Lab assignment 4: Support Vector Machines (SVMs)

January call, academic year 2024/2025

Module: Introduction to computational models
4th course Computer Science Degree (University of Córdoba)

17th September 2024

Abstract

This lab assignment serves to familiarise the student with support vector machines (SVMs). In this way, we will use the SVMs for various classification problems, so that the student can better understand their behaviour and, above all, the effect of their parameters. To do this, we will use its implementation in `scikit-learn` with the `libsvm` library, one of the most efficient implementations available today, as well as being free and open source. The delivery will be done using the task in Moodle enabled for this purpose. A single compressed file must be uploaded including the deliverables indicated in this script. The deadline for the delivery is **30th September 2024**. In the event that two students submit copied work, neither will be scored.

1 Introduction

In this lab assignment, we are going to carry out different exercises to better understand the behaviour of support vector machines (SVMs) in classification problems. SVMs been during many years the state of the art in pattern recognition problems, and it is important to know how they work and how they respond to their different parameters. The lab assignment will be done using the `libsvm` library through `scikit-learn`, which is one of the most popular and efficient implementations available today. In all cases, we will consider the "C-SVC" version, included in `libsvm`.

Section 2 describes a series of experiments to be carried out using synthetic datasets, which will help us to better understand the behaviour of SVMs and their parameters. Section 3 describes other experiments to be carried out on real databases. Finally, Section 4 specifies the files to be delivered for this lab assignment.

All questions in boxes like this should be answered.

2 Synthetic datasets

In this first part of the lab assignment, we will use a series of experiments on synthetic datasets, which will help us to understand how SVMs work and what effects the two parameters that need to be specified to train them have on the results.

2.1 libsvm installation

For this lab assignment we will use Python. `scikit-learn` already has a classifier that applies the SVC algorithm, using the implementation of `libsvm`. Specifically, the classifier is `sklearn.svm.SVC`.

2.2 2D representation of SVMs

To make the graphic representation in two dimensions from Python, we are going to use the *script* `libsvm.py`.

Question [1]: Open this script and explain its contents. You will see that the first dataset is used, and the SVM is graphically represented. Comment on what type of *kernel* is being used, and what are the training parameters.

2.3 First example dataset

Execute the above-mentioned *script* but comment the lines corresponding to the SVM's graphic representation, to see only the points of the dataset (the representation should be similar to that of Figure 1, `dataset1.csv`).

Question [2]: Intuitively, which hyper plane do you think will make the least mistake in the task of separating the two kinds of points?

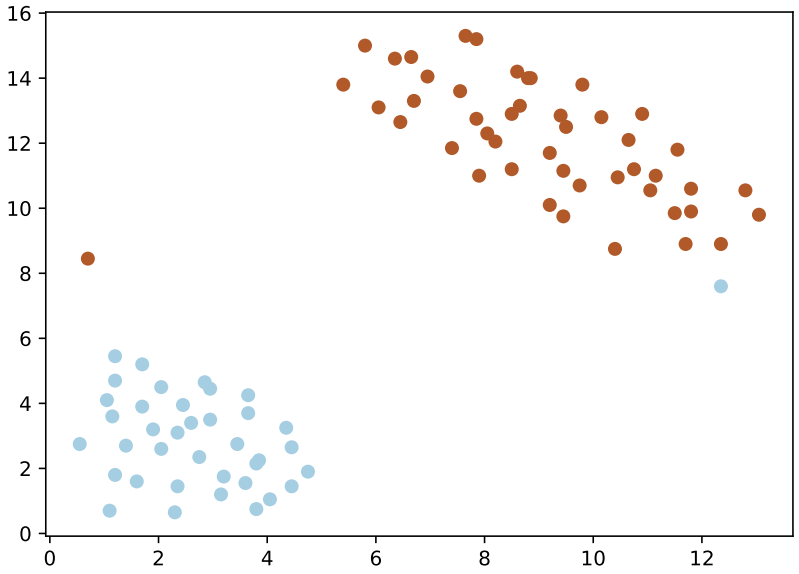

Figure 1: Synthetic or *toy* dataset #1.

In this part of the exercise, we will use a linear SVM, i.e. a SVM where the separation of the two classes will be a straight line. The way to specify the parameters of the SVC algorithm in Python can be found in `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`.

Let's try to use different values for the parameter C. Informally, we can say that the C parameter is a parameter that defines how much we will penalize the fact that mistakes are made in the classification. A very big value of C will result in an SVM that will make the minimum possible number of errors, while a small value of C will favour a classifier with the maximum margin, even if errors are made. Of course, this is related to the possible over-fitting (learning noisy training patterns that can mislead the model).

Question [3]: Modify the *script* trying different values for C, specifically, $C \in \{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}$. Observe what is happening, explaining why and select the most adequate value for C.

2.4 Second example dataset

In this case, we will work wit the *dataset* in Figure 2 (`dataset2.csv`).

Question [4]: Try running a linear SVM with the values for C used in the previous question. Do you get any satisfactory results in the sense that there are no errors in the training set? Why?

To achieve non-linear separating hyperplanes, SVMs use the so-called *kernel trick* which, explained informally, builds a linear model in a *many-dimensional* space and projects it back to the original space, resulting in a non-linear model. This is achieved through the use of *kernel* functions applied to each of the training patterns. The most common one is the RBF *kernel*, also called Gaussian. In order to apply it, the user has to specify an additional parameter (γ, gamma, in `libsvm` $\gamma = 1/radio$). A high radius tends to softer solutions, with less over-fitting, while a small radius tends to produce more over-fitting.

Question [5]: Propose a non-linear SVM configuration (using RBF or Gaussian *kernel*) that solves the problem. The result should be similar to the one in Figure 3. Move in powers of 2 using a range of the type $\{2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}\}$. What values have you considered for C and for γ? Also, include an example of a parameter configuration that produces over-fitting and another that produces under-fitting.

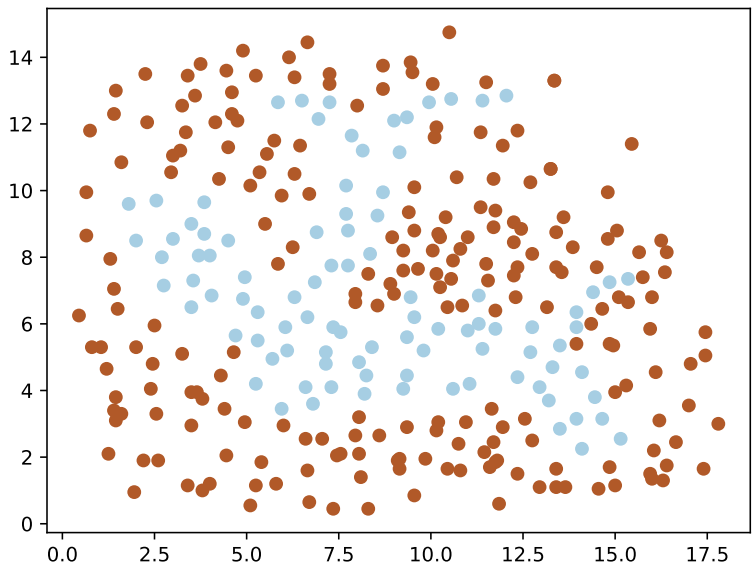

Figure 2: Synthetic or *toy* dataset #2.

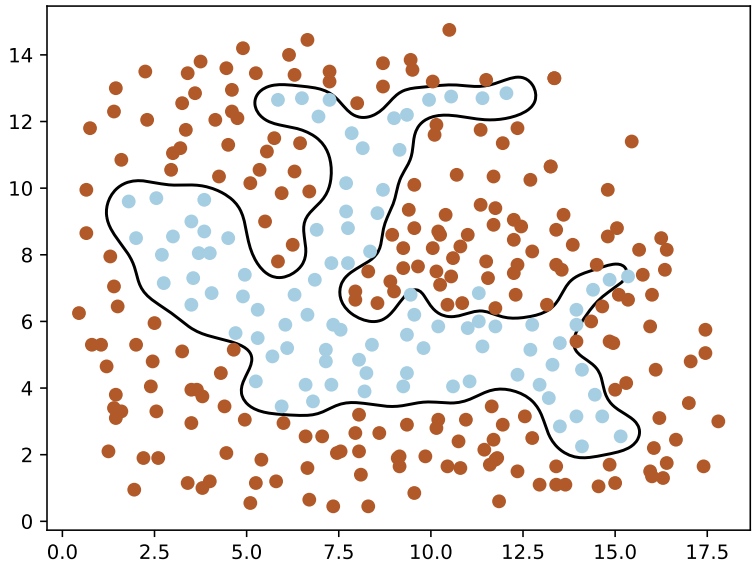

Figure 3: Synthetic or *toy* dataset #2 correctly classified.

2.5 Third example dataset

Finally, we are going to work with the *dataset* of Figure 4 (`dataset3.csv`).

Question [6]: In this case, is the *dataset* linearly separable?. At first sight, do you detect points that are presumably 'outliers'? Why?

Question [7]: Run a SVM to classify the data, in order to obtain a result as close as possible to that of Figure 5. Move in powers of 2 using a range of the type $\{2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}\}$. Set the value of the optimal parameters. In addition, include an example of a parameter configuration that produces over-fitting and one that produces under-fitting.

2.6 Console interface

Although we are using the Python interface, we will specify some features of the `libsvm` console interface[1], to better know the training process of a SVM. The console interface of `libsvm` is composed of three executable files or programs, which can be runned from the console[2]:

1. `svm-scale`: standardises a database file, so that it can be processed with the SVM classifier. Note that, using an RBF-type *kernel*, all input variables must be on the same scale, otherwise some variables will receive more importance than others when calculating distances. `svm-train`: trains an SVM model for a database file. The result of the training is saved in a file with the extension `.model`.

2. `svm-predict`: uses an already trained model to see how it generalizes into a new database (which must have the same structure). In other words, it carries out the *test* phase of the classifier, checking how it works for data that have not been used during training.

[1] Available `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[2] Read the file `README` from the root directory of `libsvm` to check how these programs work and which arguments they receive.

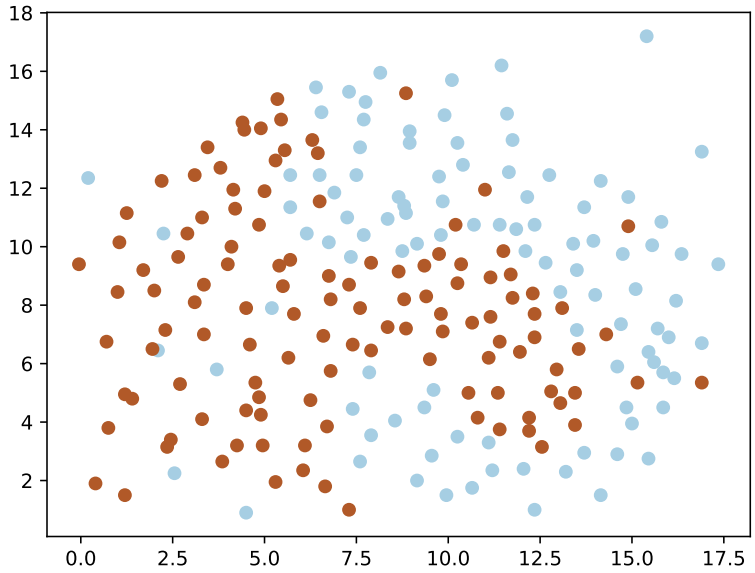

Figure 4: Synthetic or *toy* dataset #3.

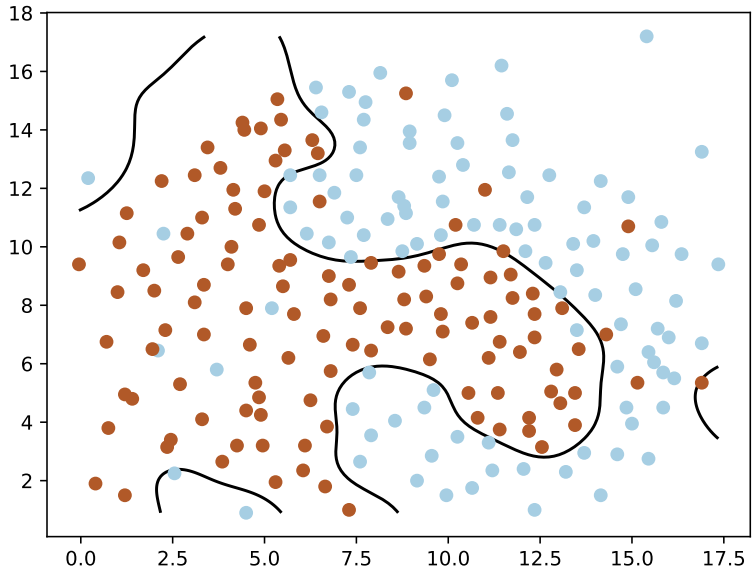

Figure 5: Synthetic or *toy* dataset #3 correctly classified.

Suppose we have two data files `train.libsvm` and `test.libsvm`. The standard SVM training process would involve first training the SVM using `svm-train` and the `train.libsvm` file, generating a SVM model. Secondly, we would use the program `svm-predict` to see how it generalizes using the data `test.libsvm`.

If we use the class `sklearn.svm.SVC` of `scikit-learn`, we will have to apply the standardization ourselves, making use of the StandardScaler[3] object. Here is a small example about how data can be standardised:

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

On the other hand, the class `sklearn.model_selection.StratifiedShuffleSplit` makes one or several partitions of a database in a 'stratified' way, that is, maintaining the proportion of patterns of each class in the original database[4]. Alternatively, you can use the `sklearn.model_selection.train_test_split` method[5].

Question [8]: We are going to reproduce this process in Python. Divide the synthetic dataset `dataset3.csv` into two stratified random subsets, with a 75% of patterns for training and a 25% of patterns for the test set. Make the complete training process (standardization, training and prediction), optimizing again the values of C and γ that you obtained in the last question. Check the accuracy that is obtained for the test set. Repeat the process more than once to check that the results depend a lot on the seed used to make the partition.

This process has a small inconvenience and it is that it is necessary to specify a value for the parameter C and a value for the parameter γ (width of the *kernel*, in case we want a non-linear classifier) and, as you already saw in the first part of the lab assignment, these parameters are very sensitive. To avoid this, there is a way to automatically adjust the parameters, following a *K-fold nested cross validation*:

1. We make a K-fold partition (where K is a parameter) of the training data. That is, we divide the training data into K disjoint subsets.

2. For each combination of parameters, we perform K training processes. For each training k from the K total processes:

   (a) We use the subset k as a test set (validation) and the rest of the subsets as a training set.

   (b) We accumulate the test error in a variable.

   At the end, we calculate the average test error made during the K trainings. The cross-validation process is repeated in full for all possible combinations of C and γ parameters. For example, if we are going to give 9 values to C and 9 values to γ and we are going to make a 5-fold cross-validation, this implies that we will make $9 \times 9 \times 5 = 405$ trainings. Note that up to now the test data has not been used.

3. We choose the combination of parameters that resulted in the least average error in the previous step and take those parameter values as the optimal values.

4. Using these parameters, we repeat the training process but now consider the complete training set (no subsets).

[3] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[4] `http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html`

[5] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

5. With the model obtained in the previous step, we check what is the error in test (actual test phase).

In `scikit-learn`, we can apply this process using the object `sklearn.grid_search.GridSearchCV`[6]. Here is a small example:

```
from sklearn.model_selection import GridSearchCV
Cs = np.logspace(-5, 15, num=11, base=2)
Gs = np.logspace(-15, 3, num=9, base=2)
optimal = GridSearchCV(estimator=svm_model, param_grid=dict(C=Cs,gamma=Gs),
                       n_jobs=-1,cv=5)
optimal.fit(X_train,y_train)
print optimal.score(X_test,y_test)
```

Question [9]: Extend the above code to perform the training of question 8 without the need to specify the values of C and γ. Compare the optimal values obtained for both parameters with those you obtained by hand. Extend the range of values to be explored, if you consider it necessary.

Question [10]: What drawbacks do you observe in adjusting the value of the parameters "by hand", checking the accuracy in the test set (which was done in question 8)?.

Question [11]: To be sure that you understand how the parameter search is performed, implement manually (without using GridSearchCV) the *K-fold nested cross validation* explained in this section. You may find useful the use of compression lists and the class `StratifiedKFold`. Compare the results with those you get using `GridSearchCV`.

3 Real-world dataset

Once you have become familiar with the SVM classifier and its linear and non-linear versions, in this part of the practice we are going to tackle various real problems, in order to see its applicability to more complex databases.

3.1 *COMPAS* dataset

ProPublica COMPAS: This dataset is about the performance of COMPAS algorithm, a statistical method for assigning risk scores within the United States criminal justice system created by Northpointe. It was published by ProPublica in 2016 [7], claiming that this risk tool was biased against African-American individuals (we will deal with this in the following assignment). In this dataset, they analyzed the COMPAS scores for "risk of recidivism" so each individual has a binary "recidivism" outcome, that is the prediction task, indicating whether they were rearrested within two years after the first arrest (the charge described in the data). We reduced the original dataset from 52 to 9 attributes similarly to the original dataset: sex, age, age_cat, juv_fel_count, juv_misd_count, juv_other_count, priors_count, race, c_charge_degree. The prediction variable is whether the individual will be rearrested in two years or not.

1. sex: binary sex.
2. age: numerical age.
3. age_cat: categorical (age $<$ 25, age $\geq$ 25 and age $<$ 45, age $\geq$ 45).
4. juv_fel_count: a continuous variable containing the number of juvenile felonies.

[6] `http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html`

[7] `https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm`

5. juv_misd_count: a continuous variable containing the number of juvenile misdemeanors.
6. juv_other_count: a continuous variable containing the number of prior juvenile convictions that are not considered either felonies or misdemeanors.
7. priors_count: a continuous variable containing the number of prior crimes committed.
8. race[8]: binary attribute (0 means 'white' and 1 means 'black').
9. c_charge_degree: Degree of the crime. It is either M (Misdemeanor), F (Felony), or O (not causing jail time).

This database presents a class imbalance problem for each group, as there are 1361 negative vs 1313 positive labels for black people whereas there are 839 negative vs 544 positive labels for white people. Typically, this translates into models that will tend to label black people as reoffenders whilst it will tend to label white people as not reofenders. This can be assessed though the false positive rate for each group. We will perform an exploratory analysis of this database within the practical sessions. In this lab assignment, the input variables of this dataset have been previously standarized in the `csv` version. The race variable is the last one in the `csv`.

Question [12]: Use the script you developed in question 9 for the training of this database, without doing the split and the standarization. Use the following range $C, \gamma \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$. Pay attention to the $accuracy$ value obtained for the generalization set. At the end, please, take a look to the optimal values obtained for the parameters.

Question [13]: Find out where the value of K for internal cross validation is specified and the range of values used for the parameters C and γ. How could you reduce the computational time needed to carry out the experiment? Try to set $K = 3$, $K = 5$ and $K = 10$ and compare, using a table, the computational times obtained and the results for the test set in terms of CCR.

3.2 *spam* classification dataset

One of the main fields where machine learning is successfully used is the automatic detection of spam on mail servers. As a significant fact, bear in mind that the mail you receive every day in your account at the University of Cordoba (UCO) is only 5% of the real mail you receive, so the remaining 95% is automatically detected and discarded by anti-spam filters. Most of these filters use binary classification techniques internally (spam mail, $y = 1$, or non-spam, $y = 0$) and the use of techniques based on SVMs is gaining more and more acceptance. In this section, we are going to explain the steps carried out to preprocess the text of an email and convert it into a feature vector able to be used as input in any classifier. Later, you will be asked to use the SVM approach to classify a real database.

```
> Anyone knows how much it costs to host a web portal ?
>
Well, it depends on how many visitors youre expecting. This can be
anywhere from less than 10 bucks a month to a couple of $100. You
should checkout http://www.rackspace.com/ or perhaps Amazon EC2 if
youre running something big..
To unsubscribe yourself from this mailing list, send an email to:
groupname-unsubscribe@egroups.com
```

Figure 6: Example of an email

Let's imagine an email whose body looks like the one in Figure 6. We can see how this email contains an URL, an email address (at the end), numbers and a quantity followed by a dollar

[8]Recommended reading: There's No Scientific Basis for Race–It's a Made-Up Label

symbol. Many emails will have these kinds of things (email addresses, URLs) but, surely, the concrete value (the same URL or the same email) will be different in each e-mail. Normally, for the concrete task of classification (whether it is spam or not), you are interested in the fact that the email contains an URL, an email or a number, but not their values. Therefore, the process we are going to follow "normalises" all the URLs so that they are treated in the same way, as well as all the numbers, all the email addresses, etc. To do this, every time an `http` address appears in our email, we will replace it with the reserved word `httpaddr` indicating that there was an address. This has been proved to increase the percentage of good classification in spam detection, because spammers almost always change the URL addresses randomly (being, in this way, more difficult to detect them).

An email database pertaining to the repo *SpamAssassin Public Corpus*[9] is provided. Each email of this database has received the following preprocessing:

1. All the mail has been rewritten in lowercase letter (i.e. "`InDICAtE`" will be treated as if it was written as "`Indicate`" or "`indicate`"). It is common for spammers to use this technique to avoid been detected.
2. Remove all the HTML code: all the emails will be processed as plain text, in such a way that all the HTML labels will be removed.
3. Normalise the URL: all the URL will be replaced by `httpaddr`.
4. Normalise the email addresses: all the email address will be replaced by `emailaddr`.
5. Normalise the numbers by replacing them by `number`.
6. Normalise the dollars: all the dollar symbols ($, highly associated with spam emails) will be replaced by `dollar`.
7. Word stemming: stemming a word is keeping its root. For the detection of spam is a good idea to use the root of the words instead of the complete words. For instance, the words "`discount`", "`discounts`", "`discounted`" and "`discounting`" will be replaced by "`discount`" . Stemmers are those programmes performing this process in an automatic way, using specific diand compare it to the one obtained in previous practices.ctionaries.
8. Remove anything that is not words, for example, punctuation marks or unusual symbols. In addition, all spaces, tabs or line breaks that follow each other are replaced by a single space character.

The result of all these steps on the previous email can be seen in Figure 7. With this email representation is much easier to extract characteristics that transform the email into a vector of numbers.

```
anyon know how much it cost to host a web portal well it depend on how mani visitor your
     expect thi can be anywher from less than number buck a month to a coupl of
    dollarnumb you should checkout httpaddr or perhap amazon ecnumb if your run someth
    big to unsubscrib yourself from thi mail list send an email to emailaddr
```

Figure 7: Example of a preprocessed email

After this step, we must choose which of the resulting words will be used to classify spam. This is called a vocabulary list. In this case, 1899 words have been selected, which are the ones that appear most frequently. These words are in the `vocab.txt` file and are also shown in Figure 8. These words are those that appeared at least 100 times in the total number of emails of the database.

[9] http://spamassassin.apache.org/publiccorpus/

```
1 aa
2 ab
3 abil
4 ...
5 916 know
6 ...
7 1898 zero
8 1899 zip
```

Figure 8: List of vocabulary

By using this vocabulary list, the email in Figure 7 is reduced to a set of indexes representing each of the words (assuming the word is in the vocabulary list, otherwise it is simply ignored as being a very rare word). The result of this transformation can be seen in Figure 9.

```
86 916 794 1077 883 370 1699 790 1822 1831 883 431 1171 794 1002 1893 1364 592 1676 238
    162 89 688 945 1663 1120 1062 1699 375 1162 479 1893 1510 799 1182 1237 810 1895
    1440 1547 181 1699 1758 1896 688 1676 992 961 1477 71 530 1699 531
```

Figure 9: Example of a preprocessed email in which the word indices have been extracted

From the spam detection point of view, we are highly interested in whether the word appears in the vocabulary list, but not its order. Therefore, the last step consists in transforming the previous indices vector in a binary-valued vector. In our case, we will have a vector $\mathbf{x} \in \mathbb{R}^{1899}$ and the i-th position will be 1 ($x_i = 1$) if the i-th word of the vocabulary appears in the email (otherwise, $x_i = 0$ if it does not appear). This vector will be the vector used for classifying whether there is a spam email or not. Note that using this representation, the number of times the words appear is not considered.

You have two data files where all this process has already been carried out. The training file contains 4000 emails, whereas the test file has 1000 emails. Both use the 1899 word vocabulary list contained in `vocab.txt`. Therefore, each pattern has 1899 binary values.

Question [14]: A **linear** SVM model with the values $C = 10^{-2}$, $C = 10^{-1}$, $C = 10^{0}$ and $C = 10^{1}$ must be trained. For this, use a script similar to the one used for question 9. Compare the results and stablish the best configuration.

Question [15]: For the best configuration, build the confusion matrix and establishes the misclassification emails. Check the input variables for the emails incorrectly classified and find out the reason behind it. Note that for each pattern, when $\mathbf{x}_i$ is equal to 1 it means that the i-th word in the vocabulary appears, at least once, in the email.

Question [16]: Train a non-linear SVM and compare the results obtained.

4 Deliverable files

The files to be submitted will be the following:

- Report of the practice in a `pdf` file answering to all the questions that have been raised in this script. Include the tables or images that you consider necessary to demonstrate that the exercises requested have been carried out. The report should include, at least, the following items:
  - Cover with the lab assignment number, its title, subject, degree, faculty department, university, academic year, name, DNI and email of the student

  - Index of the content with page numbers.
  - The answers to all the questions and exercises raised in this assignment.
  - Bibliographic references or any other material consulted in order to carry out the lab assignment different to the one provided by the lecturers (if any).

- A compressed file with all the scripts, data files and ouput files used to answer the questions.

Optionally, you can present both the report and the code together in a Jupyter notebook.