Ca' Foscari University of Venice

ARTIFICIAL INTELLIGENCE: KNOWLEDGE REPRESENTATION AND PLANNING

[CM0472-1]

Assignment 2: SpamFilter

Report by:
Zonelli Mattia, 870038

Code by:
Zonelli Mattia, 870038
Garbin Eleonora, 869831

Contents

1 Introduction

The goal of the assignment is to write a spam filter using **discriminative** 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).

- 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?

- Classify the same data also through a **Naive Bayes** classifier for continuous inputs, modelling each feature with a Gaussian distribution, resulting in the following model:

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

  where a_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.

- Perform **k-NN** classification with k=5

Provide the code, the models on the training set, and the respective performances in 10-way cross validation.
Explain the differences between the three models.

2 Background

2.1 Classification problems

Classification is one of the most common problem in machine learning and statistics. A classification algorithm starts from a bunch of possibly different input observations, better called Feature vectors, and use them to learn a model in order to be able to assign labels to new observations. In our case, the objects are emails and labels are 0 and 1. We have label 0 for emails classified as ham and 1 for email that represents spam. Classifiers should take some emails already labelled, learn a model such that the label, that it is going to be assigned to new emails (not seen in the learning phase), is the most suitable.
Let's have a more formal view of these type of problems.
let $X = (X_1, ..., X_m) \in \mathcal{X}$ as the predictors of our problem and let $Y \in \mathcal{Y}$ the target variable which specify the class of the observations. The classification problem consists on learning a function $\phi : \mathcal{X} \to \mathcal{Y}$ from a training set. In our case, $\mathcal{Y} = \{0, 1\}$ such that:

$$\phi = g(f(x))$$

where, $f(x)$ is the discrimination function that should separate objects labelled with 0 from object labelled 1.

$$g(z) = \begin{cases} 1(spam), & \text{if } \ z \geq \varepsilon \\ 0(ham), & \text{if otherwise} \end{cases}$$

It is a function that returns the right label to assign to a new email x according to the result of function $z = f(x)$, and where ε is the decision threshold that is learned during the training phase.

2.2 Type of classifiers

Some of most common type of learning algorithm are supervised, unsupervised and reinforcement learning. Supervised learning is based on learning from example data. An algorithm try to model relationships and dependencies between the target output and the input features such that we can predict the output labels for new data based on those relationships which it learned from the training data sets.

Unsupervised learning algorithms are trained completely with unlabelled data. They should be able to teach us something after learning patterns from data. These algorithms are useful when human expert doesn't know what to look for in the data.
Reinforcement learning algorithm (called the agent) continuously learns from the environment in an iterative fashion. The agent uses observations gathered from its interaction with the environment to make actions in order to maximize its reward or minimize its risk.[1]
In our assignment, we talk about SVM, Naive Bayes and Nearest neighbor that are some Supervised learning algorithms used to classify data. Furthermore, classifier can be divided into two classes, Discriminative and Generative.
In general, a Discriminative model models the decision boundary between the classes, learning directly $p(y|x)$ or mapping directly from the space of inputs X to the labels Y. Instead, a Generative Model explicitly models the actual distribution of each class, estimating parameters of $p(x|y)$ and $p(y)$ directly from training data and using the Bayes rule to compute $p(y|x)$.
Where,

- $p(x)$ is the probability of observing x from the domain space X.
- $p(y)$ is the probability distribution of class y.
- $p(x|y)$ is the probability of observing x from class y (prior probability).
- $p(y|x)$ is the conditional probability of given an observation x, it is classified as y.

3 Support Vector Machine (SVM)

The first classifier we have to use is the Support Vector Machine (SVM) classifier. It is a discriminative classifier, since it learns how to separate the two classes y starting from features x. Formally, we can define SVM linear classifier as:

$$h_{w,b}(x) = g(w^T x + b)$$

$$g(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if otherwise} \end{cases}$$

Where, $g(z)$ is the decision function which returns the predicted label for the observation x, and $w^T x + b$ is the separating hyper-plane for the two classes. In our case, the classifier will assign label 1 (spam) if $h_{w,b}(x) \geq 0$, otherwise 0 (ham).
The probability of a certain classification given the observation x, can be written as follow:

$$p(y|x; w, b)$$

So we can understand that the higher the value of $(w^T x + b)$, the higher the confidence that the label will be 1 (spam).

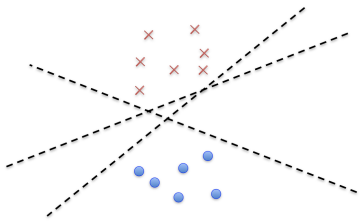

Figure 1: Two cluster separated by a lot of different possible hyper-planes.

Recalling that, observations can be seen as points in a hyper-space, the equation $w^T x+b = 0$ define a hyper-plane that represents the decision boundary for our problem. But looking at fig. 1, we can notice that even if the two clusters are linearly separable we can find an infinite number of hyper-planes that separate them, and each of these hyper-planes can produce different results.
SVM classifiers find the boundary furthest from the two clusters, in order to maximize the distance between the boundary and the points in the training set. We want this to avoid that a little perturbation of a point make the classification of such point wrong. The distance from boundary to the closest training point is called margin. The points that touch the margins are the so called support vectors. If we maximize the margin for each cluster, the margins will be the same for all the clusters.
The distance from boundary to one of the 2 margin is $\frac{1}{||w||}$, so from one margin to the other one is $\frac{2}{||w||}$.

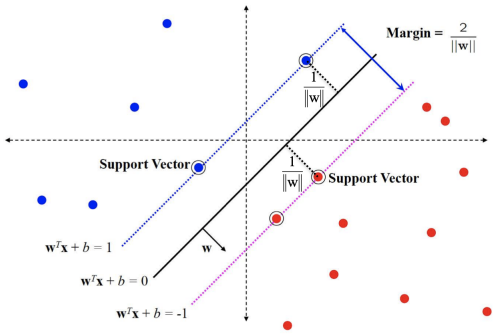

Figure 2: Graphical representation of 2 classes separated by a boundary with relative margins and highlighted support vectors.

To formalize the problem: given a training set $\mathcal{L} = \{(x_1, y_1), ..., (x_n, y_n)\}$, the goal of the SVM classifiers can be formulated as an optimization problem:

$$\max_{w} \quad \frac{2}{||w||} \tag{1}$$

$$\text{s.t.} \quad w^T x + b \geq 1, \quad \text{if } y_i = 1, \tag{2}$$

$$w^T x + b \leq -1, \quad \text{if } y_i = 0, \tag{3}$$

$$i = 1, ..., n. \tag{4}$$

It is easy to see that to maximize $\frac{2}{||w||}$ we can simply minimize $||w||$, so we can rewrite it as:

$$\min_{w} \quad \frac{1}{2}||w||^2 \tag{5}$$

$$\text{s.t:} \quad y_i(w^T x + b) \geq 1 \quad i = 1, .., n \tag{6}$$

The "new" problem optimize on a convex function with only linear constraints, so there is a unique minimum and all optima are global. Therefore, its solution is called optimal margin classifier.
SVM classifiers consider only the support vectors to determine the boundary. We have now as many inequality constraints as points in the training set. To solve the problem we use the Langrange function with m lagrange multipliers $(\lambda_1, .., \lambda_m)$ in order to rewrite the optimization problem into a one with the lagrange multipliers instead of parameters w and b. Taking advantage also of the dual representation, we can now write the problem as:

$$\max \quad L_D(\lambda_1, ..., \lambda_m) = \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

$$\text{s.t.} \quad \sum_{i=1}^{m} \lambda_i y_i = 0 \qquad \lambda_i \geq 0$$

Note, the training vectors x_i appear only as a dot product, many of the λ_i are zeros (sparse solution) and the only lagrange multipliers $\lambda_i > 0$ are the support vectors.
Thus, in the dual representation, the maximum margin hyper-plane is given by:

$$\sum_{i=1}^{m} y_i \lambda_i x_i^T x + b = 0$$

Sometimes, it can be useful to map the data from the original space into some other space. In SVM we can do this with the use of the "kernel trick". With the help of the mapping function $\phi(x)$ we can apply a mapping of a feature vector to another one. This strategy allow the SVM algorithm to learn using the vector $\phi(x)$ instead of the original feature vector x. Kernel function is the inner product between feature mapping of ϕ:

$$K(x, y) = \phi(x)^T \phi(y)$$

Since the SVM algorithms work with inner products between input vectors (x, y), we obtain:

$$\begin{aligned} \max \qquad & L_D(\lambda_1, ..., \lambda_m) = \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \\ \text{s.t.} \qquad & \sum_{i=1}^{m} \lambda_i y_i = 0 \qquad 0 \leq \lambda_i \end{aligned}$$

and also the discrimination function updates:

$$\textstyle\sum_{i=1}^{m} y_i \lambda_i K(x_i, x) + b = 0$$

Using kernels allow to learn in different spaces that sometimes can result even more precise than the original one and we don't really need to compute $\phi(x)$ but we just need to know the $K(x, y)$.

In the assignment, we are going to use the following kernels:

- Linear Kernel $\qquad K(x, y) = x^T y$;
- Polynomial Kernel of degree 2 $\qquad K(x, y) = (1 + x^T y)^2$;
- Gaussian RBF Kernel $\qquad K(x, y) = exp\{-||x - y||^2 / 2\sigma^2\}$;

3.1 Angular Kernel

The kernels that we just defined take in account the length of the vectors. So if two vector with same angle have different lengths, they won't result similar. To solve this, we should normalize the length of the vectors to make $||x||_2 = 1$. In this way kernels are no more affected by length of the vectors but they use only the cosine made by 2 vector x_i and x_j to tell if they differ or not.
To do this, we have to define a kernel and mapping function $\phi(x)$ as follow:

$$\phi : \mathbb{R}^m \to \mathbb{R}^m \qquad \phi(x) = \frac{x}{||x||}$$
$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j) = \frac{x_i}{||x_i||} \frac{x_j}{||x_j||} = cos(x_i, x_j)$$

In the assignment we are asked if this kernel which uses the cosine formed by two vectors is still positive. To prove it we have first to recall an important property.[2]

Definition 1. *A function* $K : \mathbb{R}^n x \mathbb{R}^n \to \mathbb{R}$ *is said to be* ***positive definite kernel*** *if :*

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

So, now with the help of this property we can prove that starting from a positive definite kernel and changing the kernel with one that use angular information, we are going to end up with still a positive definite kernel.

$$\begin{aligned}
&\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0 \qquad\qquad c_i, c_j \in \mathbb{R} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \left\langle \phi(x_i)^T \phi(x_j) \right\rangle = \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \sum_{k=1}^{m} \frac{x_{ik}}{||x_{ik}||} \frac{x_{jk}}{||x_{jk}||} = \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{m} c_i \frac{x_{ik}}{||x_{ik}||} c_j \frac{x_{jk}}{||x_{jk}||} = \\
&= \sum_{k=1}^{m} \left(\sum_{i=1}^{n} c_i \frac{x_{ik}}{||x_{ik}||} \right) \left(\sum_{j=1}^{n} c_j \frac{x_{jk}}{||x_{jk}||} \right) = \\
&= \sum_{k=1}^{m} \left(\sum_{i=1}^{n} c_i \frac{x_{ik}}{||x_{ik}||} \right)^2 \geq 0
\end{aligned}$$

where, n is the number of point and m the number of features.
Now, we have to prove it for the polynomial of degree 2 and RBF kernels. Their kernel functions can be written:

- Polynomial : $K(x_i, x_j) = \phi_p(x_i)^T \phi_p(x_j)$
- RBF : $K(x_i, x_j) = \phi_{rbf}(x_i)^T \phi_{rbf}(x_j)$

We already know that these two are positive definite kernels, so we just need to transform them in a way that they consider only angular information and not the length of vector.
Let's first see for polynomial of degree 2:

$$\phi'_p(x) = \phi_p\left(\frac{x}{||x||}\right),$$
$$K'_p(x_i, x_j) = \phi'_p(x_i)^T \phi'_p(x_j) = \phi_p(\frac{x_i}{||x_i||})^T \phi_p(\frac{x_j}{||x_j||}) = K_p(\frac{x_i}{||x_i||}, \frac{x_j}{||x_j||}) \geq 0$$

Now for RBF kernel:

$$\phi'_{rbf}(x) = \phi_{rbf}\left(\frac{x}{||x||}\right),$$
$$K'_{rbf}(x_i, x_j) = \phi'_{rbf}(x_i)^T \phi'_{rbf}(x_j) = \phi_{rbf}(\frac{x_i}{||x_i||})^T \phi_{rbf}(\frac{x_j}{||x_j||}) = K_{rbf}(\frac{x_i}{||x_i||}, \frac{x_j}{||x_j||}) \geq 0$$

Since the original kernels K_p and K_{rbf} are positive definite kernels, the property is satisfied also for K'_p and K'_{rbf}.
Therefore, all three kernels that use angular information are still positive definite kernels.

3.2 Implementation and Results

For the implementation of SVM classifiers, we used the library `sklearn` where these three classifiers are already provided by the library. Before applying the SVM classifier we needed to apply TF-IDF to the original dataset. The idea behind the TF-IDF representation is to give more relevance to the terms that appear in the document, but which in general are infrequent.

$$(tfidf)_{i,j} = tf_{i,j} * idf_{i,j}$$
$$tf_{i,j} = \frac{n_{i,j}}{|d_j|}$$
$$idf_{i,j} = \log_{10} \frac{|D|}{|d : i \in d|}$$

where, $n_{i,j}$ is the number of occurrences of term i in document j, $|d_j|$ is the dimension of document j, $|D|$ is the dimension of document in the whole collection and the denominator in $idf_{i,j}$ is the number of documents that contains the term i.[3]
We simply translated the previous formulas into python code, computing the tf-idf for all the terms in all the docs in the collection.

```
# compute tfidf for all docs
def tfidf_func(tf, n_doc):
    denom = np.zeros(len(tf[0]))

    for i in range(len(denom)):
        denom[i] = np.count_nonzero(np.transpose(tf)[i])

    idf = np.log10(n_doc / denom)
    return tf /100 * idf
```

After computing the tf-idf, with the library function of `sklearn` we create the three classifiers. The parameter `C` is used to control the trade-off between the accuracy with respect to the training data and the maximization of margin, basically how much violations of margins we tolerate.

```
frequences = dataset[:, :54]
classes = dataset[:, 57]
tfidf = tfidf_func(frequences, len(classes))
clf_l = SVC(kernel='linear', C=1, gamma='scale')
clf_poly = SVC(kernel='poly', degree=2, C=1, gamma='scale')
clf_rbf = SVC(kernel='rbf', C=1, gamma='scale')
```

To measure performances of these classifiers we use the 10-way cross validation that is based on the K-way cross validation iterative technique with K = 10. This technique consists on dividing the whole data in k equal parts and at each step, the k-th part is the validation part and the rest is used to train the model. This is done to avoid overfitting and asymmetric sampling of dataset

[4]. Overfitting happens when the model learns too well the behavior of the training set but when it has to predict new data, it doesn't generalize very well. Again, we used the function provided by the library `sklearn`. What follow is only a snippet of code to show how we used the library function `cross_val_score` for linear kernel over TF-IDF representation and for linear kernel that use angular information.

```
y = classes
x = tfidf
scores = cross_val_score(clf_l, x, y, cv=10)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
print ("MIN ACCURACY:", scores.min())
print ("MAX ACCURACY:", scores.max())

# with angular information
norms = np.sqrt(((tfidf + 1e-128) ** 2).sum(axis=1, keepdims=True))
normalized_lenghts = np.where(norms > 0.00, tfidf / norms, 0.)
scores = cross_val_score(clf_l, normalized_lenghts, y, cv=10)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
print ("MIN ACCURACY:", scores.min())
print ("MAX ACCURACY:", scores.max())
```

As we can se from the previous code, instead of re-writting the kernels such that they consider angular information we decided to normalize the whole dataset in order to use the already provided kernels.

Now we provide the results we got for all the models we mentioned.

Kernels	**Accuracy**	**Std. deviation**	**Min. accuracy**	**Max. accuracy**
Linear	0.61	0.00	0.6043	0.6108
Polynomial of degree 2	0.81	0.03	0.7478	0.8365
RBF	0.92	0.03	0.8478	0.9414

Table 1: Performances of kernels with tfidf and length information

Kernels	**Accuracy**	**Std. deviation**	**Min. accuracy**	**Max. accuracy**
Linear	0.92	0.03	0.8413	0.9673
Polynomial of degree 2	0.93	0.03	0.9286	0.9465
RBF	0.94	0.03	0.9285	0.9608

Table 2: Performances of kernels with angular information

Table 1 report the performances we got with kernel that are performed over tf-idf representation and that take in account length information. Instead, table 2 report the performances that we got with kernel that use angular information. As we can see the accuracy improved a lot for Linear and polynomial kernels with angular information and stabilize quite a lot for the RBF kernel with angular information.

To have a better comparison, we shuffled the dataset and divide it in 70% as training set and 30% for test set. Then, we trained the models with the training data and measured the accuracy on their prediction on the test data. Obtaining the following results From the table 3 and table 4, it is easy to see that accuracy its again better in kernels that use angular information and also for cosine kernels the percentage of ham emails predicted from the test data is nearer to the actual number of ham email (column 3 of both tables).

We have seen that SVMs consider as support vectors only the non-zeros

Kernels	**Accuracy**	**% of ham emails predicted**
Linear	0.6002	0.2474
Polynomial of degree 2	0.7726	0.4822
RBF	0.9102	0.9362

Table 3: Performances of trained SVM kernels

Kernels	**Accuracy**	**% of ham emails predicted**
Linear	0.9131	0.9923
Polynomial of degree 2	0.9312	0.9752
RBF	0.9297	0.9866

Table 4: Performances of trained SVM kernels with angular information

lagrange multipliers to establish the decision boundary. We got that our trained kernels have the following numbers of support vectors:

- Linear kernel: 2588;
- Polynomial kernel: 2064;
- RBF kernel: 1039.

While for cosine kernels the numbers of support vectors are a bit smaller:

- Linear kernel: 679;
- Polynomial kernel: 708;
- RBF kernel: 755.

We can see that use angular information instead of length, helped to decrease significantly the number of support vectors for all of the kernels. An higher number of support vectors means also a higher probability of miss-classified objects.
Another aspect we want to compare is the upper bound of `Expected Cross Validation error`.

$$\text{Expected CV error} \leq \frac{\text{Expected \# support vectors}}{\text{\# training samples}}$$

Again we report first the ratio of Expected # support vectors / # training samples for trained kernel that take care of length information.

- Linear kernel: 0.8037;
- Polynomial kernel: 0.6409;
- RBF kernel: 0.3226.

While for cosine kernels, we got:

- Linear kernel: 0.2027;
- Polynomial kernel: 0.2130;
- RBF kernel: 0.2341.

In this case the lower the value the better it is, because a lower value means the the expected cross validation error is more bounded.
If most points are support vectors it is a possible sign of over-fitting, so with lower values we have lower chance to over-fit and higher probability that the model is more generalized so it will have better predictions. These ratios confirm that cosine kernels for our case have better performance than the ones that use vector length information.

4 Naive Bayes

4.1 Introduction

While Support Vector Machines are discriminative classifiers, Naive bayes ones are classified as generative classifiers. They try to model $p(x|y)$ and $p(y)$ from the training set and then they compute $p(y|x)$ using the Bayes' theorem:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

Naive Bayes classifier assign the label that maximize the probability $p(y|x)$, so we can define as the following function:

$$argmax_y \qquad p(y|x) = argmax_y \qquad \frac{p(x|y)p(y)}{p(x)}$$

But we can avoid to compute $p(x)$ thus the Naive Bayes classification function becomes:

$$argmax_y \qquad p(x|y)p(y)$$

In the assignment, $p(x)$ is the probability of observing an email $x = \{x_1, ..., x_m\}$ where x_i is the frequency of a term i. Usally $p(x|y)$ and $p(y)$ are unknown, but we are considering the following assumption:

$$p(y = k) = a_k \qquad \forall k \in \{0, 1\}$$

where a_k is the frequency of class k.
For $p(x|y)$, the probability of observing a particular email knowing in which class it fall, we have to assume that "all the x_i are conditionally independent given y", meaning that the probability distribution of each random variables x_i and x_j is not affected by the presence of another given the class y.

Definition 2. *We say that X is **conditionally independent** of Y given Z if and only if:*

$$P(X = x_i|Y = y_j, Z = z_k) = P(X = x_i|Z = z_k) \qquad \forall i, j, k$$

This assumption allow us to write:

$$p(x|y) = p(x_1, ..., x_m|y) = \prod_{i=1}^{m} p(x_i|y)$$

An other essential assumption on $p(x|y)$ is to assume that they are a multivariate Gaussian and that each feature can be modeled with a Gaussian distribution. So we can write:

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

where μ_{ki}, σ_{ki}^2 are the means and variances of feature i given that the data is in class k.
Finally we have everything we need to compute the Naive Bayes classification function.

4.2 Implementation

Our choice for Naive Bayes implementation is to create a class that implement `BaseEstimator` in a way that we can use it as we did with SVM classifiers and to make easier the use of the functions of library `sklearn`.

```
class NaiveBayesClassifier(BaseEstimator):

        def fit(self, X, y):
                self.ham = X[y == 0, :54]
                self.spam = X[y == 1, :54]

                self.n_doc = float(len(self.ham) + len(self.spam))
                # p(y = 1)
                self.p_y1 = len(self.spam) / self.n_doc
                # p(y=0)
                self.p_y0 = len(self.ham) / self.n_doc

                # mean of features of class spam
                self.mean_1 = np.mean(self.spam, axis=0)
                # mean of features of class ham
                self.mean_0 = np.mean(self.ham, axis=0)
                # variances of feature i given that the data is in class spam
                self.var_1 = np.var(self.spam, axis=0) + 1e-128
```

```
            # variances of feature i given that the data is in class ham
            self.var_0 = np.var(self.ham, axis=0) + 1e-128

    def score(self, X, y):
            # pre produttoria
            p_x_y1_i = (2 * np.pi * self.var_1) ** (-1. / 2) *
                    np.exp(-1. / (2 * self.var_1) * ((X - self.mean_1) ** 2))
            # pre produttoria
            p_x_y0_i = (2 * np.pi * self.var_0) ** (-1. / 2) *
                    np.exp(-1. / (2 * self.var_0) * ((X - self.mean_0) ** 2))
            # p(x | y = 1)
            p_x_y1 = np.prod(p_x_y1_i, axis=1)
            # p(x | y = 0)
            p_x_y0 = np.prod(p_x_y0_i, axis=1)

            evidence = p_x_y0 + p_x_y1 + 1e-128
            p_x_y1 = p_x_y1 / evidence
            p_x_y0 = p_x_y0 / evidence


            # p_x = (p_x_y1 * self.p_y1 + p_x_y0 * self.p_y0) + 1e-128
            # p( y=1 | x) = p(x|y=1)*p(y=1) /p(x)
            p_y1_x = p_x_y1 * self.p_y1 #/ p_x
            # p( y=0 | x) = = p(x|y=0)*p(y=0) /p(x)
            p_y0_x = p_x_y0 * self.p_y0 #/ p_x

            winner_class = np.argmax([p_y0_x, p_y1_x], axis=0)
            return np.mean(winner_class == y)
```

What we do is computing all the probabilities and then for each we chose the label with the higher probability. This class allow us to compute the scores of Naive Bayes classifier easly with the `cross_val_score` function

```
clf_NB = NaiveBayesClassifier()
scores = cross_val_score(clf_NB, X, Y, cv=10)
print("Naive Bayes Classifier")
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
print("MIN ACCURACY:", scores.min())
print("MAX ACCURACY:", scores.max())
```

	Accuracy	Std. deviation	Min. accuracy	Max. accuracy
Naive Bayes	0.82	0.15	0.4695	0.9739
SVM cosine Linear	0.92	0.03	0.8413	0.9673

Table 5: Performances of naive bayes classifier

4.3 Tests and results

We found out that splitting the whole data into 70% for training test and the rest for test set, we have:

- P(ham) = p(y = 0) = 0.6043;
- P(spam) = p(y = 1) = 0.3957

As the Naive Bayes classifier assign the label $y \in \{0, 1\}$ which maximixe the probability $p(y|x)$, these results means that we have more probability to classify an email as ham rather than as spam.

Theoretically, Naive Bayes classifier should perform better than SVM, but as we can see from table 5, accuracy with naive bayes is lower than any SVM classifier with cosine kernel. So we decided to check if all the previously formulated assumptions are satisfied or not.
The first assumption is conditional independence of all x_i (term frequency), that means if you know the class of an email, knowledge of whether word i appears in the message will have no effect on your beliefs about whether a word j appears. Whereby we chose to use the Pearson's Chi squared test which tests a null-hypothesis computing the p-value. For this test the null-hypothesis is the independence of the two samples. With p-value < 0.05 we can't accept the null-hypothesis and we have to embrace the alternative hypothesis which means the two sample are dependent.

```
Pearson's Chi-squared test
        data:  spam$V16 and spam$V17
        X-squared = 53829, df = 42720, p-value < 2.2e-16


Pearson's Chi-squared test
```

```
        data:  spam$V19 and spam$V24
        X-squared = 96099, df = 62832, p-value < 2.2e-16

Pearson's Chi-squared test
        data:  ham$V16 and ham$V17
        X-squared = 17018, df = 11700, p-value < 2.2e-16

Pearson's Chi-squared test
        data:  ham$V19 and ham$V24
        X-squared = 41854, df = 20884, p-value < 2.2e-16
```

We took two pairs of features for each class, and we computed the chi-squared test. The results are that p-values are very small, hence we can easily reject the null-hypothesis and tell that there is a strong dependence between the pairs of features.
We recall that the Central Limit Theorem ensure that properly normalized sum of independent random variables tends toward a normal distribution even if the original variables themselves are not normally distributed [5]. Therefore, we take advantage of these theorem to check if the distributions of each feature, divided in spam and ham, are independent random variables.

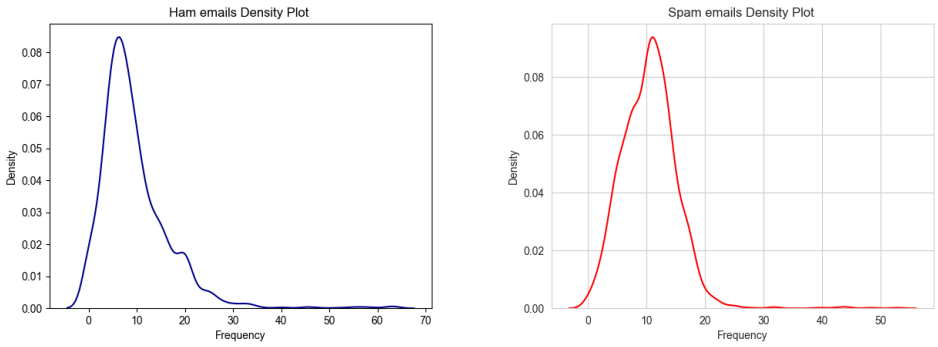

Figure 3: Density plots of emails classified as ham and as spam.

As we can see from the figure 3, the two distributions don't approximate

very well into a normal distribution so probably the distributions of the features aren't conditionally independent.

The other assumption is that the $p(x|y)$ are modeled as a multivariate Gaussian distribution. To check this hypothesis we used the Saphiro normality test and the D'Agostino's K^2 test which are already implemented in the `scipy` library. These two are both based on the computation of the acceptance of a null-hypothesis based on the computed p-value. The null-hypothesis is that the sample looks like a multivariate Gaussian distribution. We got the following results:

```
-------------------------Saphiro normality test--------------------------
Statistics=0.636, p=0.00000000
Ham prob distrib does not look Gaussian (reject H0)
Statistics=0.637, p=0.00000000
SPam prob distrib does not look Gaussian (reject H0)

-------------------------D-Agostino-s K**2 Test--------------------------
Statistics=16549.070, p=0.00000000
Ham prob distrib does not look Gaussian (reject H0)
Statistics=16403.238, p=0.00000000
SPam prob distrib does not look Gaussian (reject H0)
```

All the p-values are rounded to 0, so we can't accept the null-hypothesis and we have to embrace the alternative hypothesis that the $p(x|y)$ don't follow a multivariate Gaussian distribution.
To double check if the assumption "underlying distributions is a multivariate Gaussian", we take 4 random features from ham class and 4 from spam class in order to draw their Q-Q plots. Q-Q plot compare the cumulative distribution of a given sample with the cumulative distribution of a Gaussian. If the two distribution are similar their points should overlap.
As we can see from figure 4 and figure 5 none of the 8 cumulative distribution of the selected feature looks like the distribution of the normal. Hence we can tell that the Naive Bayes classifier isn't more accurate than the SVM because its assumptions aren't completely satisfied, $p(x|y)$ don't approximate correctly a multivariate Gaussian distribution and all the x_i aren't conditionally independent.

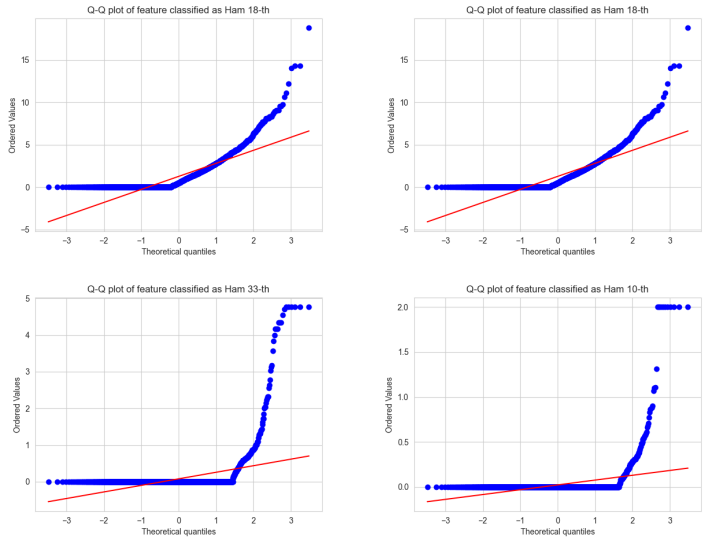

Figure 4: Q-Q plots of 4 random features of emails classified as ham, points colored in blue represent the cumulative distribution of the according feature.

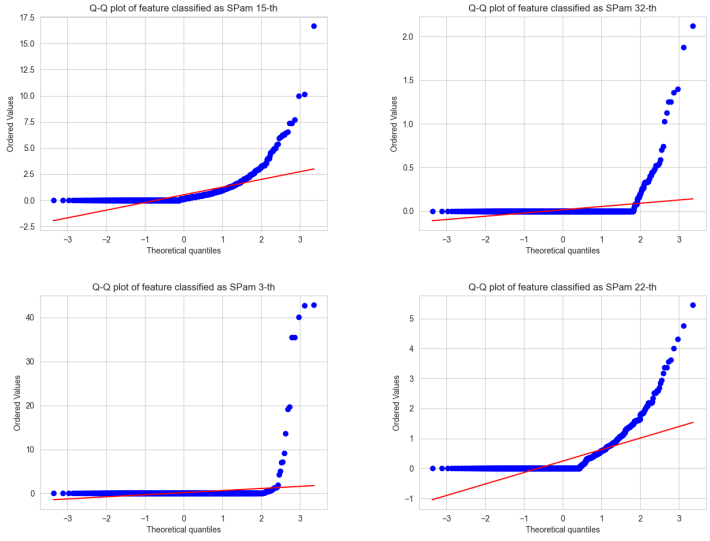

Figure 5: Q-Q plots of 4 random features of emails classified as spam, points colored in blue represent the cumulative distribution of the according feature.

5 K-NN

K-NN is an extension of the NN (Nearest Neighbor) classifier, a non-parametric method used for classification. Given a new object to classify, we extract its features in order to obtain a point x in the input space. Then, the NN classifiers search in the training set for the closest point z to x and assigns to x the same label of z. Positions in the space of the training points define a partition of the input space (decision regions). The resulting partition follows the property of the Voronoi Tessellation. In NN classifiers, the dimension of the training set is fixed, so there is not a training phase in which a model are built but it just uses the information given by the training set to classify new objects. Since classifier doesn't properly learn a model to use for future classification, it needs to store in memory the whole training set. This is why it is both one of the simplest algorithm in the machine learning and considered as a lazy learning algorithm.
In particular K-NN, to classify a point x, instead of looking only at the closest point, it looks for the k-closest points in the input space and assign a label to x taking a majority vote among the k-neighbors. The voting method should improve the robustness of the classifier. Even if it seems very good and simple, there are some drawbacks. As stated before we need to keep loaded in memory all the data from the training set and if this is very large we require an efficient algorithm to find the closest points otherwise the search for neighbors would take very long times.
In the assignment it is required to set k = 5, so the model when has to label a point will look at the 5 nearest neighbors and take a majority vote among these five.

5.1 Implementation and Results

We chose to use the implementation of K-NN provided by the library `sklearn`. The only thing we need to do was to set the parameter `n_neighbors` equals to 5 and run the 10-way cross validation.

```
clf = KNN(n_neighbors=5)
scores = cross_val_score(clf, x, y, cv=10)
```

	Accuracy	Std. deviation	Min. accuracy	Max. accuracy
K-nn	0.93	0.01	0.9086	0.9434

Table 6: Performances of K-nn classifier

	Accuracy	% of ham emails predicted
K-nn	0.9036	0.9522

Table 7: Performances of trained K-nn classifier

Then we splitted the dataset into two parts, training set 70% and test set 30%. In the end we modelled the model, performed the K-nn on the training set in order to measure accuracy and to predict the labels for the test set.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
clf_knn = KNN(n_neighbors=5).fit(X_train, y_train)
print("Actual accuracy of trained K-nn: ", clf_knn.score(X_test, y_test))
print("Classifier find: ", np.count_nonzero(clf_knn.predict(X_test)),
      "ham emails while in the feature there was: ", np.count_nonzero(y_test))
```

From table 6 we can see that accuracy is pretty high and the standard deviation is low. Indeed it doesn't fluctuate too much, the difference between the minimum accuracy and the maximum one isn't very large. Also the percentage of ham email guessed is very near to the effective number.

In order to understand if K=5 is the proper number of nearest neighbors, we used function `GridSearchCV` provided by `sklearn`. In our case, it tested all the values in the range from 1 to 30.

```
parameters = {"n_neighbors": range(1, 30)}
gridsearch = GridSearchCV(KNN(), parameters)
gridsearch.fit(X_train, y_train)
print(gridsearch.best_params_)
```

The result was that the best value for K is in the range from 3 to 9 but most of the times we got K=5. Which means that 5 is the right number of nearest

neighbors to avoid strong changes from one point to another and to prevent that points are too close to each other, resulting in every prediction being the same.
RMSE (root-mean-square error) is a measure of accuracy, to compare forecasting errors of different models for a particular dataset, the results are called residuals if RMSE is performed over training data and are called prediction errors when performed over test data.
A further test we did, is to compute residuals and errors of K-nn classifier with K=5 and compare them with the residuals and errors given with K equal to the number of neighbors given by the GridSearchCV. We got the following results:

- residual of K-nn with k=5 : 0.2637;
- residual of K-nn with gridsearch : 0.2648;
- prediction error of K-nn with k=5: 0.2822;
- prediction error of K-nn with gridsearch: 0.2867.

Since both residuals and prediction errors almost coincide we can say again that the value 5 for K was a good choice. Another thing that we can notice is that the difference between residual and error is small, meaning that the classifier does not suffer from overfitting in fact it is generalizing quite well.

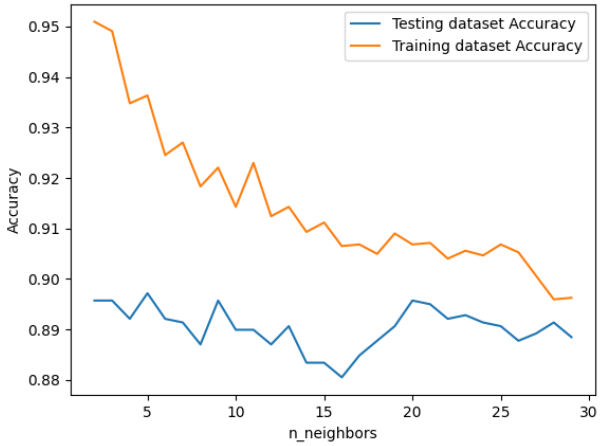

Figure 6

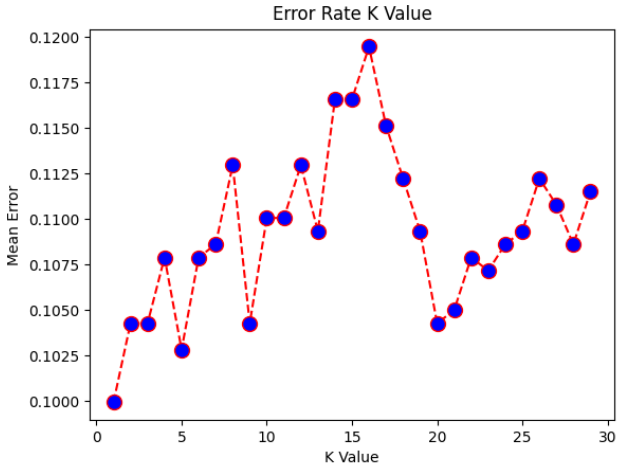

Figure 7

Plot in figure 6 shows that the maxima of the Test dataset Accuracy is when K=5 but at that point isn't close to the accuracy of training set. From looking only at this plot we could think that K=20 could be a better solution. However from figure 7 we see that the minima is when K is very small and at K=20 the mean error isn't very low. Although we can see that another good point is at K=5, where we have a very low error rate. So we can conclude that from these two figures a good trade-off, to have an high accuracy and a mean error as small as possible, is K=5.
Therefore from all these test we can tell that the best compromise is K=5 but in case we need more accuracy or a lower error rate we have a few other choices to fulfil our necessities.

Models	Times	Accuracy	Residuals	Pred. errors
SVM linear kernel	5.0625	0.61	0.62999	0.6171
SVM cosine RBF kernel	2.3424	0.94	0.2114	0.2451
Naive Bayes	0.0234	0.82	0.4229	0.4271
K-nn	0.0112	0.93	0.2637	0.2822

Table 8: Performances of different classifiers

6 Comparison and Conclusion

The goal of the assignment is to build some spam filters with three different classifiers SVM, Naive Bayes and K-nn, perform them over a dataset of emails and compare their results. The dataset already represents spam/ham email through a bag-of-words representations, each classifier should find a function that let them assign a label 1 (spam) or 0 (ham) for each given emails.
The first difference between these three classifiers is that SVM and Naive Bayes try to learn a model to use for future classifications while the K-nn doesn't properly learn, it just maps the training points into a input space in order to later compare them with the test points to classify. In fact for the first two classifiers don't matter if we maintain in memory the training data because after creating a model they won't ever use such data. On the other side for the K-nn we need to store the whole training set for all the time we need the classifier. This could be a disadvantage of K-nn because if we have a lot of data, we'll need also a lot of space in memory to store them and we'll need also more time to find the K neighbours of a particular point. A big difference in the behaviours of SVM and Naive Bayes classifiers is that the first one is considered as a Discriminative classifier and the second one is a Generative classifier.
Discriminative classifier try to model a decision boundary $h(x)$ which has the role to divide as good as possible the points classified as spam from the points labelled as ham, without taking care of how the points could be generated.
Generative classifier, instead, focus on learning which is the underlying distribution. Taking advantages of some assumption, it tries to model the joint distribution $p(x, y)$ from data. One advantage of Generative classifiers is that once the model has knowledge of the joint distribution, it will be easy to generate new possible pairs (x, y).
Now let's analysis the case where we have an outlier. SVM classifiers can

easily classify it, just looking at in which side of boundary it is, and if it is very far from the boundary we may assumes that the point is simply affected by noise. K-nn also can label it with the search for its K neighbours, and maybe only the distance from the outlier to its k closest point will be a bit larger than others. Obviously also with Naive Bayes classifiers we can do it just assigning the label of the class with higher probability $p(x|y)$ but Generative classifiers and so Naive Bayes can also identify the point as a possible outlier. If both $p(x|y = spam)$ and $p(x|y = ham)$ are very small, we can say that there is another possible distribution and so we may discover some underlying distributions.
Another comparison metric is how much time the three different models need to complete their learning phase. As we can see from table 8, SVMs are the slowest because they need a lot of iterations to find the best fitting hyperparameters in order to find the right hyperplane, even if they take in account only the support vectors. Instead, Naive Bayes and K-nn are way faster. In Naive Bayes, training phase is done with a single iteration where it only computes all the probabilities it needs and it uses less data than SVM. K-nn is even faster than Naive Bayes because it simply maps all the training set into a input space and it's done, the drawback is that K-nn need to keep in memory all the points of training set while for the other two we don't need to do it.
Now we focus on the accuracy. from table 1 and 8 we see that SVMs that use length information aren't very accurate and their accuracy can oscillate quite a lot. But with a simple improvement, using angular information, we get pretty high and stable accuracy. K-nn seems to be pretty accurate too in classification of emails. Also Naive Bayes should be accurate, unfortunately only theoretically. A weak point of Naive Bayes is that It is based on the assumptions that variables are conditionally independent and features can be modelled as multivariate Gaussian distributions. As we have seen performing some test, these two assumptions aren't completely satisfied, hence, with the increase of the size of the dataset, K-nn and SVM with cosine kernel perform way better than Naive Bayes classifier. This mainly happen because, in our case, Naive Bayes classifier can't capture the dependency structure when the dataset' size increase. Usually if both assumptions are fulfilled, Naive Bayes should learn well the underlying structure of the data and outperform the other two classifiers.
The last comparison we did is with the confusion matrix. Looking at table 9 we can see that SVM classifiers with kernels that take in account vector

	Ham	Spam
Ham	832	0
Spam	547	2

	Ham	Spam
Ham	820	12
Spam	271	278

	Ham	Spam
Ham	801	31
Spam	74	475

	Ham	Spam
Ham	781	53
Spam	46	501

	Ham	Spam
Ham	794	40
Spam	49	498

	Ham	Spam
Ham	794	40
Spam	44	503

Table 9: Confusion matrix of SVM classifier with length information (up) and angular information(down). SVM with linear kernel (left), SVM with polynomial kernel (center), SVM with RBF kernel (right).

	Ham	Spam
Ham	595	218
Spam	20	548

	Ham	Spam
Ham	788	54
Spam	69	470

Table 10: Confusion matrix of Naive Bayes (left) and K-NN (right) classifiers

length information misclassified a lot of objects. For example, SVM with linear kernel classified as "ham" 547 object that should be labelled as "spam" and SVM with polynomial of degree 2 kernel classified as "ham" 271 object that should be labelled as "spam". Also Naive Bayes model has a lot of misclassified objects, as we can see from table 10, 218 ham emails are misclassified as spam, almost 30% of the total ham emails. Instead K-nn classifier worked well, and the best are SVM with cosine polynomial kernel and SVM with cosine RBF kernel. As we imagined, the results given by the confusion matrix are the same we got looking at the accuracies of the various models.

To sum up we can say that we started trying three different types of kernels for the Support Vector Machine classifiers over TF-IDF representation of the data. Then, we saw that we can apply the "kernel trick" to map the feature space into another one, with same or higher dimensionality. So, taking advantage of "kernel trick" and knowing that SVM use the dot product, and this one is affected by the length of the feature vectors, we normalized the vectors' length to 1 in order to evaluate only the angular informations. So we

obtained SVM with cosine kernels that outperformed the first three models. Then we saw, a generative classifier, the Naive Bayes which tries to learn the underlying distribution of the data but is based on two fundamental assumptions which, however, can turn out to be its weakness. In the end we saw a lazy learning algorithm the K-nn, which doesn't really learn from the training set but it rather maps all the objects in the training set into a input space that will be used to choose the labels for new objects.

Therefore we can conclude,to classify emails, SVM with RBF kernel that use angular information is the classifier with the highest scores and accuracy but it needs a long time for the training phase. On the other hand, if we want a fast algorithm we can opt for the K-nn model that for a bit less accuracy and precision can be trained very fast but needs to store all the training data. Unfortunately, probably because of the unsatisfied assumptions on the dataset, Naive Bayes didn't performed as expected. and it wasn't able to learn very good any patterns.

References

[1] Classifiers: https://towardsdatascience.com/machinelearningclassifiers-a5cc4e1b0623

[2] Positive definite kernel function: https://en.wikipedia.org/wiki/Positive-definite_kernel

[3] TF-IDF representation: https://it.wikipedia.org/wiki/Tf-idf

[4] K-cross validation: https://it.wikipedia.org/wiki/Convalida_incrociata

[5] Central Limit Theorem: https://en.wikipedia.org/wiki/Central_limit_theorem

[6] RMSE: https://realpython.com/knn-python/#fit-knn-in-python-using-scikit-learn

[7] Confusion Matrix: http://guidetodatamining.com/assets/guideChapters/DataMining-ch5.pdf