Spam Filter

Tommaso Buoso

April 2021

Contents

1 Introduction

A spam filter is a program that is used to detect unsolicited and unwanted emails and prevent those messages from getting to a user's inbox. [1]

In this report, we will analyze a spam filter that uses discriminative and generative classifiers (developed in collaboration with Nicanor Tintari 866510), written in python, that works on a Spambase dataset [2] which represents spam/ham messages through a bag-of-words representation through a dictionary of 48 highly discriminative words, in the form of 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail, and 6 characters, in the form of 100 * (number of CHAR occurences) / total characters in e-mail. We ignored features 55-57, and feature 58 is the class label (1 spam - negative entry/0 ham - positive entry). [9]

The classifier implemented can be distinguished in three main categories based on the supervised learning model:

- SVM: algorithm that outputs an optimal hyperplane which categorizes new insances, e.g. in two dimensional space case this hyperplane is a line dividing a plane into two parts wherein each class lay in either side [4];
- Naïve Bayes: algorithm that uses the Bayes theorem (we'll explain better in section 4) to calculate the probabilities for every document and then choose the label with the highest probability;
- k-NN: non-parametric algorithm that given an element on a hyperplane, and the nearest k element of the training set on the hyperplane, takes a majority vote to choose the label.

The library contains algorithms written by hand and algorithms that use the library scikit-learn. [5]

1.1 Evaluation methods

For each classifier, we will compute a first model and prediction based on a train-test split of the Spambase dataset weighted with the TF-IDF representation (see section 2), then for this result we calculate:

- time: how much time the model spent on training and/or prediction;
- confusion matrix: given the true labels and the predicted labels, it computes a 2×2 matrix that contains on the diagonal the number

of true-positive (TP) and true-negative (TN) predicted by the model, and on the antidiagonal the false-positives (FP) and false-negatives (FN);

- precision: it is the ability of a classifier to not label an actual positive or negative instance as its opposite, e.g. for positive instances is calculated as $\frac{TP}{TP+FP}$;
- recall: it is the percentage of positive/negative instances correctly classified, e.g. for positive instances is calculated as $\frac{TP}{TP+FN}$;
- accuracy: measure how close a measured value is to the actual value, i.e. the proportion of correct predictions, and it is computed as $\frac{TP+TN}{TP+TN+FP+FN}$.

To avoid overfitting or selection bias and provide a good idea of the generalization ability of the model, we use the cross-validation technique: an algorithm that divides data into k subset, train on k-1 subsets and test on the held-out subset, and return the average test score, e.g. the accuracy, overall k tests. In our case, the scores computed are on the label 1, i.e. *spam*.

2 TF/IDF representation

Given that we are developing a spam filter, we have to weight correctly doc terms in the vectors, such that terms that appear often in a document should get high weights and terms that appear in many documents should get low weights. In order to do that we use the TF/IDF representation, so for each term i of the document j:

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right)$$

where $tf_{i,j}$ is the number of occurrences of i in j, df_i is the number of documents containing i and N is the total number of documents.

2.1 Implementation

In addition to following the definition of tf-idf, to obtain the value of $tf_{i,j}$, we have divided the values of spambase by 100, as shown in listing 1, since they represent a percentage.

```
N = len(spambase)

df = np.zeros(len(spambase[0]))
for i in spambase:
  df[i > 0] += 1

w = copy.deepcopy(spambase)
for j in range(len(spambase)):
  for i in range(len(spambase[j])):
    w[j][i] = spambase[j][i] / 100 * np.log(N/df[i])
```

Listing 1: TF/IDF representation implementation

3 SVM

As written in the introduction, an SVM is a classifier defined by the separation of hyperplanes, that categorize the new data that arrive on them.

The classifier is formally defined by the function: $h(x) = sign(x \cdot w + b)$, that defines the separate hyperplane which works as a decision boundary. We can define the functional margin as: $\gamma^{(i)} = y^{(i)}(wx^{(i)} + b)$. Our purpose is to maximize the distance between data and decision boundary, i.e. the geometric margin, defined as $\gamma_{geom} = \min_{i=1;...;m} \gamma_{geom}$, respect to the training set $S = \{(x^{(i)}; y^{(i)}); i = 1; ...; m\}$, where $\gamma_{geom} = \frac{\gamma}{||w^*||}$.

So, we have the maximization problem:

$$\max_{\gamma_{geom},w,b} \frac{\gamma}{||w^*||} \; s.t. \; y^{(i)}(wx^{(i)} + b) \geq \gamma \; i = 1; ...; m$$

where $y^{(i)}(wx^{(i)} + b) \geq \gamma$ means that points must have minimum distance γ from the decision boundary. Then if we pick the margin to closest points to 1 we have that:

$$+1(w \cdot x_1 + b) = 1$$
$$-1(w \cdot x_2 + b) = 1$$

and by combining this we obtain: $\frac{w}{||w||}(x_1 - x_2) = \frac{2}{||w||}$. Now we can impose the scaling constraint $\gamma = 1$ and transform our maximization problem in an optimization problem as a minimization of $||w||^2$:

$$\min_w \frac{1}{2}||w||^2 \; s.t. \; y^{(i)}(wx^{(i)} + b) - 1 \geq 0 \; i = 1, ..., m$$

Then, we turn the constrained problem into an unconstrained problem thanks to the Lagrange multiplier, in order to obtain:

$$\min_w \left(\frac{1}{2}||w||^2 - \sum_i \alpha_i (y^{(i)}(wx^{(i)} + b) - 1) \right) \ \alpha_i \geq 0, i = 1, ..., m$$

Then we can find the optimal set of w^* os a function of α:

$$L(w, b) = \frac{1}{2}||w||^2 - \sum_i \alpha_i (y^{(i)}(wx^{(i)} + b) - 1)$$

that is minimized when: $w^* = \sum_i \alpha_i y^{(i)} x^{(i)}$ and $\sum_i \alpha_i y^{(i)} = 0$. By substituting w^* in the function L, we get the dual Lagrangian:

$$L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y^{(i)} y^{(k)} x^{(i)} \cdot x^{(k)}$$

Now the optimization problem become: $\max_\alpha L(\alpha)\ s.t.\ \sum_i \alpha_i y^{(i)} = 0,\ \alpha_i \geq 0, i = 1; ...; m$, so when $\alpha_i = 0$ the constraint is satisfied with no distortion, otherwise if $\alpha_i \geq 0$ the constraint is satisfied with equality and $x^{(i)}$ is a support vector, i.e. a small fraction of the dataset that determine the classifier.

In order to reduce the effects of misclassification, as shown in figure 1, we have to lighten the margin constraint by adding slack variables to allow some errors in the boundary.

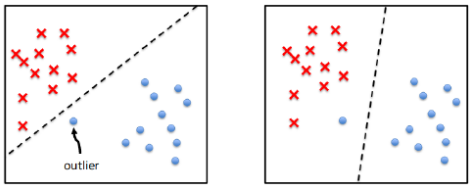

Figure 1: Outlier effect example in SVM

So, we can transform the optimization problem in:

$$\min \frac{1}{2}||w||^2 + C \sum_i \zeta_i\ s.t.\ y^{(i)}(wx^{(i)} + b) \geq 1 - \zeta_i,\ \zeta_i \geq 0,\ i = 1, ..., m$$

from that, we can observe that the optimal slack satisfy: $\zeta_i = (1 - y^{(i)}(w \cdot x + b))_+$, where $(k)_+ = \max(k, 0)$. With this relaxation SVM can be seen as

minimizing the loss function: $V(f(x,y)) = (1 - yf(x))_+$ called Hinge Loss. Then, the dual representation of the problem can be reformulated as:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y^{(i)} y^{(k)} x^{(i)} \cdot x^{(k)} \; s.t. \sum_i \alpha_i y^{(i)} = 0, \, 0 \leq \alpha_i \leq C, \forall i$$

where, the higher the C, the more stringent the classifier, and vice versa.

All of that is valid for linearly separable cases, but the assignment given to us requires to use different kernels, that are useful for SVM if we want to transform the feature values in a non-linear way, to transform what was not linearly separable into one that is. So, when training samples are not separable in the original space, we can transform them into a higher dimensional space to find separation. In order to get the transformation, we use the function $\phi(x)$, and, given that SVM only use dot products, to train the classifier we need $\phi(x^i) \cdot \phi(x^j)$, while, to run classifier we need $\phi(x^i) \cdot \phi(u)$. Then, we need a way to compute dot products in the transformed space as a function of vectors in the original space, so we need a function K, such that $K(x^i, x^j) = \phi(x^i) \cdot \phi(x^j)$, on the low-dimensional inputs.

In this analysis we used:

- Linear kernel: $K(x^i, x^j) = x^i \cdot x^j$;
- Polynomial kernel of degree 2: $K(x^i, x^j) = (1 + x^i \cdot x^j)^2$;
- Radial Basis Function: $K(x^i, x^j) = exp(-\frac{1}{2\sigma^2}|x^i - x^j|^2)$.

The assignment also requires transforming the kernels to make use of angular information only and determine if they are still positive definite.

In order to answer the first request, we don't have to consider the length of the vector, so we can make use of the cosine information given from the equation derived by using the Euclidean dot product:

$$x^i \cdot x^j = ||x^i|| \, ||x^i|| cos(x^i, x^j) \rightarrow \frac{x^i}{||x^i||} \frac{x^j}{||x^j||} = cos(x^i, x^j)$$

Then, as explained before, we simply define $\phi(x) = \frac{x}{||x||}$, and we obtain for:

- Linear kernel: $K(x^i, x^j) = \phi(x^i) \cdot \phi(x^j) = \frac{x^i}{||x^i||} \frac{x^j}{||x^j||} = cos(x^i, x^j)$;
- Polynomial kernel of degree 2: $K(x^i, x^j) = (1 + \phi(x^i) \cdot \phi(x^j))^2 = (1 + \frac{x^i}{||x^i||} \frac{x^j}{||x^j||})^2 = (1 + cos(x^i, x^j))^2$;

- Radial Basis Function: $K(x^i, x^j) = exp(-\frac{1}{2\sigma^2}|\phi(x^i) - \phi(x^j)|^2) = exp(-\frac{1}{2\sigma^2}|\frac{x^i}{||x^i||} - \frac{x^j}{||x^j||}|^2) \rightarrow$ for the law of cosines $\|A - B\|^2 = (A - B)^\mathsf{T}(A - B) = \|A\|^2 + \|B\|^2 - 2A^\mathsf{T}B$, and if A and B are normalized to unit length, i.e. $\|A\|^2 = \|B\|^2 = 1$, this expression is equal to $2(1 - \cos(A, B))$ [3] $\Rightarrow exp(-(1 - \cos(x^i, x^j)))$.

For our implementation, we can easily normalize the data in order to make use of the cosine similarity, as shown in listing 2.

```
w_angular = [i / np.sum(i) for i in w]
```

Listing 2: TF/IDF representation implementation

Instead, for the second request, we can use Mercer's Reproducer Theorem:

Theorem 3.1 *Let $K : X \times X \rightarrow R$ be a positive-definite function, then there exist a vector space Y (possibly infinite-dimensional) and a function $\phi : X \rightarrow Y$ such that $K(x_1, x_2) = \phi(x_1) \cdot \phi(x_2)$.*

This means that we can substitute each x, initially positive-definite, with $\frac{x}{||x||}$ and still have a positive-definite kernel because they both belong to the same vector space X.

3.1 Linear kernel

For this kernel, as for the next, we use the scikit-learn library to perform the training and testing operation. In our analysis, we first perform a train and a test on a single split of the dataset in order to have an idea of the portion of prediction performed by the classifier, as shown in the listing 3.

```
from sklearn import svm, model_selection, metrics
X_train, X_test, y_train, y_test = model_selection.
    train_test_split(w, label, test_size = 0.33, random_state =
     2)
clf = svm.SVC(kernel = "linear")

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Listing 3: SVM linear kernel

This single training and prediction took us about 0.81 seconds and, from the confusion matrix shown in figure 2, we can observe that this model classify almost every entry as 0, so as *ham*, and we can notice a great number of false-positive results.

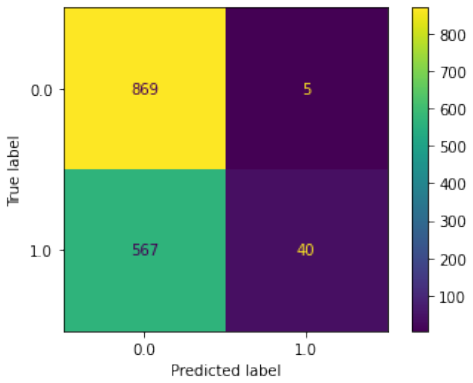

Figure 2: Confusion matrix - SVM classifier with linear kernel

Indeed, if we compute the classification report on this split, as shown in table 1, we can notice that the precision for the ham mail is equal to 0.61, i.e. 61%, while for the spam is equal to 0.89 ($= \frac{40}{40+5}$), i.e. 89%. At the same time, the recall is very high for ham and very low for spam.

label	precision	recall	f1-score	support
0.0	0.61	0.99	0.75	874
1.0	0.89	0.07	0.12	607

Table 1: Classification report - SVM linear kernerl

Furthermore, after the computation of the 10 ways cross-validation, as shown in listing 4, and by getting the average between the result vector, we get:

- Mean accuracy = 0.665;
- Mean precision = 0.952;
- Mean recall = 0.18;
- Mean fit time = 0.983 s;
- Mean score time = 0.087 s.

```
res = model_selection.cross_validate(clf, w, label, scoring = [
    "accuracy", "precision", "recall"], cv = 10)
```

Listing 4: SVM linear kernel - 10 ways cross-validation

By looking at these results, the accuracy, around 66%, reflects that the model does not fit well the data, but we get the major information from the precision. If we look at the 10 different precision values computed by the cross-validation, we get the vector $[1, 0.96, 1, 1, 0.97, 0.95, 1, 1, 0.73, 0.91]$, in which we can notice that, in the majority of cases, the models have a precision of 100% or almost 100%, probably because these data are not linearly separable and almost all data are classified as 0 by the model, and the few remaining values are correctly classified as 1 and this lead to high precision. Consequently, the recall has a very low value because of that high portion of false-positive values.

So, we can assert that this classifier is not suitable for our purpose, because it tends to classify every e-mail as *ham* and can't recognize a *spam* e-mail.

Even the speed of the algorithm, on average 1 second for training and 0.09 seconds for the evaluation, is not optimal (we will compare it to the others further on).

3.1.1 Linear kernel with angular information only

As requested from the assignment, we try also to evaluate the model with a kernel that makes use of angular information only, so as explained before we simply have mapped the data to their normalized values (*w_angular*) and followed the same procedure as before, as shown in listing 5.

```
from sklearn import svm, model_selection, metrics
X_train, X_test, y_train, y_test = model_selection.
    train_test_split(w_angular, label, test_size = 0.33,
    random_state = 2)
clf = svm.SVC(kernel = "linear")

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Listing 5: SVM linear kernel with angular information only

In this case, the single training and prediction took us about 0.33 seconds, and, from the confusion matrix shown in figure 3, we can observe that, differently from the previous case, the result are more balanced and accurate, with far less false-positive results.

Indeed, if we compute the classification report on this split, as shown in table 2, we can observe high values of precision and recall for both *spam* and *ham*, this means that in this specific case we classify the majority of the entry as their correct label, with only a minor part of false-positive or negative classifications.

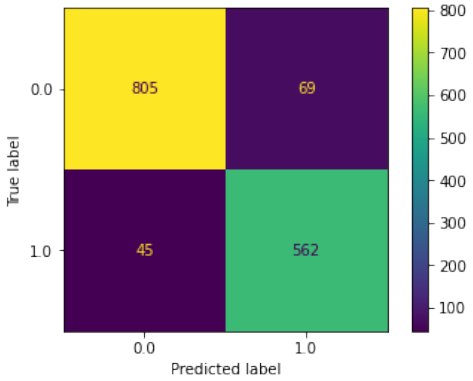

Figure 3: Confusion matrix - SVM classifier with linear kernel with angular information only

label	precision	recall	f1-score	support
0.0	0.95	0.92	0.93	874
1.0	0.89	0.93	0.91	607

Table 2: Classification report - SVM linear kernerl with angular info only

To confirm this, with the computation of the 10 ways cross-validation, we obtain:

- Mean accuracy = 0.912;
- Mean precision = 0.861;
- Mean recall = 0.937;
- Mean fit time = 0.391 s;
- Mean score time = 0.033 s.

From these results, we can confirm that in this version of the kernel, the model is more accurate than before (very good accuracy of 91%), and we have far better recall because the model produces only a small percentage of false-positive/negative. At a first look, we could have argued that precision was better in the previous case (mean precision = 0.952), but this is not true, because such a high precision was reached for label 1 thanks to all the false-positive classification on label 0 and in label 0 case higher the

number of false-positive lower the precision. So, given that our model has higher accuracy and recall, that confirms a lower number of false-positive and false-negative, and we can assert that this model, which uses angular information only, is also more precise than the previous one.

But why there is such a difference in using or not angular information only on this type of model? Because, we can prove by experiment, that if we take a document d and append it to itself, and call this document d', "semantically" d and d' have the same content, but the Euclidean distance between the two documents can be quite large. However, the angle between the two documents is 0, corresponding to maximal similarity. [8]

Then, we can also observe that the mean fit and score time for this model is more or less half of the previous kernel, probably because this SVM uses about half of the number of support vector, [468, 454] for this model, against [1192, 1189] of the previous, given us by the attribute *n_support_* of the classifier, because the number of iteration performed by the algorithm is linear proportional to the number of support vector used.

3.2 Polynomial kernel of degree 2

Using a polynomial kernel of degree 2 is the equivalent to mapping:

$$\phi(x) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_,1]$$

so, we can prove that [7]:

$$\begin{aligned}\phi(x) \cdot \phi(z) &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 + 2x_1z_1 + 2x_2z_2 + 1 \\ &= (1 + x_1z_1 + x_2z_2)^2 \\ &= (1 + x \cdot z)^2 \\ &= K(x, z)\end{aligned}$$

Even for this case, we used the scikit-learn library with an analog procedure as before, as shown in listing 6, with a first single train-test prediction, and then a 10 ways cross-validation.

```
X_train, X_test, y_train, y_test = model_selection.
    train_test_split(w, label, test_size = 0.33, random_state =
     2)
clf = svm.SVC(kernel = "poly", degree=2)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
6
7 res = model_selection.cross_validate(clf, w, label, scoring = [
    "accuracy", "precision", "recall"], cv = 10)
```

Listing 6: SVM polynomial kernel

The first single training and prediction took us about 0.71 seconds, and from the confusion matrix, shown in figure 4, we can notice that almost all the *ham* mail were classified correctly, while we have almost a split in half of the *spam* mail between true-negative and false-positive.

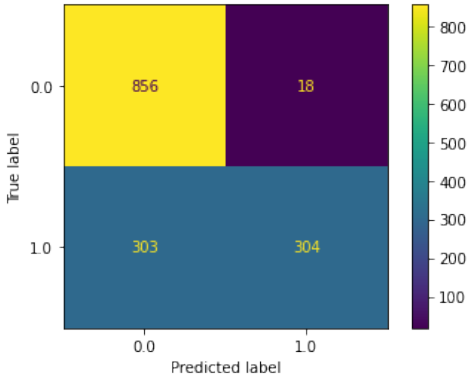

Figure 4: Confusion matrix - SVM classifier with polynomial kernel

More precisely, if we also compute the classification report on this split, as shown in table 3, we can observe for label 1 a high precision, because of the low number of false-negative obtained (only 18), and with a low recall, 0.5, because about 50% of the *spam* mail were classified as *ham*, so as false-positive, so if $TN \sim FP$, then $Recall = \frac{TN}{TN+FP} = \frac{TN}{2TN} = \frac{1}{2} = 0.5$.

label	precision	recall	f1-score	support
0.0	0.74	0.98	0.84	874
1.0	0.94	0.50	0.65	607

Table 3: Classification report - SVM polynomial kernel

Furthermore, in order to confirm this first look, we computed the cross-validation and we obtained:

- Mean accuracy = 0.821;
- Mean precision = 0.927;

- Mean recall = 0.606;
- Mean fit time = 0.86 s;
- Mean score time = 0.076 s.

Thanks to these results, we confirm that, while the precision is almost the maximum, the recall of around 60% indicates to us that almost half of the *spam* mails are classified as *ham*, so as false-positive. At the same time, the accuracy of the classifier, around 82%, is obtained because almost all the *ham* mail are classified correctly while, as written before, half of the *spam* are not.

If we look at the speed of this model, we can notice that it's slower than the model that uses angular information only, but a little faster than the first 'base' linear model, e.g. mean fit time of 0.86 s instead of 0.98 s. Also in this case probably because the number of support vectors, [996, 988], is respectively higher and lower than the number of support vectors used by the SVM with linear kernel that uses only the angular information and not only them.

We can conclude that this is not a good candidate for our purpose, given that it tends to classify correctly only about half of the *spam* mail, and its speed is not optimal with the respect to the second linear model that is both faster and more accurate.

3.2.1 Polynomial kernel of degree 2 with angular information only

Also for this model, we propose the version of the kernel transformed to make use of angular information only, implemented as shown in listing 7.

```
1 X_train, X_test, y_train, y_test = model_selection.
      train_test_split(w_angular, label, test_size = 0.33,
      random_state = 2)
2 clf = svm.SVC(kernel = "poly", degree=2)
3
4 clf.fit(X_train, y_train)
5 y_pred = clf.predict(X_test)
```

Listing 7: SVM polynomial kernel with angular info only

The first single training and prediction took us about 0.33 seconds, and from the confusion matrix, shown in figure 5, we can notice a more accurate prediction with a similar number of true-positive results, and a majority of true-negative results compared to the false-positive (558-49), differently from the previous half split (304-303).

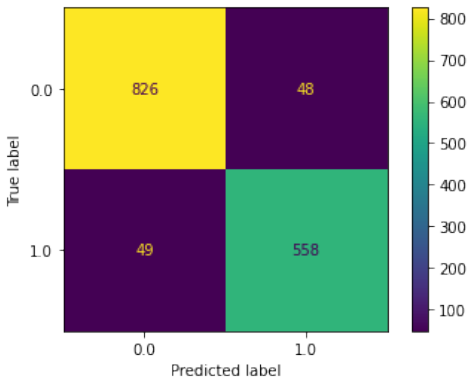

Figure 5: Confusion matrix - SVM polynomial kernel with angular info only

This is confirmed also from the classification report in table 4, where we can notice a higher precision for the classification of the label 0, and a much greater recall for the feature 1 (from 0.5 to 0.91), given by the increase of the number of correctly classifier *spam* mail and the drop of the false-positive results.

label	precision	recall	f1-score	support
0.0	0.94	0.95	0.94	874
1.0	0.92	0.92	0.92	607

Table 4: Classification report - SVM polynomial kernel with angular info only

Furthermore, with the computation of the 10 ways cross-validation, as shown in listing 8, we obtain the follow average results:

- Mean accuracy = 0.927;
- Mean precision = 0.901;
- Mean recall = 0.926;
- Mean fit time = 0.405 s;
- Mean score time = 0.032 s.

```
res = model_selection.cross_validate(clf, w_angular, label,
    scoring = ["accuracy", "precision", "recall"], cv = 10)
```

Listing 8: SVM polynomial kernel with angular info only

So, we can deduce that, also for the polynomial kernel, the use of angular information only brings a better accuracy, precision and recall to the model, compared to the previous SVM with polynomial kernel, for the reasons shown in the section 3.1.1.

Also, if we look at the mean fit and score time, we can observe a speedup of more or less 2 times compared to the previous model. Even in this case, we got this result because the number of support vectors used by this classifier, [441, 418], is almost half of the previous, [996, 988], with this new data representation.

3.3 RBF kernel

As shown before, when we use an RBF kernel, our kernel function is in the form of:

$$K(x^i, x^j) = exp(-\frac{1}{2\sigma^2}|x^i - x^j|^2)$$

In this case, the classifier is based on the sum of Gaussian bumps centered on support vectors.

As for the previous analysis, we first compute a single training and prediction on the same partition of data as before. The computation took us 0.52 seconds, and, by looking at the confusion matrix in figure 6, we can, differently from the first case of the linear and polynomial kernel, that the data are already pretty good classified, with only a small portion of false-positive and even less false-negative.

Indeed, by looking at the classification report in table 5, we can observe a higher precision for the *spam* and a higher recall for the *ham*, because we have, even if in a small percentage, a higher portion of false-positive result than false-negative.

label	precision	recall	f1-score	support
0.0	0.90	0.96	0.93	874
1.0	0.94	0.86	0.89	607

Table 5: Classification report - SVM classifier with RBF kernel

To confirm this, with the computation of the 10 ways cross-validation, we obtain:

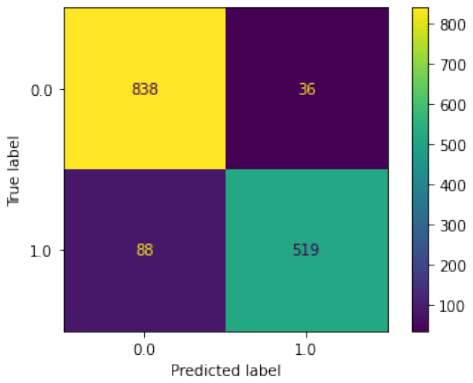

Figure 6: Confusion matrix - SVM classifier with RBF kernel

- Mean accuracy = 0.916;
- Mean precision = 0.913;
- Mean recall = 0.878;
- Mean fit time = 0.624 s;
- Mean score time = 0.059 s.

If we look at the precision and recall obtained, we can assert that, similarly to previous cases, we have a portion of values classified as *ham*, around 10%, that has as true label *spam*, and this, combined with a lower percentage of false-negative results, leads to and higher precision and a lower recall. Despite this, we obtained pretty good results, with a mean accuracy higher than 90%, and precision and recall around 90%.

Even the speed of the model is better compared to the previous two that does not make use of angular information only, e.g mean fit time equal to 0.624 seconds lower than the previous 0.86 seconds for the polynomial kernel model, also in this case because of lower the number of support vector used ([524, 487]).

3.3.1 RBF kernel with angular information only

Lastly, we computed the version of the RBF kernel with angular information only, following the same procedure as before. For the first computation of a single training and prediction, that took us about 0.4, we evaluate the

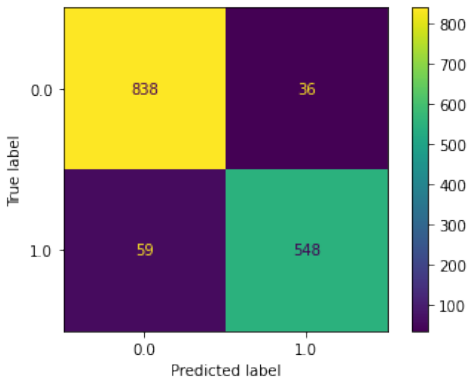

Figure 7: Confusion matrix - SVM RBF kernel with angular info only

confusion matrix, shown in figure 7, from which we can observe a decrease of the false-positive result, by maintaining the same number of true-positive.

The classification report in table 6, shows that, thanks to the decrease of false-positive, we have for both *spam* and *ham* precision and recall greater or equal to 90%.

label	precision	recall	f1-score	support
0.0	0.93	0.96	0.95	874
1.0	0.94	0.90	0.92	607

Table 6: Classification report - SVM classifier with RBF kernel

This results are confirmed from the cross-validation that computed the follow results:

- Mean accuracy = 0.926;
- Mean precision = 0.911;
- Mean recall = 0.91;
- Mean fit time = 0.476 s;
- Mean score time = 0.044 s.

These results confirm, as written in the previous paragraph, that the number of false-positive decrease with this model, so we obtain optimal accuracy,

precision and recall, all higher than 90%. As for the previous cases, also the speed of the algorithm has improved, e.g. mean fit time equal to 0.476 seconds compared to the previous 0.624 seconds, given that this SVM uses less support vector than the previous ([381, 348]).

3.4 Improving SVM

In order to improve the performances of these three SVM classifiers, in addition to transforming the kernel to make use of angular information only, we tried to use a higher C parameter, as shown in listing 9, to make the classification more stringent for the outliers, because, with the default C=1 of scikit-learn, SVM tends to misclassify a lot of values.

```
clf = svm.SVC(kernel = "linear", C = 100)
```

Listing 9: Improving SVM with linear kernel by set C

Indeed, if we recompute the 10 way cross-validation for the linear, polynomial, and RBF kernel in this way, and we compare the results with the previous version of the classifier, as shown in table 7, we can notice that a more stringent classification leads to:

- improved accuracy, all three higher than 90%, and also precision and recall are improved because, even if in some previous cases the precision for the label 1 is higher, from the larger recall we can deduce that the new models have a lower number of false-positive results, then the precision is higher for label 0;
- improved speed, also in this case, because of the lower number of support vectors used by the SVM, e.g. SVM with linear kernel uses [1192 1189] support vectors in the first case and [503 495] in the second case.

	$\text{Lin}_{C=1}$	$\text{Lin}_{C=100}$	$\text{Poly}_{C=1}$	$\text{Poly}_{C=100}$	$\text{RBF}_{C=1}$	$\text{RBF}_{C=100}$
Accuracy	0.665	0.906	0.821	0.911	0.916	0.927
Precision	0.952	0.888	0.927	0.93	0.913	0.919
Recall	0.18	0.882	0.606	0.843	0.878	0.9
Fit time	0.983	0.329	0.86	0.383	0.624	0.469
Score time	0.087	0.031	0.076	0.03	0.059	0.032

Table 7: 10 ways cross-validation - SVM with C=1 vs C=100

Contrariwise, if we try to make more stringent the SVM that makes use of angular information only, the new models overfit and have the worst performances and speed, as shown in table 8 of the previous classifiers.

	$\text{Lin}_{C=1}$	$\text{Lin}_{C=100}$	$\text{Poly}_{C=1}$	$\text{Poly}_{C=100}$	$\text{RBF}_{C=1}$	$\text{RBF}_{C=100}$
Accuracy	0.912	0.908	0.928	0.917	0.926	0.926
Precision	0.861	0.859	0.901	0.894	0.911	0.908
Recall	0.937	0.926	0.926	0.904	0.91	0.909
Fit time	0.319	0.571	0.336	0.658	0.398	0.454
Score time	0.029	0.022	0.029	0.019	0.039	0.031

Table 8: 10 ways cross-validation - SVM angular info only, with C=1 vs C=100

4 Naïve Bayes

The Naivë Bayes classifier assumes that x_i's (the feature of the dataset) are independent given y. So, in our case of spam detector, if we tell that y = 1 (that a particular piece of email is spam), the knowledge of whether "buy" appears in the message will not affect our beliefs about whether "price" appears. Note that this is not the same as saying that "buy" and "price" are independent. [6]

Given a training set in the form $(x^i, y^i); i = 1, ..., m$, where x^i has length n, we can write that:

$$p(x_1^i, ...x_n^i|y^i) = \prod_{j=1}^{n} p(x_j^i|y^i)$$

then, we can write the joint likelihood of the data as:

$$L(\phi_y, \phi_{i|y=0}, \phi_{i|y=1}) = \prod_{i=1}^{m} p(x^i, y^i)$$

where $\phi_y = p(y = 1), \phi_{i|y=0} = p(x^i|y = 0), \phi_{i|y=1} = p(x^i|y = 1)$.

After that, we maximise this with the respect to $\phi_y, \phi_{i|y=0}$ and $\phi_{i|y=1}$ to

obtain the maximum likelihood estimates:

$$\phi_y = \frac{\sum_{i=1}^{m} 1\{y^i = 1\}}{m}$$
$$\phi_{i|y=0} = \frac{\sum_{i=1}^{m} 1\{x_j^i = 1 \wedge y^i = 0\}}{\sum_{i=1}^{m} 1\{y^i = 0\}}$$
$$\phi_{i|y=1} = \frac{\sum_{i=1}^{m} 1\{x_j^i = 1 \wedge y^i = 1\}}{\sum_{i=1}^{m} 1\{y^i = 1\}}$$

Then, to classify, we compute:

$$\begin{aligned} p(y=1|x) &= \frac{p(x|y=1)p(y=1)}{p(x)} \\ &= \frac{(\prod_{j=1}^{n} p(x_j|y=1))p(y=1)}{(\prod_{j=1}^{n} p(x_j|y=1))p(y=1) + (\prod_{j=1}^{n} p(x_j|y=0))p(y=0)} \end{aligned}$$

and choose the class with the higher probability.

In our classification problem, we choose a parametric estimation to obtain the probability $p(x|y)$, i.e. we model each feature with a Gaussian distribution, so we get:

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

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.

4.1 Implementation and results

For our implementation of this classifier, we developed a python class called *MyNB*, that performs the training of the model with the function *fit*, e.g. takes as input X and y and calculates α, μ and σ^2, then the model also perform the classification with the function *predict*, as shown in listing 10, e.g. estimates the probability $p(x|y)$ and then choose the higher probability given by Bayes theorem.

```
1 def guassian(self, test_row, k):
2 res = 1
3
4 for i in range(len(test_row)):
```

```
  res *= ( ( 2*math.pi*self.sigma2[k][i] ) ** (-1/2) ) * math.
    exp(-( 1 / (2*self.sigma2[k][i]) ) * ( (  test_row[i] -
    self.mu[k][i]) ** 2 ) )
  #Note that a value greater than 1 is ok here, because it is a
     probability density rather than a probability

return res

def predict(self, X):
y_pred = []

for row in X:
  y_pred.append( int( ( self.guassian(row, 1) * self.alpha[1] )
     >= ( self.guassian(row, 0) * self.alpha[0] ) ) )

return y_pred
```

Listing 10: Naivë Bayes implementation

We also implemented the Naivë Bayes classifier version of the scikit-learn library, as shown in listing 11, in order to compare results.

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Listing 11: Naivë Bayes implementation with sklearn

For both the classifier implemented, we computed the confusion matrix on a single training and prediction, shown in table 9, from which we can observe that the results from the classifiers are almost equal, and we can notice that this model tends to misclassify about $\frac{1}{4}$ of the *ham* mails as *spam*, so we get a notable number of false negatives. This can lead to a lower precision for label 1, i.e. $Precision_1 = \frac{TN}{TN+FN} = \frac{576}{576+232} = 0.71$, and a lower recall for label 0, i.e. $Recall_0 = \frac{TP}{TP+FN} = \frac{649}{649+232} = 0.74$.

MyNB	GaussianNB
649, 232 24, 576	650, 231 23, 577

Table 9: Classification report - Naivë Bayes classifier

Then, we compute the 10 ways cross-validation on both classifier and we get the results shown in table 10. From these results, we can notice

that also in this case the values obtained are almost the same, so we have e not good precision for *spam* mail classification, and combining this with an high recall, we can confirm that the model produce a notable portion of false negative result. This may be because the Naivë Bayes classifier assumes that the feature of the dataset are distributed as a normal distribution, while in our case they are not, as we can see from figure 8 for example this data are skewed on the right tail.

	MyNB	GaussianNB
Mean accuracy	0.83	0.833
Mean precision	0.716	0.733
Mean recall	0.955	0.957
Mean fit time	0.007 s	0.004 s
Mean score time	0.189 s	0.003 s

Table 10: 10 ways cross validation - Naivë Bayes classifier

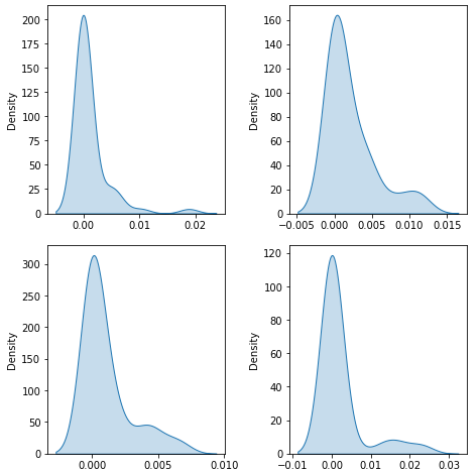

Figure 8: Example of data distribution

Despite a good accuracy, around 83%, this classifier can be not ideal for our purpose, because, given the percentage of false negative, we risk to classify a *ham* mail, that maybe can be important, as *spam*, so we can miss it. Indeed we prefer an opposite model that misclassify more *spam* rather than more *ham*.

Despite the classification, this model has an optimal fit and score time,

in the first case because the training of the model is simply the computation of α, μ and σ^2, while the time for classification of a new elements is linear dependent only on the number of feature of the document.

5 k-NN

The last classifier that tested was K-Nearest Neighbor. It is a non-parametric classifier where the only assumption is that the training set is fixed. For the classification, given a new point x, k-NN search in the training set for the k closest point to x and assigns the class with the greater number of voting. As n, the number of elements in the training set, and k increase, k more slowly than n, k-NN converges to the optimal classifier, in parallel grows the temporal complexity of the algorithm for the classification.

For the implementation of the algorithm, we preferred to use the cosine distance rather than euclidean distance to find the k closest point to x, for the reason explained in section 3.1.1.

5.1 Implementation and results

As for the SVM, for the implementation of the k-NN classifier, we used the scikit-learn library as shown in listing 12.

```
from sklearn import neighbors

X_train, X_test, y_train, y_test = model_selection.
    train_test_split(w, label, test_size = 0.33, random_state =
     2)
clf = neighbors.KNeighborsClassifier(5, metric = 'cosine')

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Listing 12: k-NN implementation

With this implementation we computed a fist single training and prediction, from which we extracted the confusion matrix, shown in figure 9, that shows a very good number of correctly predicted instances, with only a small percentage of false positive/negative results, i.e. less then 10%.

This is confirmed also by the classification report in table 11, with the only marginal flaw of the recall for the label 1 lower than 90%, given that with this split the model classifies an higher portion of data as false positive than false negative, so this recall is lower than that computed for label 0.

Then we computed the cross validation, and we get:

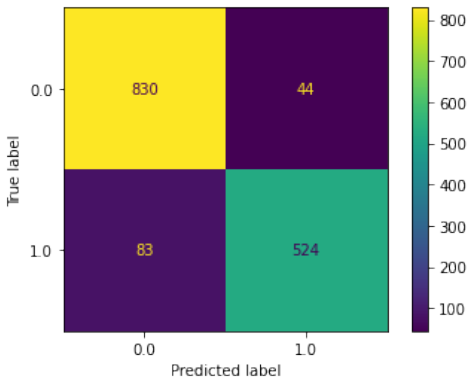

Figure 9: Confusion matrix - k-NN classifier

label	precision	recall	f1-score	support
0.0	0.91	0.95	0.93	874
1.0	0.92	0.86	0.89	607

Table 11: Classification report - k-NN

- Mean accuracy = 0.911;
- Mean precision = 0.891;
- Mean recall = 0.892;
- Mean fit time = 0.002 s;
- Mean score time = 0.067 s.

So, differently from the previous split, we can notice that all accuracy, precision and recall are around 90%, differently from the results obtained with the initial split, so this classifier is suitable for our purpose. Moreover also the speed of the algorithm is optimal, 0.002 seconds of mean fit time and 0.067 for scoring, probably because we work on a small dataset with a small k that does not affect the temporal complexity.

6 Comparison and conclusions

For choosing the best classifier to perform our spam detection, we must consider:

- classification metrics: if we look at accuracy, as shown in figure 10, we can notice that the different versions of the SMV that make use of angular information only and k-NN have an accuracy higher than 90%. So, we have to consider also precision and recall, and we get the best results of SVM with polynomial and RBF kernel with the angular information only, because with this models we have a lower risk to classify an *ham* mail as *spam*, so we have a lower risk to lose important information;

- speed: when we look at the mean times, we can observe that the best times, in decreasing order, are given from SVM with linear and polynomial kernel with the angular information only, because of the lower number of support vector used, Naivë Bayes, because of the almost constant time complexity of the training, and k-NN, because of the fast learning and the limited dimension of the dataset.

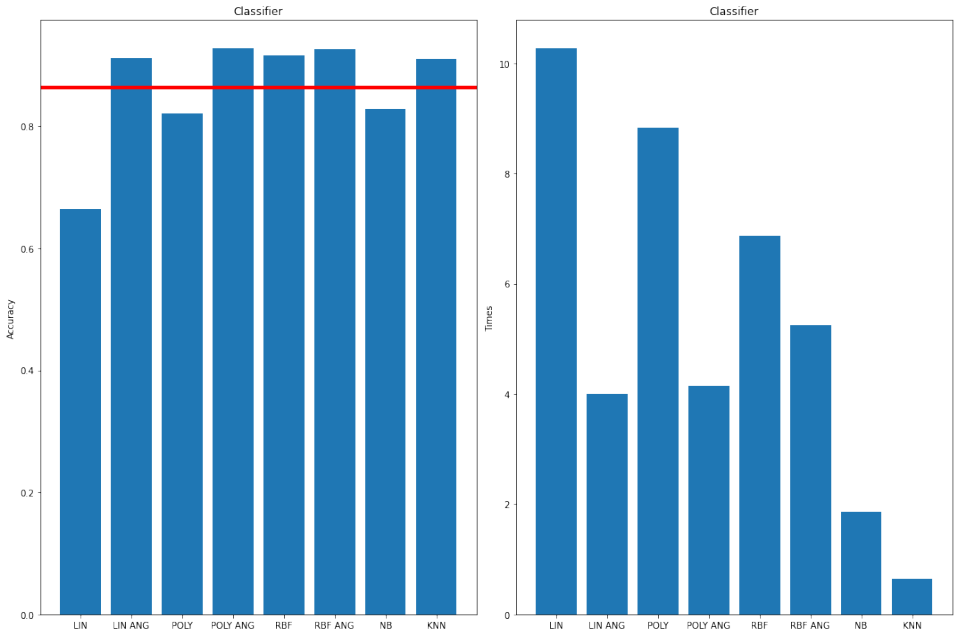

Figure 10: 10 way cross validation - Mean accuracy and time for each classifier

So, if we don't care so much about time, but we are only interested in obtaining the best result we'll choose the SVM with polynomial kernel that makes use of angular information only. Contrariwise, if we are looking for

the best absolute speed up, we'll prefer the Naivë Bayes classifier, because even with a larger dataset than the Spambased used it will maintain optimal performances. In conclusion, if we are looking for the best classifier for the problem tested from this report, we'll select the k-NN classifier, given that it is the fastest algorithm on this dataset and it has the results in line with the other best classifiers.

References

[1] URL: https://searchsecurity.techtarget.com/definition/spam-filter.

[2] URL: https://archive.ics.uci.edu/ml/datasets/spambase.

[3] URL: https://en.wikipedia.org/wiki/Cosine_similarity.

[4] Savan Patel. *Chapter 2 : SVM (Support Vector Machine) — Theory*. 2017. URL: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] Prof. Andrea Torsello. *Artificial Intelligence: Knowledge representation and planning - Generative*. 2021.

[7] Prof. Andrea Torsello. *Artificial Intelligence: Knowledge representation and planning - Linear Classifiers*. 2021.

[8] Prof. Andrea Torsello. *Artificial Intelligence: Knowledge representation and planning - Vector Model*. 2021.

[9] Prof. Andrea Torsello. *Assignment 2: Spam filter*. 2021. URL: https://moodle.unive.it/mod/page/view.php?id=294985.