Universitá
Ca'Foscari
Venezia

Master course

in Computer Science

Artificial Intelligence: Knowledge Representation and Planning

-

Assignment 2

Studente
Francesco Bruno
Matricola 875812

Anno Accademico
2021 / 2022

Contents

Chapter 1

Introduction

1.1 Requirements

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

1. Perform SVM classification using linear, polynomial of degree 2, and RBF kernels over the TF/IDF representation.
   Can you transform the kernels to make use of angular information only (i.e., no length)? Are they still positive definite kernels?

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

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

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

   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.

3. 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.

P.S. you can use a library implementation for SVM, but do implement the Naive Bayes on your own. As for $k - NN$, you can use libraries if you want, but it might just be easier to do it on your own.

1.2 Classification Problem in Machine Learning

Identifying mails as spam ones can be considered a **classification problem**, a very common problem in machine learning.
The basic idea of classification is that starting from an input, a classifier is trained by correctly labelling each input to a possible output thanks to some features taken from the input objects, this allows to obtain a model where future unknown inputs will be correctly mapped to a label.
In our case the algorithm takes as input **feature vectors**, they are basically vectors that represent the object they are identifying by containing the features they own.
At the end of the algorithm execution each feature vector will be mapped to a class label, in our case labels are value 1 if the email is considered as spam, 0 otherwise.
In order to evaluate the performance of the classifier (the accuracy of the prediction), some indexes representing the classification accuracy are gained after each execution of the classifier.
This task will provide some results about different classifiers, showing their classification accuracy and comparing them in order to figure out which is the best classifier that should be used in order to have a good spam filter.

1.3 Discriminative classifier

The aim of a **discriminative classifier** is to learn the best boundary (*decision boundary*) between dataset classes so to correctly apply labels later on, in this task both *Support Vector Machine* and $k - NN$ classifiers belong to this class.

1.4 Generative classifier

The aim of a **generative classifier** is to learn the probability distribution of dataset classes so to correctly label new examples later on, usually classifiers that belong to this model are computationally expensive compared to discriminative ones. In this task the *Naive-Bayes* classifier takes part of this class of classifiers.

1.5 Dataset

A valid dataset is necessary to obtain a correct classifier, thus for this task the dataset used is `https://archive.ics.uci.edu/ml/datasets/spambase`, this one has been obtained by collecting 4601 emails and each one is characaterized by a list of attributes where are indicated if a certain word or character is used and how frequently it is.
In particular, citing the website above, the attributes are defined as:

- 48 continuous real [0,100] attributes of type word_freq_WORD = percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
- 6 continuous real [0,100] attributes of type char_freq_CHAR = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail
- 1 continuous real [1,...] attribute of type capital_run_length_average = average length of uninterrupted sequences of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_longest = length of longest uninterrupted sequence of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail
- 1 nominal {0,1} class attribute of type spam = denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

1.6 Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is a function used to estimate how much important is a word into a document or a collection of documents.
To better understand *TF-IDF* function some definitions are given, so tf_{ij} represents the number of occurences of the term i in the document j, divived by the number of terms of document j, in other words is the frequency of term i inside the document j, thus:

$$tf_{ij} = \frac{n_{i,j}}{|d_j|}$$

idf_i is the inverse document frequency, the importance/rilevance of the term i inside the collection of documents, called D.

$$idf_i = log_{10}\frac{|D|}{|\{d: i \in D\}|}$$

TF-IDF measure is then obtained merging the definitions above:

$$(tf - idf)_{ij} = tf_{ij} \times idf_i$$

The implementation of *TF-IDF* representation into the program is the following one:

```
    def tf_idf(features):
      ndoc = features.shape[0]
      idf = np.log10(ndoc/(features != 0).sum(0))
      return (features/100.0)*idf

```

Code 1.1: tf_idf method

This function is applied into each classifier analyzed to the first 54 attributes of each mail so to obtain a more consistent representation of data.

Chapter 2

Support Vector Machine Classifier

2.1 Theory

First model analyzed is the *Support Vector Machine* one, a supervised learning[1] model which core idea is to find an hyperplane that divides in a better way a dataset into two classes, thus it involves a binary classification such as spam filtering problem.
In order to find the best hyperplane, some definitions are needed. The closest points of a dataset to the hyperplane are called **support vectors**, the distance between the support vectors of two different classes and the hyperplane is called *margin* (see Figure 2.1).
At the beginning SVM try to find a **linearly separable** hyperplane that maximizes the margin but this isn't always possible because it could happen that the hyperplane doesn't exist, if this happens than the training data is transformed into an higher dimension where a linearly separable hyperplane

[1]Set of machine learning algorithms where dataset is already correctly labelled

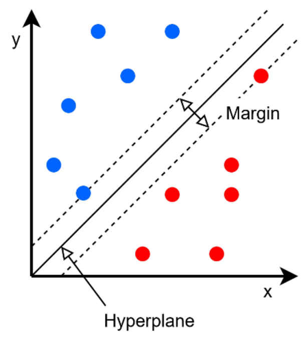

Figure 2.1: Hyperplane, margin and support vectors

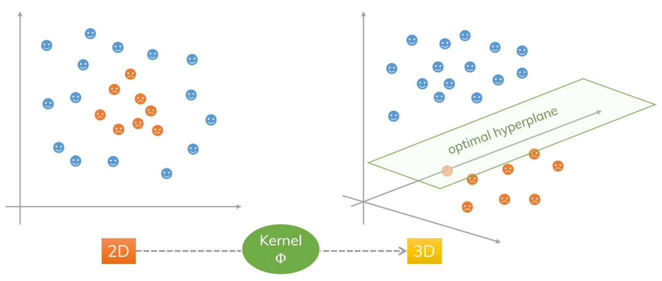

Figure 2.2: Kernel Concept

can be found, this can be done thanks to the **kernel** concept (see Figure 2.2).

Kernel functions and soft margin

Kernel fuctions as I said above take as input some data and transform it into a certain form where is possible to divide it into 2 classes through a linearly separable hyperplane.
There are different kernel functions, for this task are used only 3 of them:

1. **Linear**: $K(x_i, x_j) = x_i^T x_j$
2. **Polynomial of degree 2**: $K(x_i, x_j) = (1 + x_i^T x_j)^2$
3. **Gaussian Radial Basis Function**: $K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$

where the function $K : R^n \times R^n \rightarrow R$ is called **positive-definite kernel**, in particular this definition holds if and only if:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c j K(x_i, x_j) \geq 0$$

$\forall\ x_1, ..., x_n \in X$ and $n \in N$ and $c_1, ..., c_n \in R$.

SVM model is very rigid and can cause overfitting in the training set, so the constraint of maximizing the margin of the hyperplane should be relaxed. In order to do this, **soft margins** help us by tolerating few points to get misclassified, thus it creates a compromise between the finding of the perfect hyperplane with the maximum margin and the minimization of point misclassification.

Angular kernels

In order not to use the length of the vectors but only their angular information (as the requirements ask), the previous kernel function can be used but the input vectors must be normalized by dividing them by $\|x\|_2$, assuming $\alpha(x) \doteq \frac{x}{\|x\|_2}$ the function that handles vector normalization we'll have:

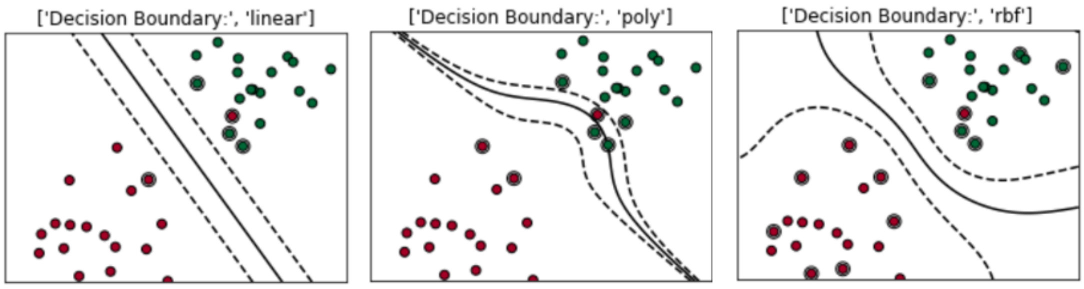

Figure 2.3: Kernel functions

1. **Linear**: $K(\alpha(x_i), \alpha(x_j)) = (\frac{x_i}{\|x_i\|_2})^T(\frac{x_j}{\|x_j\|_2}) \doteq \cos(x_i, x_j)$, now this kernel considers only the cosine of the angle between the two vectors x_i and x_j and not the length of them.
   This kernel is still a positive-definite one, then in order to demonstrate it we must prove that:

   $\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(x_i, x_j)$

   $= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(\alpha(x_i), \alpha(x_j))$

   $= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \sum_{k=1}^{m}(\frac{x_{ik}}{\|x_{ik}\|_2}) \cdot (\frac{x_{jk}}{\|x_{jk}\|_2})$

   $= \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m} c_i(\frac{x_{ik}}{\|x_{ik}\|_2}) \cdot c_j(\frac{x_{jk}}{\|x_{jk}\|_2})$

   $= \sum_{k=1}^{m}(\sum_{i=1}^{n} c_i(\frac{x_{ik}}{\|x_{ik}\|_2})) \cdot (\sum_{j=1}^{n} c_j(\frac{x_{jk}}{\|x_{jk}\|_2}))$

   $= \sum_{k=1}^{m}(\sum_{i=1}^{n} c_i(\frac{x_{ik}}{\|x_{ik}\|_2}))^2 \geq 0$

2. **Polynomial of degree 2**: $K(\alpha(x_i), \alpha(x_j)) = (1 + (\frac{x_i}{\|x_i\|_2})^T(\frac{x_j}{\|x_j\|_2}))^2$, this is always positive-definite since the result is calculated by a square;

3. **Gaussian Radial Basis Function**: $K(\alpha(x_i), \alpha(x_j)) = \exp(-\frac{\|\frac{x_i}{\|x_i\|_2} - \frac{x_j}{\|x_j\|_2}\|^2}{2\sigma^2})$, this is always positive-definite since the exp function doesn't return negative values;

2.2 Implementation

Support Vector Machine classifier has been implemented through *sklearn* and *numpy* libraries, the first one gives an already developed SVM classifier and

other instruments such as a method for using **cross validation**[2], the second one is used in order to simplify the usage of arrays.
Four methods have been built up, the most important are:

```
    def run(dataset):
      np.random.shuffle(dataset)
      classes = dataset[:, -1]
      features = dataset[:, :54]
      features = u.tf_idf(features)
      svm("linear", 3, features, classes)
      svm("poly", 2, features, classes)
      svm("rbf", 3, features, classes)
      features = normalize(features)
      svm("linear", 3, features, classes)
      svm("poly", 2, features, classes)
      svm("rbf", 3, features, classes)

```

Code 2.1: run method

run method (see Code 2.1) is the only one called from the main and it takes as input a dataset, from this set it filters all features and classes, performs a TF-IDF transformation over the data and only now it calls *svm* method for each type of kernel function used.
As requested from the professor, at a certain point of the method only angular information of a vector need to be considered, thus features vector is then normalized and *svm* method is called other 3 times (one for each type of kernel function), passing the new vector of features that is now normalized.
svm method (see Code 2.2) takes as input a type of kernel, a possible degree, a set of features with their classes, then it builds from these information a Support Vector Machine classifier and performs a cross validation passing the classifier built to the *cross_val_score* method. At the end a classification accuracy is obtained and results are printed.

```
    def svm(kernel, degree, features, classes):
      classifier = SVC(kernel=kernel, degree=degree)
      result = cross_val_score(classifier, features, classes,
     cv=10, n_jobs=-1)
      view(result, classifier.kernel)

```

Code 2.2: svm method

normalize method (see Code 2.3) is needed in order to normalize a vector, thus not to use the length of vectors but only their angular information, so it takes as input a vector of features and returns it normalized.

[2]Evaluation technique used for testing the performances of a machine learning model

```
def normalize(features):
  norms = np.sqrt((np.power(features+sys.float_info.epsilon
,2)).sum(axis=1, keepdims=True))
  return np.where(norms > 0.0, features / norms, 0.)

```

Code 2.3: normalize method

2.3 Results

By applying 10 way cross validation to the classifier and by considering the length of vectors, the following results have been obtained:

Classification accuracy				
Kernel	**Minimum**	**Maximum**	**Mean**	**Variance**
Linear	0.6043	0.6130	0.6087	$6.3483 \cdot 10^{-6}$
Poly	0.7852	0.8282	0.8111	0.0001
Rbf	0.9043	0.9369	0.9237	0.0001

As we can see best performances about the classification accuracy are observed when RBF kernel is used, with a mean value of 92.37%, on the other hand the worst ones are noticed with linear kernel, with a mean value of 60.87%. Let's now consider only the angle information, so features vector is now normalized. As we can see performances are drastically different.

Classification accuracy				
Kernel	**Minimum**	**Maximum**	**Mean**	**Variance**
Linear	0.8891	0.9456	0.9256	0.0003
Poly	0.9086	0.9565	0.9406	0.0001
Rbf	0.9108	0.9543	0.9404	0.0001

Best classifier is now the one with polynomial of degree 2 kernel, with a great increase of accuracy and it has now a mean value of 94.06%, RBF kernel remains almost the same with a slightly increase of accuracy and has now a mean value of 94.04%, last but not least linear kernel has observed a huge incrementation of effectiveness and it has now a mean accuracy of about 92%.
This sharply increase of performances of both linear and poly kernel when angular only information are used can be caused by the fact that those 2 kernel functions are heavily influenced by the length of vectors.
In conclusion, when angular information only is considered, all 3 kernel functions perform well with a mean accuracy value higher than 90%, otherwise RBF kernel is prefered to be used.

Chapter 3

Naive-Bayes Classifier

3.1 Theory

Like Support Vector Machine, *Naive-Bayes* is a supervised learning model and its peculiarity is that it solves problem through the usage of **Bayes theorem**.
This theorem can be summed up by the following equation:

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

where:

- x and y are events;
- $p(x)$ and $p(y)$ are the probabilities of observing events x and y without any previous condition given;
- $p(x) \neq 0$
- $p(x|y)$ is a conditional probability that express the probability of the event x to occur if the event y is already occured;
- $p(y|x)$ is a conditional probability that express the probability of the event y to occur if the event x is already occured;

In order to understand the task we can imagine $p(x)$ as the probability of observing a particular email and $p(y)$ is the probability of observing a particular class of email (spam or ham one), this can be expressed as

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

where α_k is the frequency of the class k (called *class-frequency*).
Then $p(y|x)$ is the probability of a label (I remember, 1 for spam and 0 for ham) to belong to a specific email, so it must be calculated both $p(y = 0|x)$ and $p(y = 1|x)$, thus depending on which one has higher value the label can

be associated to the email x.
Last but not least, in order to express $p(x|y)$ we assume that each feature is modelled with a Gaussian distribution, then:

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

3.2 Implementation

Requirements of the task expressly tell how not to use any already-implemented library classifier, so a "home-made" one has been developed starting from *numpy* and **BaseEstimator**[1], this extension has allowed the usage of *sklearn cross_val_score* method in order to performa cross validation.
Naive_Bayes_Classifier is the class name of the home-made classifier and it contains two methods, *fit* (see Code 3.1) and *score* (see Code 3.2).

```
  def fit(self,X,Y):
    features = X.copy()
    classes = Y.copy()
    self.spam = []
    self.no_spam = []
    for i in range(0,len(features)):
      if(classes[i] == 1):
        self.spam.append(features[i])
      if(classes[i] == 0):
        self.no_spam.append(features[i])
    self.spam = np.array(self.spam)
    self.no_spam = np.array(self.no_spam)
    total_length = len(self.spam)+len(self.no_spam)
    self.spam_prob = len(self.spam)/total_length
    self.no_spam_prob = len(self.no_spam)/total_length
    self.spam_mean = self.spam.mean(axis = 0)
    self.no_spam_mean = self.no_spam.mean(axis = 0)
    self.spam_variance = self.spam.var(axis = 0)
    self.no_spam_variance = self.no_spam.var(axis = 0)

```

Code 3.1: fit method

The first method (**fit**, see Code 3.1) takes as input a training set X with their classes Y, from this information it filters the spam and ham mails and it calculates their probabilities, mean and variance needed for the next method, in particular it computes $p(y = 0)$ and $p(y = 1)$.
The second method developed of the class, (**score**, see Code 3.2), takes as input a set of emails that needs to be classified, it predicts their classes and calculates the classification accuracy of the model, returning it at the end of the execution.
The inference of linking a mail to a specific class is made by modelling each

[1] "Base class for all estimators in scikit-learn", text taken from `https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html`

feature with the Gaussian distribution, then it calculates the productivity between all the features and finally multiplies it with the previous calculated probabilities $p(y = 0)$ or $p(y = 1)$ that have been computed in the *fit* method, obtaining a certain probability of a mail to belong to a specific class.
This process is made two times, one considering spam mail probability and the other one considering ham mail probability, thus at the end these two final probabilities obtained are compared and the mail will be associated to a class which probability has higher value.

```python
def score(self, X, Y):
  spam_prod_internal = np.power((2 * math.pi *  (self.spam_variance+sys.float_info.epsilon)),(-1/2)) * np.exp((-1./(2*self.spam_variance+sys.float_info.epsilon)) * np.power((X-self.spam_mean),2))

  no_spam_prod_internal = np.power((2 * math.pi * (self.no_spam_variance+sys.float_info.epsilon)),(-1/2)) * np.exp((-1./(2*self.no_spam_variance+sys.float_info.epsilon)) * np.power((X-self.no_spam_mean),2))

  spam_prod = spam_prod_internal.prod(axis=1)
  no_spam_prod = no_spam_prod_internal.prod(axis=1)
  spam_prob = spam_prod * self.spam_prob
  no_spam_prob = no_spam_prod * self.no_spam_prob
  labels = np.argmax([no_spam_prob,spam_prob], axis = 0)
  success = 0
  index = 0
  for i in labels:
    if i == Y[index]:
      success = success + 1
    index = index + 1
  return (success / len(labels))

```

Code 3.2: score method

Last method shown into this report (**run**, see Code 3.3) doesn't take part of *Naive_Bayes_Classifier* class and it's the only one that is called from the main. Through its usage a cross validation is performed (thanks to *cross_val_score* method), taking as input the classifier implemented (of class *Naive_Bayes_Classifier*), the features of each email (over TF-IDF representation) and their classes, then at the end of the execution the results are printed.

```
    def run(dataset):
      np.random.shuffle(dataset)
      classifier = Naive_Bayes_Classifier()
      features = dataset[:, :54]
      classes = dataset[:, -1]
      features = u.tf_idf(features)
      results = cross_val_score(classifier, features, classes,
  cv=10, n_jobs=-1)
      view(results)

```

Code 3.3: run method

3.3 Results

By applying 10 way cross validation to the classifier the following results have been obtained:

Classification accuracy			
Minimum	**Maximum**	**Mean**	**Variance**
0.7913	0.8369	0.8109	0.0002

As we can see results are not so astonishing, with only a mean accuracy of 81,09%.
This result can be attributed to the fact that a Naive-Bayes classifier is based on the assumption of a **strong indipendence** between features and as we can imagine on spam context this isn't always verified, thus results obtained are probably affected by this uncorrect assumption.

Chapter 4

K-Nearest Neighbour Classifier

4.1 Theory

This simple but efficient algorithm is based on the assumption that similar objects are close to each other, thus new data is classified comparing it with the already stored dataset and in particular is calculated the similarity (**distance**) between points. Algorithm works as follows:

1. Get k constant, dataset and the query object;
2. Prepare an empty collection;
3. For each object in the dataset:
   (a) Get the similarity between the dataset object and the query object;
   (b) Store the calculated similarity and the index of the dataset object inside the collection initilizated above;
4. Sort the collection in ascending order by checking the distance between objects;
5. Get the first k elements of the collection and their labels;
6. Return the most common label in those k objects;

There are different ways to calculate the distance between points, one of the most common method is the *Minkowski* distance, a generalization of the *Euclidean* and *Manhattan* ones. So, given:

$$X = (x_1, x_2, ..., xn) \text{ and } Y = (y_1, y_2, ..., y_n)$$

two points, the Minkowski distance of order p is:

$$D(X,Y) = \sum_{i=1}^{n}(|x_i - y_i|^p)^{\frac{1}{p}}$$

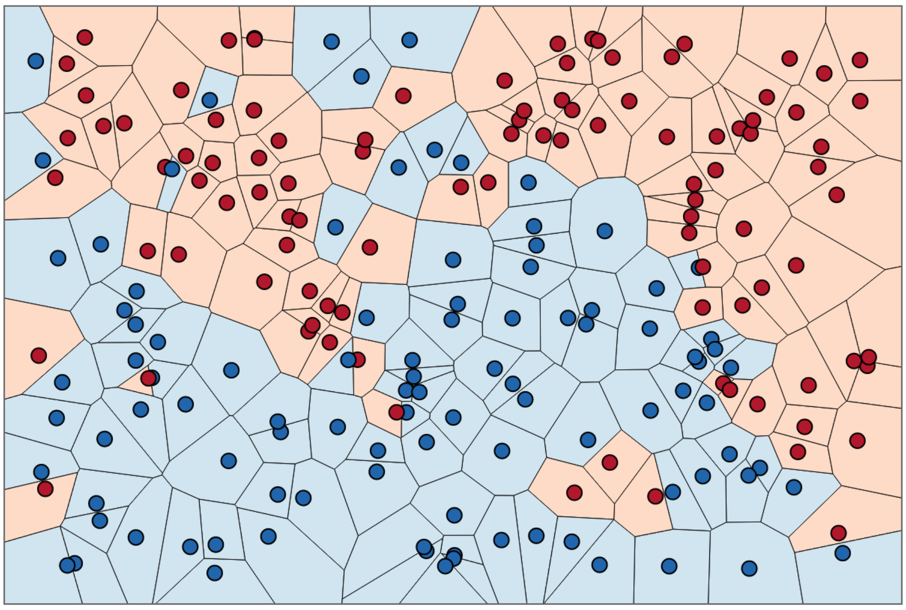

Figure 4.1: Points that are close to each other usually have the same label

In particular, when p is equal to 2 then is equivalent to use the Euclidean distance.
This algorithm is said to be **non-parametric** and **lazy learner** because it doesn't make any assumption on the underlying data distribution and doesn't immediately start learning when the training data is supplied (it's only stored) and all the computation is then performed at scoring time.

4.2 Implementation

By using *sklearn* and *numpy* libraries the implementation of an algorithm is very simple.
Two methods are used:

```
    def run(data,k):
      np.random.shuffle(data)
      classes = data[:, -1]
      features = data[:, :54]
      features = u.tf_idf(features)
      knn(k, features, classes)

```

Code 4.1: run method

```
    def knn(k,features, classes):
      classifier = KNeighboursClassifier(n_neighbors=k)
      result = cross_val_score(classifier, features, classes,
     cv=10, n_jobs=-1)
      view(results)

```

Code 4.2: knn method

run method (see Code 4.1) gets as input the dataset and the k constant value, it gets from the dataset the features and classes of each email, it applies TF-IDF function over the attributes of the mails and then calls the next method called **knn**, passing as input to this method the list of features, classes and k.
Once the *knn* method (see Code 4.2) is called, it creates an already implemented *K-Nearest Neighbors* classifier thanks to sklearn *KNeighborsClassifier*, passing it as input the value k. It's important to underline that the default value of order p, if absent from the parameters passed to the constructor, has value 2, thus is equivalent to use the euclidean distance.
Once the classifier is obtained, the *cross_val_score* function is called, at the end of its execution a classification accuracy of the classifier is gained and results will be printed through *view*, another method not shown that only prints the results obtained.

4.3 Results

Results obtained by using a 5-Nearest Neighbour classifier and 10 way cross validation are very promising:

Classification accuracy			
Minimum	**Maximum**	**Mean**	**Variance**
0.8956	0.9478	0.9263	0.0002

In fact, this type of classification works pretty well applied to this type of problem despite its simplicity, having an accuracy mean of 92.63%, also it has a very fast training phase and dataset is very good for this type of classifier since data is almost at the same scale.

Chapter 5

Conclusion

This assignment has required to analyze different classifiers in order to compare their performances.
First classifier analyzed is the Support Vector Machine one, by applying kernel trick the accuracy of this classifier has reach very high precision, in particular all of three kernel functions (linear, polynomial of degree 2 and RBF) have an accuracy over 90%, this happens only by not considering the length of the vector but only its angle, otherwise accuracy drop dramatically for linear kernel and loses about 10% from its previous value for polynomial of degree 2 kernel.
Second classifier seen is the Naive-Bayes one, its performances aren't bad but neither amazing, with a mean accuracy of about 81%, this is probably caused by the features indipendence assumption made from this model.
Last classifier analyzed is the $k - NN$ one, results of this simply classifier are good as the SVM ones when angular information only is considered so results have shown how discriminative classifiers perform well on this type of problem, in particular SVM with polynomial of degree 2 angular kernel has obtained the best results.

List of Figures