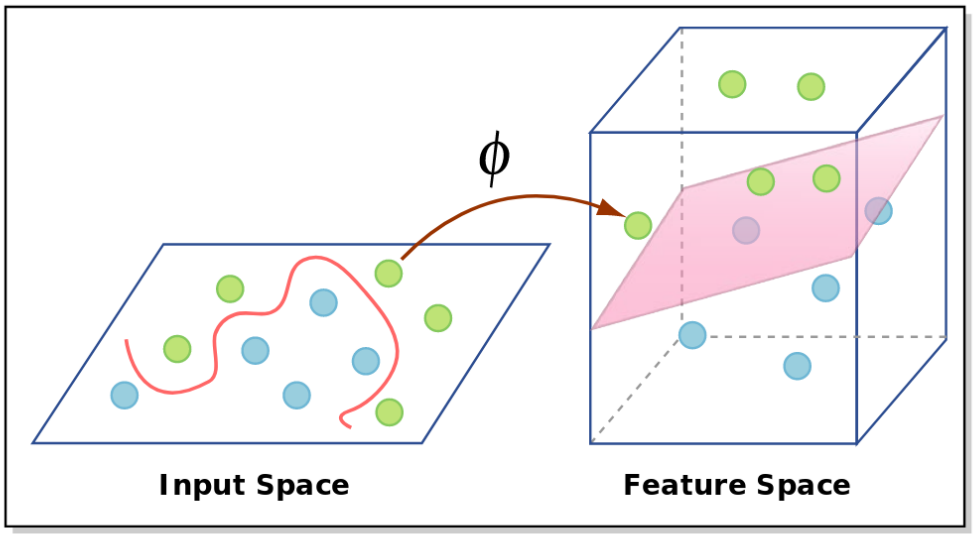
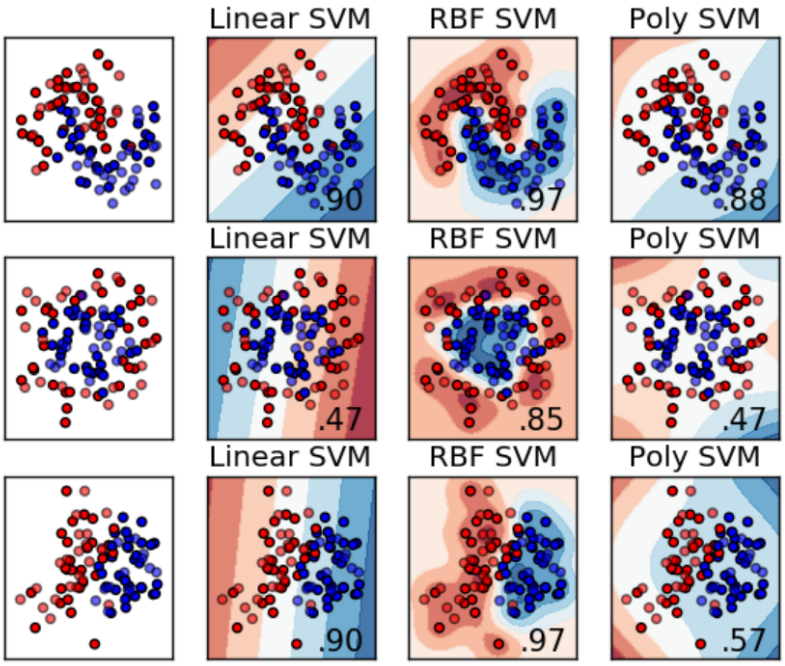


Image by MIT OpenCourseWare.

7.4.1 Our three kernels

We will use three kernels in this assignment:

- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial of degree 2: $K(x_i, x_j) = (1 + x_i^T x_j)^2$ (others use a generic $\gamma > 0$ instead of the 1)
- Radial Basis Function (RBF): $K(x_i, x_j) = e^{-\frac{|x_i - x_j|^2}{2\sigma^2}}$



7.4.2 Positive definiteness

Another important property of kernels is positive definiteness. We say that a symmetric function $K : \mathbb{R}^n \times \mathbb{R}^n$ is a positive definite kernel if $\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \quad \forall c_i, c_j \in \mathbb{R}$.

7.4.3 Angular kernels

The three kernels we defined are affected by the length of vectors, i.e. two vectors with identical direction (angle) but different length are not considered as similar. We are curious to try and see whether considering only angular information would provide an improvement in the classification process. This might be the case because it might be that the direction of the vector tells us more when considered alone than when considered together with its length. Of course, in order to only consider the angle between vectors, we need to normalize them, i.e. to consider the versors given by $\frac{w}{|w|}$.

7.4.3.1 Positive definiteness of the Linear Kernel Let's consider the function $\phi : \mathbb{R}^n \to \mathbb{R}^m$, with $\phi(x) = \frac{x}{|x|}$.

Then we can define our linear kernel K as $K(x_i, x_j) = \phi(x_i)^T \phi(x_j) = \frac{x_i}{|x_i|}\frac{x_j}{|x_j|} = \cos(\widehat{x_i x_j})$. Now we need to prove that it is still positive definite.

From Mercer's theorem we know that a kernel is positive definite if the following holds:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \quad \forall c_i, c_j \in \mathbb{R} \quad \text{with} \quad \sum_{i=1}^{m} c_i = 0$$

Then we can compute $\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(x_i, x_j)$.

In the following, we have n = number of vectors (observations), l = number of features of a vector, and $x_i^{(k)}$ identifies the k-th feature of vector x_i.

$$
\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(x_i, x_j) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j (\phi(x_i)^T \phi(x_j)) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \sum_{k=1}^{l} \frac{x_i^{(k)}}{|x_i^{(k)}|} \frac{x_j^{(k)}}{|x_j^{(k)}|} \\
&= \sum_{k=1}^{l} \left[\sum_{i=1}^{n} c_i \frac{x_i^{(k)}}{|x_i^{(k)}|} \cdot \sum_{j=1}^{n} c_j \frac{x_j^{(k)}}{|x_j^{(k)}|} \right] \\
&= \sum_{k=1}^{l} \left(\sum_{i=1}^{n} c_i \frac{x_i^{(k)}}{|x_i^{(k)}|} \right)^2 \geq 0
\end{aligned}
$$

7.4.3.2 Positive definiteness of the Polynomial Kernel The polynomial kernel of degree 2 is the function $K(x_i, x_j) = (1 + x_i^T x_j)^2$. Also here we need Mercer's theorem to be verified. This can be easily verified exploiting our proof for the linear kernel: as a matter of fact, if we call K_L the linear kernel and K_{P2} the poly-2 kernel, we have the reationship $K_{P2}(x_i, x_j) = (1 + K_L(x_i, x_j))^2$.

Notice, however, that this could also be proved using a simpler trick. In fact, we know that the polynomial kernel is positive definite, therefore we might simply consider a function $\xi(x) = \phi(\frac{x}{|x|})$ and consider the kernel

$$
K'(x_i, x_j) = \xi(x_i)^T \xi(x_j) = \phi\left(\frac{x_i}{|x_i|}\right)^T \phi\left(\frac{x_j}{|x_j|}\right) = K\left(\frac{x_i}{|x_i|}, \frac{x_j}{|x_j|}\right)
$$

Since we are using the original poly2 kernel with different parameters, we have proved that the "new" kernel is also positive definite.

7.4.3.3 Positive definiteness of the RBF kernel The proof here is trivial and is very similar to the last step of the previous one. We know that the RBF kernel is positive definite, therefore we can do the same trick as before.

Let's consider a function $\xi(x) = \phi(\frac{x}{|x|})$ and consider the kernel

$$
K'(x_i, x_j) = \xi(x_i)^T \xi(x_j) = \phi\left(\frac{x_i}{|x_i|}\right)^T \phi\left(\frac{x_j}{|x_j|}\right) = K\left(\frac{x_i}{|x_i|}, \frac{x_j}{|x_j|}\right)
$$

As before, since we reduced the new kernel to the original RBF kernel with different parameters, we have the proof that the "new" kernel is also positive definite.

8 Naïve Bayes

8.1 Intuition

Naïve Bayes is a probabilistic classifier inspired by Bayes' theorem; the underlying assumption is that the attributes are **conditionally independent**. Of course, this assumption is not obvious and doesn't always hold, therefore the algorithm is subject to huge swings in performance depending on whether it is satisfied or not. However, it sometimes performs decently even if the assumption is not satisfied.

Recall that we define two random variables X and Y as conditionally independent given Z iff $P(X = x|Y = y, Z = z) = P(X = x|Z = z) \ \ \forall x, y, z$. In the simpler case regarding two variables, we have that if X and Y are conditionally independent, then $P(X = x|Y = y) = P(X = x) \ \forall x, y$. From this we can also say that $P(X \cap Y) = P(X) * P(Y)$ iff X and Y are conditionally independent.

8.2 Learning

While SVMs learn $p(y|x)$, Naïve Bayes usually learns the prior probability $p(x|y)$ and the class probability $p(y)$, which when multiplied give the joint probability $p(x, y)$. As a matter of fact, Naïve Bayes applies Bayes' theorem obtaining the following results:

$$p(y|x) = \frac{p(x, y)}{p(x)} = \frac{p(x|y)p(y)}{\sum_{c \in \mathcal{C}} p(x|c)p(c)}$$

being $\mathcal{C}$ the set containing all the different classes/labels (in our case, *spam* and *ham*). When given an input x, Naïve Bayes labels it with the label y that maximizes $p(y|x)$. Formally, we have that the learnt function $h(x)$ labels an input using the following:

$$h(x) = \underset{y}{\operatorname{argmax}}\, p(y|x) = \underset{y}{\operatorname{argmax}}\, \frac{p(x|y)p(y)}{p(x)} = \underset{y}{\operatorname{argmax}}\, p(x|y)p(y)$$

we can avoid dividing by $p(x)$ because it does not depend on the class y.

8.3 Gaussian distribution

In this assignment we will assume that the probability distribution of x given a class y, i.e. $p(x|y)$, follows a Gaussian (normal) distribution. In other words, we define the conditional probability of x given y as $p(\boldsymbol{x}|y = k) = \prod_{i=1}^{D} \left[(2\pi\sigma_{ki}^2)^{-1/2} exp\left\{ -\frac{1}{2\sigma_{ki}^2}(x_i - \mu_{ki})^2 \right\} \right]$

Notice that this is an assumption, and this will need to be checked together with the conditional independence in order to understand whether and why Naïve Bayes works better or worse than SVM in our case.

8.4 Some known problems

A subtle issue with Naïve-Bayes is that if we have no occurrences of a class label and a certain attribute value together (e.g. class="nice", shape="sphere") then the frequency-based probability estimate will be zero. Given Naïve-Bayes' conditional independence assumption, when all the probabilities are multiplied you will get zero and this will affect the posterior probability estimate. This is called the *zero probability problem*. When the conditional probability is zero for a particular attribute, it fails to give a valid prediction. This problem happens when we

are drawing samples from a population and the drawn vectors are not fully representative of the population. Lagrange correction and other schemes have been proposed to avoid this undesirable situation.

In classification tasks we need a big dataset in order to make reliable estimations of the probability of each class. Using Naïve Bayes with a small dataset might lead to low precision and recall.

9 Implementation

9.1 Language and library

We decided to use Python and the Scikit-Learn library because they easily let us work with both SVMs and Naïve Bayes. In particular, since the assignment requires us to work with a Gaussian Naïve Bayes, we used `GaussianNB` from `sklearn.naive_bayes`; on the other hand, for SVMs, we used `SVC` from `sklearn.svm` since it lets us choose between different kernels. In both cases, cross validation is applied using Sklearn's built-in model selection tools.

9.2 Cross Validation

9.2.1 *k*-fold cross validation

In this assignment we use 10-fold cross validation. CV is commonly used when the goal of a project is prediction and the user wants to estimate how accurately a predictive model would perform in practice. k-fold cross validation is used in order to try and avoid selection bias and overfitting, and consists in partitioning a sample of data into k complementary subsets, performing the learning phase on $k-1$ subsets (*training set*) an validating the model on the remaining subset (*testing set*). In order to reduce valiability, this process is usually run multiple (k) times, and the validation results are combined over the rounds to have a better idea about the model's performance.

9.3 SVM

We put the three required SVMs (one per kernel) in a Python dictionary and ran a 10-fold cross validation. This can be achieved by creating the dictionary and then running the CV on each of the SVMs. Notice that we do the same for the normalized input, i.e. when considering angular information only.

```
results, results_normalized = {}, {}

classifiers = {
    'Linear': SVC(kernel='linear', gamma='auto'),
    'Poly2': SVC(kernel='poly', gamma='auto', degree=2),
    'RBF': SVC(kernel='rbf', gamma='auto')
}

for k, v in classifiers.items():
    results[k] = cross_val_score(v, x, y, cv=10, n_jobs=-1)
    results_normalized[k] = cross_val_score(v, x_normalized, y,
        cv=10, n_jobs=-1)
```

9.3.1 Results

Here are the results of the SVM classifier on 10-fold Cross Validation without using angular information:

Kernel	Minimum Accuracy	Mean Accuracy	Maximum Accuracy
Linear	0.5969	0.5997	0.6063
Poly2	0.5969	0.5970	0.5982
RBF	0.5969	0.5970	0.5982

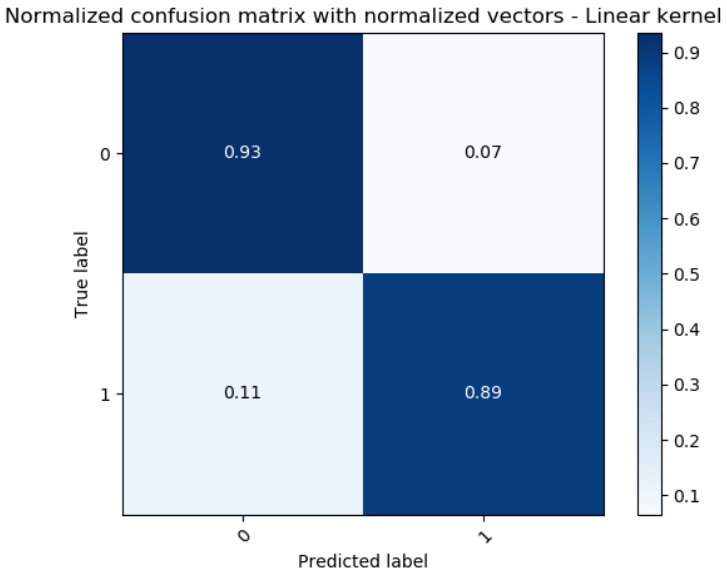
As we can see, the results are not that great: we correctly classify ~59% of the emails, which is not really satisfactory.

Here are the results of the SVM classifier on 10-fold Cross Validation using angular information:

Kernel	Minimum Accuracy	Mean Accuracy	Maximum Accuracy
Linear	0.9042	0.9258	0.9510
Poly2	0.5969	0.5970	0.5982
RBF	0.8953	0.9204	0.9488

As we can see, while the linear and RBF kernels get huge improvements (+30%), while the polynomial kernel works the same way as before. It would be very interesting to study the reason behind the absence of improvements in the polynomial kernel: a guess could be that we are "forcing" the data to be separated from a paraboloid while the data don't really follow this trend. However, I wonder whether in that case the coefficient of the highest-degree term could be lowered to move towards a separator that is almost linear.

Here is the confusion matrix related to the SVM (kernel) that helps visualizing the results:



9.4 Naïve Bayes

Luckily for us, `sklearn` provides a pre-built Gaussian Naïve Bayes. Here we don't need to specify anything but the target variable (y) and the set of feature vectors (x). After this, we perform a 10-fold CV.

```
gaussian_bayes = GaussianNB()
y_pred = gaussian_bayes.fit(x, y).predict(x)
r = cross_val_score(gaussian_bayes, x, y, cv=10, n_jobs=-1,
    scoring='accuracy')
pprint(classification_report(y, y_pred, output_dict=True))
```

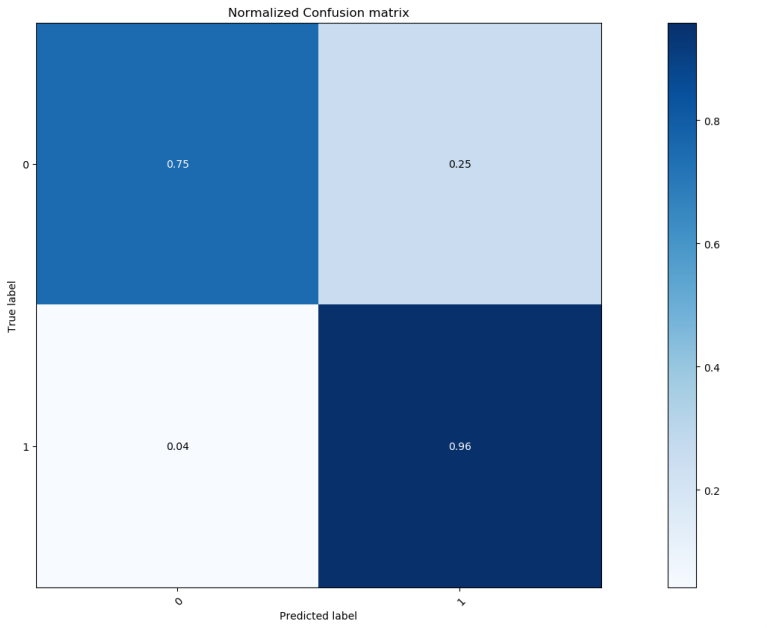
In particular, the classification report gives us more information than the simple accuracy: we also get the precision, recall and F1 score.

9.4.1 Results

- Number of mislabeled points out of a total 4487 points: 750
- Minimum accuracy: 0.7795
- Average accuracy: 0.8311
- Maximum accuracy: 0.8597

Class	Precision	Recall	F1 Score
0 (ham)	0.9639	0.7480	0.8424
1 (spam)	0.7197	0.9585	0.8221

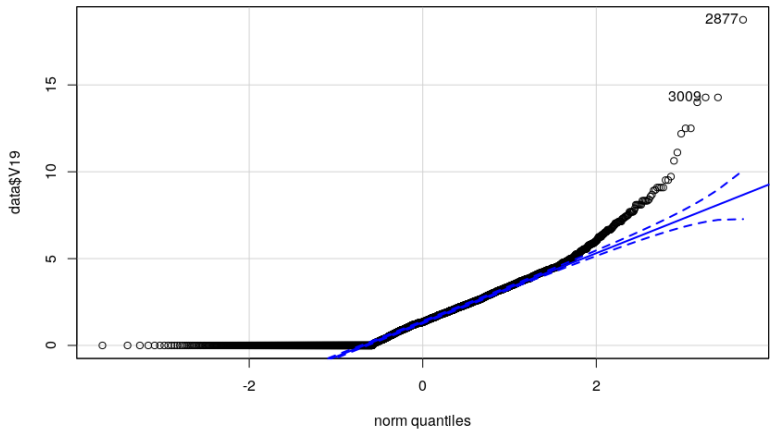
We can also see the results with a confusion matrix that helps visualizing them:

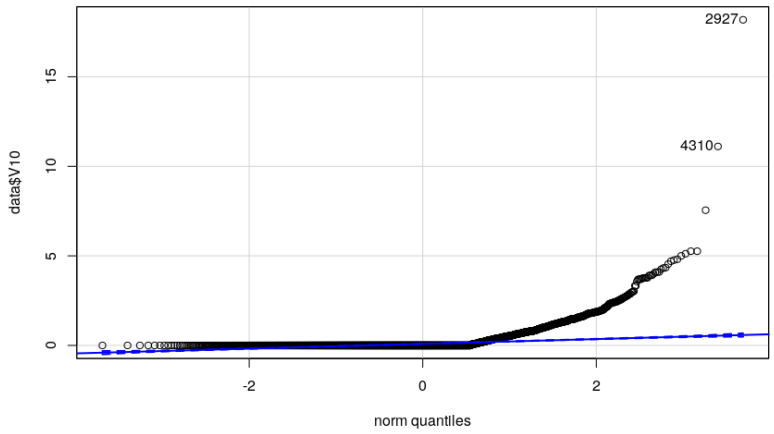


9.4.2 Data distribution

The results of Naïve Bayes are decent, but the SVM can do better. We might wonder why this is the case. The first reason that comes to mind is that the data might not be normally distributed, which is an assumption that we require to be true for Naïve Bayes to work well. There are many ways to check whether the data are normally distributed: for example, we might have a look at the QQplots of some features and see whether they might fit a Gaussian distribution. Also, there are tests like Henze-Zirkler that can tell us whether the distribution of the data resembles the one of a Multivariate Normal.

As we can see, the QQplots show that the features aren't normally distributed.





Also, running a Henze-Zirkler test at a 95% confidence level rejects the hypothesis of multivariate normality.

9.4.3 Conditional independence

Another assumption made by Naïve Bayes is that the data are conditionally independent. Also this assumption should be verified, and unfortunately it does not hold. First of all, if we look at the correlation matrix for (e.g.) the first 20 columns of the dataset, we notice that some of them are highly correlated while some others are not. This already feels like a situation in which some patterns are recurrent. If we try to run some tests on random variables of the dataset, we often get p-values that are basically 0 for both χ^2 tests and Pearson correlation tests, meaning that it would be almost impossible to have such a dataset if the data were actually conditionally independent. This resembles the intuitive idea that words like "free" and "money" are probably found together in many emails, especially spam ones.

In the case of a Pearson Correlation test *free* and *money*, we have a 0.095 correlation and a p-value of 10^{-10}; running a χ^2 test on the same words leads to a p-value smaller than $2.2 * 10^{-16}$. In other words, under the hypothesis that *free* and *money* are conditionally independent, it would be practically impossible to draw such a dataset.

10 Comparison

As we have seen, the SVM algorithm that exploits angular information seems to be the best one in terms of accuracy. The fact that angular information improves the perfomance so drastically is very interesting: it seems like what we are looking for in this analysis is the "direction" of vectors -intuitively, whether they "point towards" spam or ham- instead of their length. In fact, considering the length makes two vector with the same direction but different lengths different between each other, while considering angular information makes them similar. Another interesting consideration is that, in this case, the best-performing SVM kernel is the linear one, which means that in this case more complex kernels are not providing better performance: this reminds us that while complex models might perform better than simple ones, over-engineering is also to be avoided and we should always consider many different possibilities. Also in the case of this assignment the hypotheses behind Naïve Bayes are not satisfied, therefore it doesn't perform as well as the SVM with angular information. In fact, while generative classifiers can do more things than discriminative ones (for example, generate new samples and help identify outliers), they require stronger assumptions in order to do so. In this case, data are not normally distributed and conditional independence also seems not to hold, therefore a SVM with a big enough dataset outperforms Naïve Bayes indeed because its "naive" assumptions are not satisfied. We might as well say that while generative classifiers provide a wider range of possible use cases, they also require stronger assumptions, therefore it makes sense to use them either when the latter are satisfied or when we specifically need tools that discriminative classifiers do not provide at all. Lastly, it would also be interesting to consider the usage (with some adaptations) of *multinomial* Naïve Bayes instead of the Gaussian one: in fact, multinomial Naïve Bayes has often been used with good results in document classification, and it might behave well in our spam filter - still keeping in mind that the assumptions are not verified.

11 References

- Professor's slides
- Personal notes
- scikit-learn's documentation
- https://stackoverflow.com/questions/20058036/svm-what-is-a-functional-margin
- https://davidrosenberg.github.io/mlcourse/Labs/3-SVM-Notes_sol.pdf
- http://cs229.stanford.edu/notes/cs229-notes2.pdf
- http://cs229.stanford.edu/notes/cs229-notes3.pdf
- https://towardsdatascience.com/understanding-support-vector-machine-part-1-lagrange-multipliers-5c24a52ffc5e
- https://towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d
- https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
- https://stats.stackexchange.com/questions/149889/prove-that-a-kernel-is-conditionally-positive-definite
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- https://en.wikipedia.org/wiki/Cross-validation_(statistics)#k-fold_cross-validation
- https://en.wikipedia.org/wiki/Tf%E2%80%93idf
- https://gist.github.com/WittmannF/60680723ed8dd0cb993051a7448f7805
- http://www.cs.colostate.edu/~cs545/fall15/lib/exe/fetch.php?media=wiki:18_naive_bayes.pdf