SPAM FILTER

MUHAMMAD TAJWAR
MATRICULATION No. 888394

April 2021

ARTIFICIAL INTELLIGENCE : KNOWLEDGE REPRESENTATION AND PLANNING

PROFESSOR ANDREA TORSELLO

Contents

1 Problem Introduction

This assignment discusses the implementation of a Spam filter. The goal is to write an algorithm for a spam filter using discriminative and generative classifiers. To this aim, we are using a spam base dataset which represents spam/ham messages through bag-of-words representations. The representations employ a dictionary of 48 highly discriminative words and 6 characters to complete the task. The first 54 features correspond to word/symbols frequencies which are essential, however, the rest of the features, i.e., 55, 56, 57, and 58 can be ignored. We denote the class label '1' for the spam and '0' for the ham.
On this dataset, we apply 3 classification algorithms, i.e. support vector machines (SVM), Naive Bayes, and K-NN. The detailed tasks are as follows:

1. Classify the data-set using SVM. Employ three different kernels over the TF/IDF representations, i.e. linear, polynomial of degree 2, and radial basis functions (RBF) kernels.
2. Classify through a Naive Bayes classifier for the continuous inputs and model each feature as a Gaussian distribution.
3. Perform classification with k-NN with k=5.

Our task is as follows: i) provide the code, ii) provide the models for the training set, iii) perform the 10-way cross-validations, finally, compare the results.

2 Background

2.1 Classification Problem

The classification problem is widely known in the subject of machine learning and artificial intelligence. In this report, we focus on a simple classification problem that classifies a given message as a spam or not. To this end, we construct a model and train it via a given dataset. We also divide the data-set into training and testing data. Using the training data we learn the features vectors for our model and later on, employ this trained model to assigning the labels for our testing data-set.In our case, we assign the label '0' if message is ham and '1' if it is a spam.
Classification problems can be described via mathematical equations. Therefore, let $X = (X_1, ..., X_m) \in \mathcal{X}$ is the data-set that we are going to predict and let $Y \in \mathcal{Y}$ is the target variable that classifies the class as '0' or '1'. Moreover, we have a learning function from a training set, i.e. $\phi(x) : \mathcal{X} \to \mathcal{Y}$ given as:

$$\phi(x) = g(f(x)) \ , \tag{1}$$

where the $f(x)$ is function that separates the class '0' from class '1'. Moreover, the function $g(f(x))$ assigns the label and defined as:

$$g(f(x)) = \begin{cases} 1 & \text{if the massage is spam } f(x) \geq \epsilon \ , \\ 0 & \text{if its is a ham } \ . \end{cases}$$

2.2 Discriminative and Generative Classifiers

There are mainly two types of machine learning problems, i.e., **Supervised Learning** and **Unsupervised Learning**. In supervised learning, we have both labelled training and labelled testing sets. On the other hand, in unsupervised learning, we have unlabelled data and algorithms that try to learn and predict without labels. In this report, we focus on the supervised learning problem.
Classifiers are divided into two categories: i) **Discriminative**, and ii) **Generative**. We define them using the probabilities notations $p(x), p(y)$ and the conditional probabilities $p(x|y)$ and $p(y|x)$. The discriminative classifiers work on the posterior probability $p(y|x)$ which simply maps the input x to the y labels, whereas the generative classifiers mainly focus on the joint probability. They learn a model from the joint probability $p(x, y)$ and apply it to the conditional probability $p(x|y)$.

2.3 Support Vector Machine

SVM is the most widely used classifier in the machine learning. It is a discriminative classifier that works on principles of supervised learning. It is associated with the learning models that analyze the data for classification and regression analysis and is based on the statistical framework and VC theory.
Given a training set that is marked/labelled as two categories, the SVM trains a model and assigns labels to each class. The SVM works on linear and non-linear classifications. SVM classifies the two classes by sketching a hyperplane with the following equation:

$$w^T x - b = 0 \ . \tag{2}$$

The equation (**??**) is linear and used for the linear classification, however, in the case of the non-linear classification, we employ kernel tricks. There are several types of the kernel available but in this report we will consider only linear, polynomial kernel with degree 2, and Gaussian Radial kernels.

2.3.1 Linear Kernel

Linear kernel is mostly used when the data is linearly separable and can be separated by single line or hyperplane. Mathematically, it is given as $K(x_i, x_j) = x_i^T x_j$. For an illustration see the diagram below:

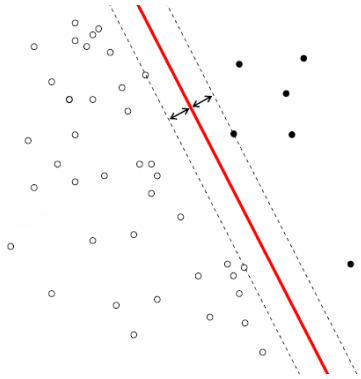

Figure 1: SVM(Linear Kernel)

2.3.2 Polynomial Kernel

Polynomial kernels are used in learning of the non-linear models and it represents the similarities of training samples over the polynomials. It is given as $K(x_i, x_j) = (1 + x_i^T x_j)^2$, where the square represents the polynomial with degree 2. See the figure below:

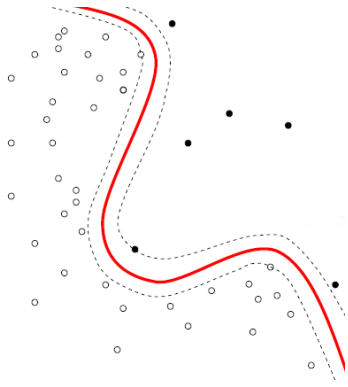

Figure 2: SVM(Polynomial Kernel)

2.3.3 Gaussian Radial Basis Function

Gaussian radial basis function also known as RBF kernel is also a very popular in machine learning community. The term radial basis function means it forms circles to separate the classes and it follows the function $K(x_i, x_j) = \exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$, where $||x_i - x_j||^2$ is a squared euclidean distance of two feature vectors and σ is the free parameter. See the figure below:

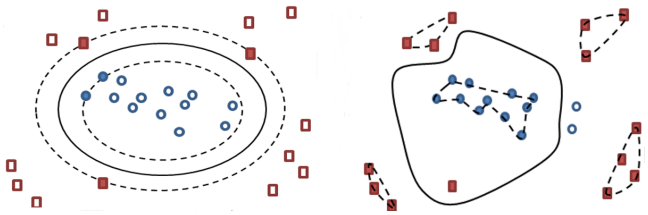

Figure 3: SVM(RBF Kernel)

2.4 Naive Bayes

There are other kinds of classifiers that are related to the probabilistic classifiers and naive Bayes is one of them. The main working principle behind it is Bayes theorem. This model coupled with the kernel density may achieve a higher level of accuracy. These type of classifiers are highly scalable and require several variables of linear parameter features and predictors in a learning problem.
There are two kind of Bayes classifiers: i) simple Bayes, and ii) independence Bayes. Both employ naive bayes models in the background. In this assignment, we mainly focus on Gaussian Distribution for our model given as:

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

where 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, respectively.

2.5 K Nearest Neighbour

K nearest neighbor also known as K-NN is used for data regression and data classification and it uses the non-parametric classification method. It takes k values closest to the training examples in the dataset as inputs and gives an output. However, the output depends on the method we choose, for instance, it can be for regression or classification tasks. It this assignment, we only discuss the K-NN for the classification.
In K-NN classification, the parameter k decides the number of the neighbours for the plurality vote and the object will assign a class with higest votes, for example, if k=1 then it will be assigned to a class that is closest to single nearest neighbor.

3 Support Vector Machine

An SVM learns a model from a given training set and predicts the class of an object from the set $\{1,-1\}$. An SVM linear classifier can be defined as:

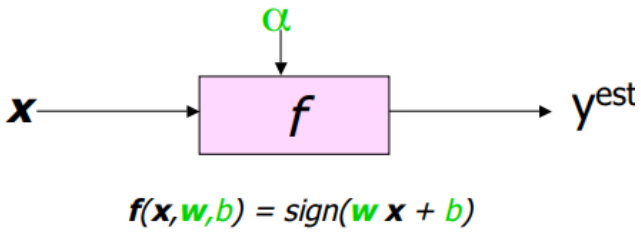

Figure 4: SVM

where $wx + b$ represents the equation of the hyperplane that separates the two classes and the function f is a decision function that predicts the class. In this assignment, the SVM will predict '1' if it spams that is $wx + b \geq 0$ and predict '-1' if it is a ham. We can define the probability of classification as:

$$p(y = 1|x; w, b) \ . \tag{3}$$

The higher values of $wx + b$ will also increase the probability $p(y = 1|x; w, b)$, therefore, the higher chances that label would be '1' and vice versa.

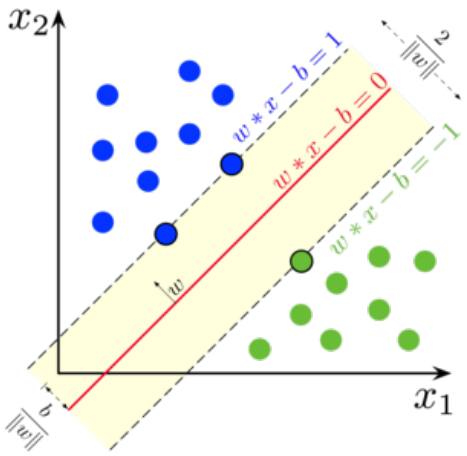

Figure 5: Classification

The Figure 5 demonstrates how the classification works. We have 2 classes colored as blue and green. The classifier will label blue as 1 and green as -1 and red line represents the hyperplane which is the decision boundary.

One problem that must be taken into consideration is choosing the right hyperplane as we can see below:

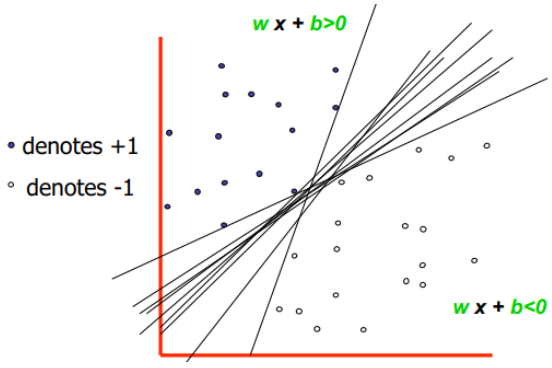

Figure 6: Infinite Hyperplane

As demonstrated in the Figure 6 there are an infinite number of a hyperplane that can classify the two classes but each hyperplane has its own consequences. Each of them can have different results and different performances and it can also affect the strength of the classifier.
Therefore, in order to choose the best hyperplane, we have to understand what is functional margin and what is the geometric margin?

3.1 Functional Margin

Functional margin is used for the verification of a point if it completely classified or not. To this aim, we calculate the hyperplane margin and compare it with the prediction, since our dataset is already labeled, it is defined as:

$$\hat{\gamma} = y_i(w^T x + b) \ . \tag{4}$$

When we put $y_i = 1$ to maintain the larger functional margin there exists a large positive number for $w^T + b$ and vice versa for the $y_i = -1$ instead we need a large negative number so where $\hat{\gamma}_i$ is positive the SVM will predict the correct class for all the values. On the other hand, if we want to maximize the functional margin, we have to scale w and b arbitrarily since the classification depends upon the $h_{w,b}$ sign, not on it's magnitude value.

3.2 Geometric margin

The geometric margin represents the euclidean distance from decision boundary to a data point. In the Figure 7, consider the data point "A" and the margin $\hat{\gamma}_i$ which represents the scalar length from the point x_i to the hyperplane. Moreover, define $\frac{w}{|w|}$ as the unit direction which is orthogonal to the hyperplane, i.e., the shortest path from "A" to hyperplane "B" defined as:

$$\vec{A} - \gamma_i \left(\frac{\vec{w}}{|\vec{w}|} \right) \ . \tag{5}$$

In other words:

$$\vec{A} - margin = \vec{B} \ . \tag{6}$$

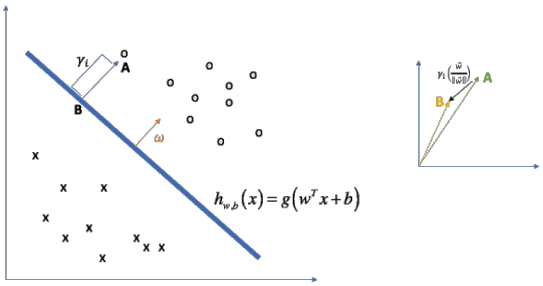

Figure 7: Geometric Margin

The scalar distance from point B to A is denoted as $\hat{\gamma}_i$ and then multiplied by the vector that is orthogonal to the hyperplane $\frac{w}{|w|}$. The decision boundary is that line which classifies the point as 1 or -1, therefore, the points that lies on the hyperplane must satisfy this equation $w^T x + b = 0$. Thus, the point B must satisfy

$$w^T x_B + b = 0 \ . \tag{7}$$

This point x_B on the decision boundary can be computed as:

$$x_B = x^i - \gamma_i \left(\frac{\vec{w}}{|\vec{w}|} \right) . \tag{8}$$

Combining equation (7) (8) we get

$$w^T \left(x^i - \gamma_i \left(\frac{w}{|w|} \right) \right) + b = 0 , \tag{9}$$

by solving this equation we get the geometric margin γ_i as follows:

$$\gamma_i = \frac{w^T x^i + b}{|w|} = \left(\frac{w}{|w|} \right)^T x^i + \frac{b}{|w|} \tag{10}$$

In the case for both classes y=1 and y=-1, we need small changes to keep the margin positive

$$\gamma_i = y_i \left(\left(\frac{w}{|w|} \right)^T x_i + \frac{b}{|w|} \right) . \tag{11}$$

If we compare the two margins, i.e., functional and geometric margins. We can see that they are same except w and b are extended by $\frac{1}{|w|}$. Therefore, we need smaller margins for the hyperplane, i.e

$$\gamma = \min_{i=1,..,m} \gamma_i . \tag{12}$$

Support Vector Machine does not only classify the two classes but also have to solve an optimization problem given below :

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

The constraint $||w|| = 1$ shows that the geometric margin is equal to the functional margin and it can be written as:

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

Due to the non convex constraint $||w|| = 1$ the optimization problem for the geometric margin turns into the functional margin, hence, for $\hat{\gamma} = 1$ the minimization of $||w||^2$ is given as:

$$\begin{aligned} &\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2 \\ &s.t \quad y^{(i)}(w^T x^{(i)} + b) \geq \gamma \qquad i = 1, ..., m . \end{aligned}$$

Another way to optimize the problem is using the Lagrange function with Lagrange multipliers $\lambda_1, ..., \lambda_m$. Using this function we can express the new optimization problem as:

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

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

3.3 Kernel Trick

Previously, we were using the SVM in a linear way, i.e., we worked on the data which is linearly separable. Now, the question is what if the data is not linear ? how we can separate the data ? The answer to both questions is, by using kernel tricks. An SVM uses different kernels to separate different kinds of data. Kernel trick allows the SVM to use mapping transformations, i.e., by taking data points and maping them into new space, probably into higher dimensions and this function is defined as:

$$K(x, z) = \phi(x)^T \phi(z) \; . \tag{13}$$

It is basically a similarity measure between the predicted mapping x and z. In order to be a valid kernel there is a restriction that function K must satisfy that is:

$$K(x, z) = \phi(x)^T \phi(z), \qquad \forall x, z \in S \; . \tag{14}$$

In this case we also have to introduce the positivity of the kernel property. The function K is said to be positive definite kernel if it satisfies the following equation

$$K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

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

Some of the positive definite kernels we used in this assignments are:

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

3.4 Angular Kernels

According to the Task 1 of this assignment we have to show that whether these kernel can be converted into the angular kernels or not? The aforementioned kernels are affected by only the length, i.e., they have the same directions but different lengths. If we are considering the direction of each vector we have to normalize this vector such that at the end $||x||_2 = 1$:

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

From these equations, we can conclude that length does not matter but only thing to be considered is cosine angle between the vectors x_i, x_j. In the following we prove that they are positive kernels as well. See the proof below:

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

This concludes the proof for the linear kernel. For the other two kernels we can write the kernels as:

$$K_p(x_i, x_j) = \phi_p(x_i)^T \phi_p(x_j) \qquad \text{Polynomial}$$
$$K_{rb}(x_i, x_j) = \phi_{rb}(x_i)^T \phi_{rb}(x_j) \qquad \text{Radial}$$

We can map these kernels basis on the angular information as:

$$\phi_p^{'}(x) = \phi_p\left(\frac{x}{||x||}\right) \qquad \phi_{bf}^{'}(x) = \phi_{bf}\left(\frac{x}{||x||}\right)$$

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

$$K_{bf}^{'}(x_i, x_j) = \phi_{bf}^{'}(x_i)^T \phi_{bf}^{'}(x_j) = \phi_{bf}\left(\frac{x_i}{||x_i||}\right)^T \phi_{bf}\left(\frac{x_j}{||x_j||}\right) = K_{bf}\left(\frac{x_i}{||x_i||}, \frac{x_j}{||x_j||}\right) \geq 0$$

Hence, it is proved that the K_p and K_{bf} kernels are positive definite kernels are not affected by the length of vectors.

3.5 Coding and Accuracy results of SVM

In this modern era many developers have already implemented the support vector machine, therefore, we use the python library sklearn for the given data

set with build-in kernels. Before learning the model we transform the data in TF-IDF formation which is given below:

$$TFD(i,j) = tf_{i,j} \times idf_i$$
$$idf_i = ln \frac{|D|}{|d : i \in d|} .$$

```
#transforming it to the TFIDF representation
def tfidf(mail):
    ndoc = mail.shape[0]
    idf = np.log10(ndoc / (mail != 0).sum(0))
    return mail / 100.0 * idf
```

After reading the data file and convert it in TFIDF, we pass it to the sklearn SVM functions with the three kernels.

```
#Taking the required classes and prediction class
Y = mail[:, 57] # classes
X = mail[:, :54] # values

X = tfidf(X)


#Classify using Linear Kernel and measuring the accuracy
#also doing the cross validation for 10 times
svm_linear = SVC(kernel="linear", C = 1.0)
acc_linear = cross_val_score(svm_linear, X, Y, cv = 10, n_jobs= 8)
print("Min Accuracy Linear Kernel: " + str(acc_linear.min()) + "\n"
    )
print("Max Accuracy Linear Kernel: " + str(acc_linear.max()) + "\n"
    )
print("================================")


#Classify using Polynomial Kernel with degree 2 and measuring the
    accuracy
#also doing the cross validation for 10 times
svm_poly2 = SVC(kernel="poly", degree = 2, C = 1.0)
acc_poly2 = cross_val_score(svm_poly2, X, Y, cv = 10, n_jobs= 8)

print("Min Accuracy Polynomial  Kernel: " + str(acc_poly2.min()) +
    "\n")
print("Max Accuracy Polynomial Kernel: " + str(acc_poly2.max()) + "
    \n")
print("================================")


#Classify using Gaussian Radial Basis Function Kernel and measuring
     the accuracy
#also doing the cross validation for 10 times
svm_rbf = SVC(kernel="rbf", C = 1.0)
acc_rbf = cross_val_score(svm_rbf, X, Y, cv = 10, n_jobs= 8)
print("Min Accuracy RBF Kernel: " + str(acc_rbf.min()) + "\n")
print("Max Accuracy RBF Kernel: " + str(acc_rbf.max()) + "\n")
print("================================")
```

As for the angular kernels, the functions will remain same but we only have to convert them in the angular form and give them as an argument to the three models. If we look at the classifier functions we can see an argument C which is the learning rate. If the value of C would be large it gives the solution with less errors but with a smaller margin and vise versa.

As the other part of the task we perform the 10-way cross-validation over the models. Basically, it is a technique that is used to measure the excellence of the models by iterating over the data and splitting them into the training set and the test set. Usually, we perform this step in order to avoid overfitting. In our case, we iterate 10 rounds over the dataset.
Following results are obtained with the three kernels:

Kernels	Min Accuracy	Max Accuracy
Linear Kernel	0.60520	0.61086
Polynomial 2 Kernel	0.78524	0.85217
RBF Kernel	0.90434	0.95000

After normalizing the whole dataset for the angular kernels, we performed the same tasks yielding the better results:

Kernels	Min Accuracy	Max Accuracy
Linear Kernel	0.90434	0.95652
Polynomial 2 Kernel	0.92608	0.95652
RBF Kernel	0.92391	0.95869

The tables show the increase in accuracy by using the angular kernels. There is a significant increase in standard linear kernel as compared to angular linear kernel. Moreover, for the polynomial kernel with degree 2 there is reasonable increase in the accuracy as well. Finally, for the RBF kernel we still observe a slight increase.
One of the main benefits of using SVM is that it also considers **Support Vectors**. Support vectors are the points that lie very close to the decision boundary or we can say they are points with the min geometric margins. These points are across the boundary that are miss-classified, some of the examples are:

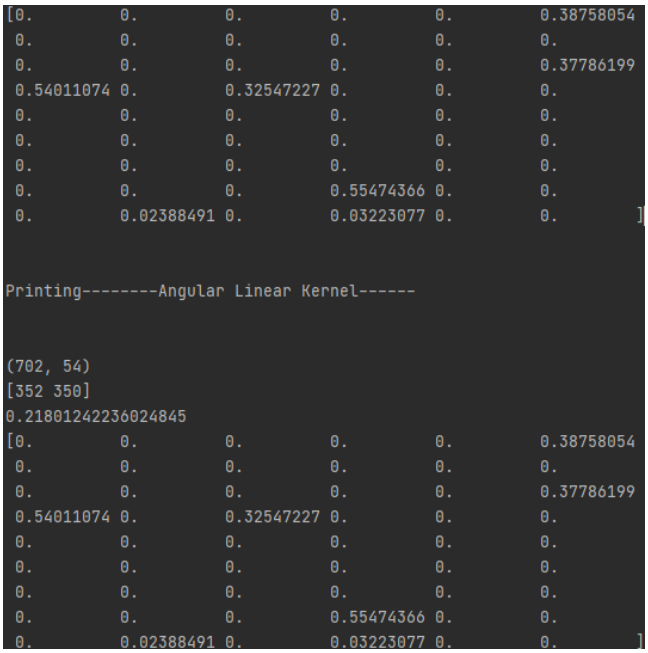

Figure 8: Support Vectors

Talking about the **Accuracy**, we have seen that our model is accurate to 0.92857 with the 3220 training samples which is also coherent to the number of support vectors. The expected cross validation error can be calculated as:

$$Expected\ \ Cross\ \ Validation\ \ Error \leq \frac{E[No.\ \ support\ \ vectors]}{No.\ \ of\ \ Training\ \ samples}$$

$$0.082450 \leq \frac{702}{3220} = 0.2180\ .$$

With the standard linear kernel the accuracy we achieve is 0.60403 with 2560 supporting vectors. In this case expected cross validation error is:

$$0.2180 \leq \frac{2560}{3220} = 0.7950$$

Therefore, with the angular kernels we have more support vectors but less accuracy.

4 Naive Bayes

Naive Bayes classifier is a generative classifier which works with the probabilities. It tries to learn the class's conditional density which is $p(x|y)$ for all the values of y and also learns class prior $p(y)$. This classifier works on the Bayes theorem stated as:

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)p(y)}{\sum_{k \in C} P(x|k)p(k)} \ . \tag{15}$$

- $p(y)$: probability of a hypothesis for being true, it is also called prior probability of y.
- $p(x)$: Data probability.
- $p(y|x)$: probability of hypothesis y given the data x, it is called as posterior probability.
- $p(x|y)$: probability of data x given the hypothesis y true.

Naive bayes classifier also assigns the labels by maximizing the probability $p(y|x)$ formally defined as:

$$f(x) = \arg\max_{y} \ \ p(y|x) = \arg\max_{y} \ \ \frac{p(x|y)p(x)}{p(x)} \ , \tag{16}$$

since the $p(x)$ is not depended on the class, we can rewrite the function as:

$$f(x) = \arg\max_{y} \ \ p(x|y)p(y) \ . \tag{17}$$

Since we are working on spam filtration, in this case $p(x)$ would be the probability of mails $x = \{x_1, x_2, ..., x_m\}$ where the x_m is frequency of word m and C is the set of labels {1,0}.
If we have unbiased estimator and with the correct assumptions the probability $p(y)$ can be written as:

$$p(y = k) = a_k, \quad \forall k \in C \ , \tag{18}$$

where the a_k is frequency of the class k.
While using the naive Bayes we assume that all x_i are conditionally independent given to y, therefore, the composition of the words in a given class is independent.

4.1 Conditional Independence

Given that X is conditionally independent of Y and Z only if:

$$P(X = x_i|Y = y_i, Z = z_k) = P(X = x_i|Z = z_k) \quad \forall i, j, k \ , \tag{19}$$

resulting in

$$p(x|y) = p(x_1, x_2, ..., x_m|y) = \prod_{i=1}^{m} p(x_i|y) \ . \tag{20}$$

We are using the Gaussian distribution model to train our dataset and we assume $p(x|y)$ to be multivariate. It is mandatory to assign a certain and known distribution to $p(x_i|y)$, thus, we have:

$$
\begin{aligned}
&x|y=0 \sim N(\mu_0, \Sigma_0) \\
&x|y=1 \sim N(\mu_1, \Sigma_1) \\
&p(x|y=0) = (2\pi|\Sigma_0|^2)^{-\frac{1}{2}} exp\left\{ -\frac{1}{2}(x-\mu_0)^T|\Sigma_0|^{-1}(x-\mu_0) \right\} \\
&p(x|y=1) = (2\pi|\Sigma_1|^2)^{-\frac{1}{2}} exp\left\{ -\frac{1}{2}(x-\mu_1)^T|\Sigma_1|^{-1}(x-\mu_1) \right\}.
\end{aligned}
$$

For the max probability estimator for μ_0,μ_1 and Σ, we have:

$$
\begin{aligned}
&p(y=k) = \frac{1}{n}\sum_{i=1}^{n} 1\{\ y^{(i)}=k\} \ = a_k \\
&\mu_0 = \frac{\sum_{i=0}^{n} 1\{\ y^{(i)}=1 x^{(i)}\}}{\sum_{i=0}^{n} 1\{\ y^{(i)}=1\}} \\
&\mu_1 = \frac{\sum_{i=1}^{n} 1\{\ y^{(i)}=1 x^{(i)}\}}{\sum_{i=1}^{n} 1\{\ y^{(i)}=1\}} \\
&\Sigma_0 = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)}-\mu_0)(x^{(i)}-\mu_0)^T \\
&\Sigma_1 = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)}-\mu_1)(x^{(i)}-\mu_1)^T \ .
\end{aligned}
$$

The mean and variance of the two classes can be different, therefore, no assumptions are made. We can build our naive Bayes classifier as follows:

```
def Prediction(x)
    if p(y=1|x) > p(y=0|x) then
        return 1
    otherwise
        return 0
```

Basic property of the naive Bayes classifier is to assign labels to maximize the probability $p(y=k|x)$ by taking cheap computation such as $p(y=0), p(y=1), \mu_0, \mu_1, \Sigma_0$, and Σ_1.

4.2 Probability Distribution

In the naive Bayes classifier we are using the Gaussian distribution which is a continuous probability distribution for a real random variable. The probability distribution is given as:

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

where α_k is the frequency of class k, and μ_{ki}, σ_{ki}^2 are the means and variances demonstrated as below:

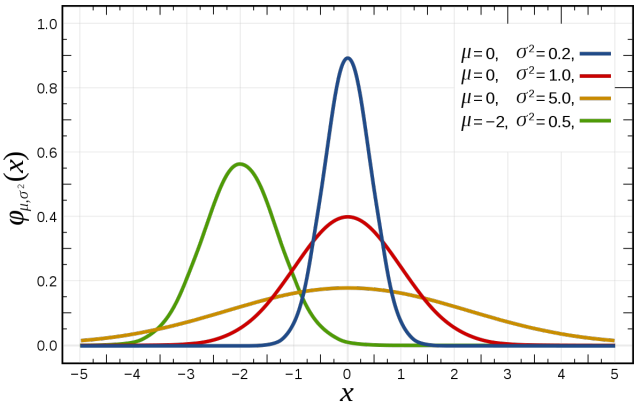

Figure 9: Normal Distribution

We have to prove the assumption that the $p(x|y)$ is a multivariate Gaussian with the two normality tests, i.e., **Mardia's** and **Henze-Zirkler's** tests implemented in R.

4.2.1 Mardia's Test

This normality test is based on multivariate normality measures known as **skewness** and **kurtosis**. Skewness tells the multivariate normal distribution for null hypothesis computed as $\frac{n}{6}skew \sim x^2(\frac{k(k+1)(k+2)}{6})$ and whereas the Kurtosis test measures the tailness of the curves computed as $[kurt - k(k+2)]\sqrt{\frac{n}{8k(k+2)}} \sim N(0,1)$.

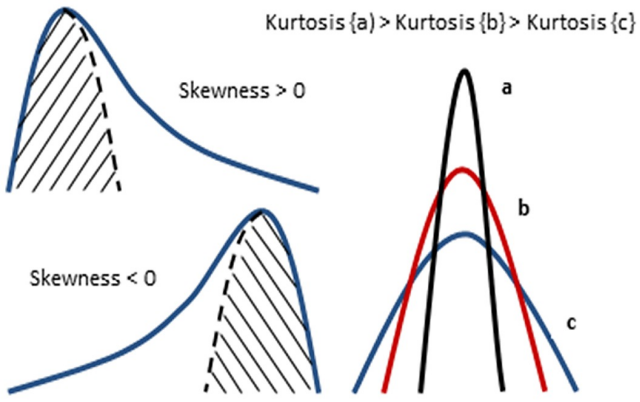

Figure 10: Skewness & Kurtosis

4.2.2 Henze-Zirkler's Test

This test is based on measurement of the functional distance which is non negative between 2 distribution functions.
These two tests show the following results when applied on the data:

```
             Test        Statistic p value Result
1 Mardia Skewness 4158941.59531139       0     NO
2 Mardia Kurtosis 4395.78574614606       0     NO
3             MVN             <NA>    <NA>     NO
> resHam <- mvn(data = spam_nz, mvnTest = "mardia")
> resHam$multivariateNormality
             Test        Statistic p value Result
1 Mardia Skewness 4158941.59531139       0     NO
2 Mardia Kurtosis 4395.78574614606       0     NO
3             MVN             <NA>    <NA>     NO
> resSpam <- mvn(data = spam_nz, mvnTest = "hz")
> resSpam$multivariateNormality
           Test       HZ p value MVN
1 Henze-Zirkler 9.756951       0  NO
> resHam <- mvn(data = spam_nz, mvnTest = "hz")
> resHam$multivariateNormality
           Test       HZ p value MVN
1 Henze-Zirkler 9.756951       0  NO
```

Figure 11: Mardia's & Henze-Zirkler's Test

These results show that the p-values are 0 which means that the hypothesis is rejected for both the class in the Mardia's test, whereas, the "NO" in the Henze-Zirkler's test also refers that the hypothesis is rejected.

Naive Bayes classifier is a learning algorithm that assigns labels and predict the value. We have to consider the **Model parameters**, in our case, we pass the algorithm 70% data for training and 30% data for learning. This yields the following results:

$$p(spam) = 0.39968$$
$$p(ham) = 0.60031\ .$$

The corresponding values of means and variance of ham and spam are given as:

```
mu spam: [1.540e-01 1.580e-01 4.070e-01 1.410e-01 5.150e-01 1.810e-01 2.820e-01
 1.960e-01 1.670e-01 3.360e-01 1.220e-01 5.600e-01 1.410e-01 8.400e-02
 1.140e-01 5.370e-01 2.910e-01 3.200e-01 2.261e+00 2.050e-01 1.389e+00
 2.540e-01 2.570e-01 2.170e-01 2.000e-02 1.100e-02 1.000e-03 2.500e-02
 1.000e-03 8.000e-03 1.000e-03 0.000e+00 1.300e-02 2.000e-03 7.000e-03
 3.000e-02 3.700e-02 5.000e-03 1.200e-02 3.900e-02 0.000e+00 3.000e-03
 9.000e-03 8.000e-03 1.180e-01 1.900e-02 2.000e-03 2.000e-03 2.000e-02
 1.040e-01 9.000e-03 5.000e-01 1.720e-01 6.800e-02]
```

Figure 12: Mean value with class spam μ_1

```
var spam: [1.050e-01 1.160e-01 2.360e-01 4.099e+00 4.980e-01 1.030e-01 3.120e-01
 2.850e-01 1.250e-01 3.530e-01 6.700e-02 4.400e-01 9.400e-02 1.100e-01
 1.560e-01 1.133e+00 4.270e-01 4.410e-01 2.465e+00 6.910e-01 1.525e+00
 2.247e+00 2.810e-01 3.920e-01 3.400e-02 1.200e-02 0.000e+00 1.300e-01
 0.000e+00 1.500e-02 0.000e+00 0.000e+00 1.000e-02 2.000e-03 5.000e-03
 2.300e-02 5.400e-02 3.000e-03 6.000e-03 2.600e-02 0.000e+00 1.000e-03
 3.000e-03 5.000e-03 1.010e-01 2.500e-02 0.000e+00 1.000e-03 8.000e-03
 3.400e-02 2.000e-03 4.890e-01 1.310e-01 1.890e-01]
```

Figure 13: Variance with class spam Σ_1

```
mu ham: [7.300e-02 2.100e-01 2.030e-01 1.000e-03 1.910e-01 4.500e-02 9.000e-03
 3.400e-02 4.300e-02 1.740e-01 2.100e-02 5.410e-01 5.900e-02 4.400e-02
 9.000e-03 8.400e-02 4.600e-02 1.010e-01 1.275e+00 7.000e-03 4.300e-01
 4.300e-02 7.000e-03 1.500e-02 9.180e-01 4.480e-01 1.326e+00 1.940e-01
 1.870e-01 1.720e-01 1.080e-01 8.400e-02 1.580e-01 8.400e-02 1.750e-01
 1.430e-01 1.940e-01 2.200e-02 1.170e-01 9.300e-02 6.900e-02 2.290e-01
 6.600e-02 1.380e-01 3.910e-01 2.580e-01 9.000e-03 4.500e-02 4.800e-02
 1.580e-01 2.200e-02 1.000e-01 1.200e-02 2.400e-02]
```

Figure 14: Mean value with class ham μ_0

```
var ham: [9.300e-02 2.271e+00 2.520e-01 1.000e-03 4.160e-01 4.400e-02 1.200e-02
 5.100e-02 5.000e-02 4.830e-01 2.100e-02 9.460e-01 6.900e-02 1.320e-01
 1.000e-02 5.060e-01 4.400e-02 1.690e-01 2.948e+00 8.000e-03 1.017e+00
 3.680e-01 5.000e-03 5.200e-02 4.468e+00 1.227e+00 1.928e+01 4.060e-01
 7.010e-01 3.480e-01 2.830e-01 1.930e-01 5.490e-01 1.930e-01 5.260e-01
 2.620e-01 2.390e-01 1.050e-01 2.840e-01 2.080e-01 1.980e-01 1.012e+00
 6.400e-02 7.310e-01 1.451e+00 1.020e+00 1.100e-02 1.020e-01 8.300e-02
 7.300e-02 1.400e-02 6.450e-01 6.000e-03 8.100e-02]
```

Figure 15: Variance values with class ham Σ_1

4.3 Implementation and Results

Naive Bayes classifier can be implemented in several ways. However, we can use the libraries or we can implement it on our own. In this assignment, we implement it by using the Gaussian distribution. The code implemented is based on the following formula:

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

```
class NaiveBayesClassifier(BaseEstimator):

    def score(self, X, Y):
        p_x_spam_i = (2 * np.pi * self.var_spam) ** (-1. / 2) * np.
    exp(
            -1. / (2 * self.var_spam) * np.power(X - self.mu_spam,
    2))
        p_x_ham_i = (2 * np.pi * self.var_ham) ** (-1. / 2) * np.
    exp(
            -1. / (2 * self.var_ham) * np.power(X - self.mu_ham, 2)
    )

        p_x_spam = np.prod(p_x_spam_i, axis=1)
        p_x_ham = np.prod(p_x_ham_i, axis=1)

        p_spam_x = p_x_spam * self.p_spam
        p_ham_x = p_x_ham * self.p_ham

        predicted_labels = np.argmax([p_ham_x, p_spam_x], axis=0)
        return np.mean(predicted_labels == Y)

    def fit(self, X, Y):
        #specifying labels for spam and ham
        self.spam = X[Y == 1, :54]
        self.ham = X[Y == 0, :54]

        self.N = float(self.spam.shape[0] + self.ham.shape[0])
        self.k_spam = self.spam.shape[0]  # frequency of spam
        self.k_ham = self.ham.shape[0]  # frequency of ham

        self.p_spam = self.k_spam / self.N
        self.p_ham = self.k_ham / self.N

        #Calculating means for both the classes
        self.mu_spam = np.mean(self.spam, axis=0)
        self.mu_ham = np.mean(self.ham, axis=0)

        # Avoid division by zero adding a small constant
        #Calculation variances for both the classes
        self.var_spam = np.var(self.spam, axis=0) + 1e-128
        self.var_ham = np.var(self.ham, axis=0) + 1e-128
```

and for the accuracy mean and variance:

```
#10 way cross validation
scores = cross_val_score(NaiveBayesClassifier(), X, Y, cv = 10)

print("Min Accuracy: " + str(scores.min())+"\n")
print("Mean Accuracy: " + str(scores.mean())+"\n")
print("Max Accuracy: " + str(scores.max())+"\n")
print("Variance/Std Accuracy: " + str(scores.var()) +" / " +str(
    scores.std())+"\n")
```

Following are the results by naive Bayes classifier:

Scores	Min Accuracy	Mean Accuracy	Max Accuracy	Variance	Standard Deviation
Naive Bayes	0.70869	0.80507	0.85869	0.000707	0.02660

If we compare these results with the SVM classifier, we can see that SVM performs better than naive Bayes.

5 K-NN

K-Nearest Neighbour is the a widely used classifier in machine learning mostly in the field of image and video recognition but in this assignment, we used this algorithm for classification problems. K-NN is also known for its non-parametric and slow learning technique. It is non-parametric in a sense that it does not work on the assumptions related to the mathematical theory. The model determined from the data does not have to train itself from the data points which makes the training faster but slow down the testing phase. Therefore, K-NN requires more memory and space.

5.1 Working of the K-NN

K is a constant number of the nearest neighbors. The value of K is a critical factor in the algorithm because the classification depends upon the number of neighbours. Moreover, it should be an odd number, for instance, if $K = 1$ the algorithm will only consider one closest neighbor and assigns the new point the class of that neighbor. It can be demonstrated via an illustration below:

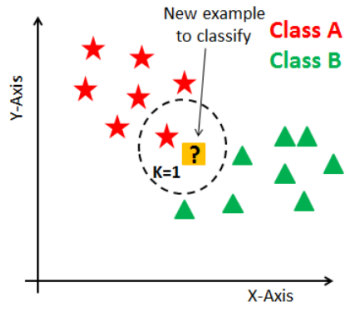

Figure 16: KNN where $K = 1$

In Figure 16, let there is a point which needs to be classified as class A or class B. The classification will be decided on the majority of votes by its K neighbors.

In this case, $K = 1$, hence, only one vote in the radius will be considered. Since there is only one class (Class A) the point will be labelled as Class A. If there have been three or more points the majority of votes is taken as the prediction. The last question that needs to be addressed is that how the closest points are calculated? Minkowski distance is used to calculate the distance between two data points and if we manipulate the value p in the formula below, we get two other distances i.e Manhattan and Euclidean Distance.
Minkowski Distance:

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} .$$

If $p = 1$, it is known as Mahattan Distance

$$d = \sum_{i=1}^{n} |x_i - y_i| .$$

If $p = 2$, it is known as Euclidean Distance

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} .$$

The algorithm follows some basic steps, first, it calculates the distance then it finds the closet neighbors, and finally, votes for the labels are considered as shown below:

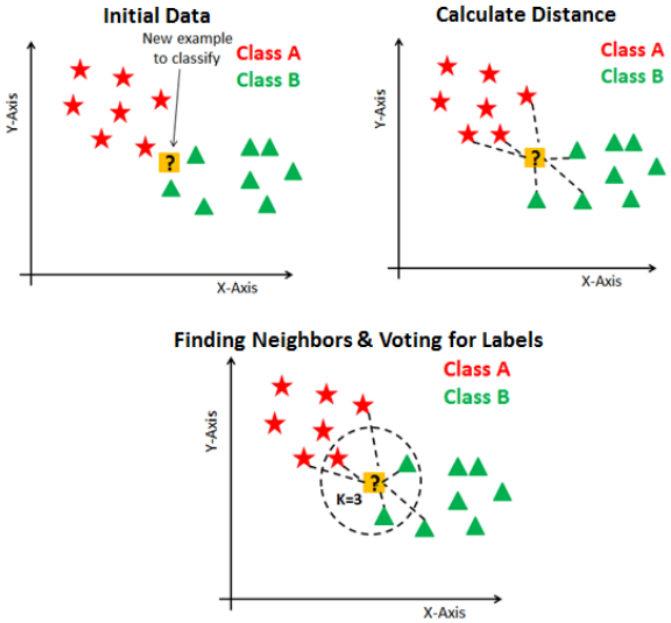

Figure 17: K-NN Working

Whenever a new data point enters the data-set, the algorithm have to assign its class(A, B). Firstly, it calculates the distance from its surrounding nearest neighbors, in this example we set the value $K = 3$, i.e., only three neighbors take part in the voting. We can see from the Figure 17 that there are two votes for the green and one for the red, as a result the K-NN assigns this new data point as green.

5.2 Eager Vs Lazy Learners

Eager learners are the ready-made and active learners that construct a model from given training points before performing the prediction on a new data point to be classified. They are active and ready to classify the even unobserved data points.

Lazy learning is also called instance-based learning learners and does not have to learn the model neither have to train the model instead it uses the points when it's given a new tuple for the prediction. It is only active when it needs to classify a data point as it stores all the training dataset. Unlike the eager, the lazy learner classifies the data point by putting less effort in the training part and have to do more work in the testing part.

5.3 Curse of Dimensionality

One problem for K-NN is the curse of dimensionality. The performance of K-NN is better when we have a smaller number of features. It becomes worst when the number of features are more than the requirement of data, but if we increase the dimension it will lead to overfitting. Therefore, we should avoid that by exponentially increasing the number of dimensions and this is known as the curse of dimensionality.

To overcome this problem we need to apply certain approaches known as feature selection or principle component analysis (PCA). Both the approaches do not depend on the euclidean distance but cosine similarity which is less affected by the higher dimensions.

5.4 Deciding the K Value

Choosing the K value is very crucial in this classifier. This K value is also known as the hyperparameter of K-NN. We have to choose this K value according to the requirements of the data set, for example, if we choose small value for the K there would be a high impact on the results, whereas, if we choose larger K it would take more time in computations. Moreover, with the smaller value of K the model would fit better and flexible with the low biasness and high variance. On the other hand, larger values of K yield low variance, high biasness, and smooth decision boundary.
We choose the hyperparameter K as an odd number because of the voting

process. Suppose if the value of $K = 4$ and we have to predict the class of new data point. The voting is carried out and if there are two votes for both green and red, then there would be a clash and algorithm may predict wrongly. We can choose K as follows:

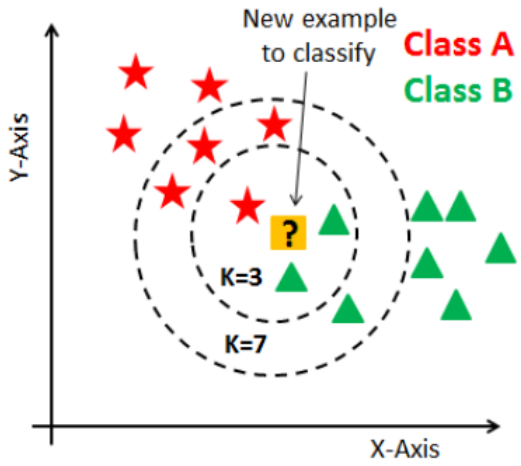

Figure 18: Choosing K

5.5 Implementation and Results

For the coding of the K-NN, we use the sci-kit-learn library of the python class and we compared on three different values of K for the analysis.

```
#Learning by the model
from sklearn.model_selection import train_test_split
from sklearn import metrics
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size
    =0.3)

#Classifying using KNN with K=3,5,7
knn = KNeighborsClassifier(n_neighbors=3)
knn = KNeighborsClassifier(n_neighbors=5)
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
acc_KNN = cross_val_score(knn, X, Y, cv = 10, n_jobs= 8)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Min Accuracy #NN: " + str(acc_KNN.min()) + "\n")
print("Max Accuracy #NN: " + str(acc_KNN.max()) + "\n")
print("Mean Accuracy #NN: " + str(acc_KNN.mean()) + "\n")
print("Variance Accuracy #NN: " + str(acc_KNN.var()) + "\n")
print("================================")
```

Comparing the results

KNN	Accuracy	Min Accuracy	Mean Accuracy	Max Accuracy	Variance
K=3	0.9102	0.9021	0.9239	0.9132	0.00004
K=5	0.9051	0.8847	0.9217	0.9069	0.0001
K=7	0.9000	0.8872	0.9217	0.9069	0.0001

There is a very slight difference among the three models, however, we can see that $K = 3$ model has higher accuracy than the other two models.

6 Comparison Between Three Models

The last task of this assignment is to compare the three models, i.e., SVM, naive Bayes, and K-NN with K equals to 5 on a given dataset of spam filter. The main goal of these three models is to generate a model which returns 1 if the mail is a spam and 0 if the mail is a ham. We have to recall the categories of classifiers. All the classifiers belong to two categories one is discriminative and the other is generative. We use two discriminative classifiers, i.e., the SVM and KNN and one generative classifier, i.e., Naive Bayes. Discriminative classifiers mainly focus on learning with the decision boundary that separates the two classes ham/spam. A hyperplane is learned in SVM and in K-NN without building the model the classifier performs the task.

On the other hand, generative classifiers work on the joint probability that leans on data distribution. The figure below demonstrates how these two classifiers build a model and predict the data points:

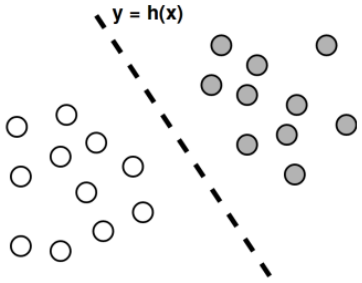

Figure 19: Discriminative Classifier

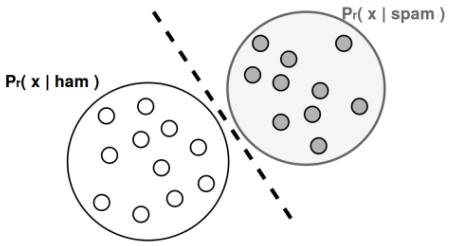

Figure 20: Generative Classifier

As shown in the Figure 20, the generative classifier is difficult to manage the classification. This method is not just dependent on the learning function but also handles the probability distributions. One benefit of the generative approach is that it creates the new pairs despite knowing the joint probability distribution $p(x, y)$. Moreover, generative classifiers can easily identify the outliers which is one of its main advantages. However, in SVM we can not identify the outliers because it calculates the distance from the decision boundary. Therefore, more the distance more the amiability outliers may be created by noise. If an arbitrary label is false or it is coming from another distribution with lower probability, naive Bayes can overcome since it checks both the probabilities of ham and spam. Although SVM and K-NN can classify it correctly.

Talking about the time and complexity of these three classifiers, the SVM requires more time as it only considers the support vectors and the algorithm have to iterates many times to find the best possible hyperplane. Therefore, more time is consumed in the learning phase for SVM, whereas, K-NN and naive Bayes require less time and space because of the single iteration. K-NN uses the lazy learned approach and classifies the dataset in a fast manner.
In naive Bayes, we are following the two assumptions: i) conditionally independence between the variable, and ii) the normal probability distribution. Since these assumptions are not truly satisfied, we can conclude the discriminative classifiers(SVM, K-NN) perform better for larger datasets, however, if we have smaller dataset SVM and K-NN might perform worse. In general, the generative classifies perform better if assumptions are satisfied.

7 Conclusion

We compared the performance of three classifiers, i.e., SVM, naive Bayes, and K-NN. We also studied how they perform in different situations. We have given a spam base dataset with more than 50 data points and one prediction class spam(1) or ham(0). We have also discussed some advantages and disadvantages of three classifiers and their behavior.
First, we used the SVM classifier which performed with higher accuracy on the linear and as well as on non-linear data. For later we employed kernel tricks (linear, polynomial with degree 2, radial basis function) which transform features space to another higher dimension. We discussed the effect of SVM regarding the vector length and transform our kernel to angular kernel which yields better results. Secondly, we used naive Bayes using the Gaussian distribution that did not perform well as compared to SVM(Angular kernels). The main reason for this is that the assumptions were not satisfied, therefore, the algorithm did not tend to learn particular patterns. Finally, we used K-NN(K nearest neighbour) with the $K = 5$ for the classification problem. For the analysis, we used three different values for K = 3,5, and 7. We conclude that the smaller K would perform better than the larger K values.