Project Report: Machine Learning Assignment - 1

VRUTIK HALANI DHRUVIL SHAH YUG AJMERA

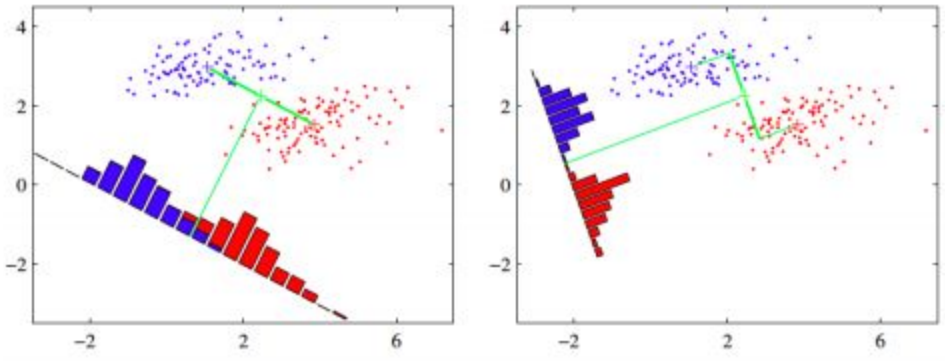
CONTENTS :

* Fischer Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a dimensionality reduction technique. As the name implies dimensionality reduction techniques reduce the number of dimensions (i.e. variables) in a dataset while retaining as much information as possible. For instance, suppose that we plotted the relationship between two variables where each color represents a different class.

WHY ?

The core idea is to learn a set of parameters w ∈ R (d×d′) ,that are used to project the given data x ∈ R (d) to a smaller dimension d′. The figure below (Bishop, 2006) shows an illustration. The original data is in 2 dimensions, d=2 and we want to project it to 1 dimension, d=1.



If we project the 2-D data points onto a line (1-D), out of all such lines, our goal is to find the one which maximizes the distance between the means of the 2 classes, after projection. If we could do that, we could achieve a good separation between the classes in 1-D. This is illustrated in the figure on the left and can be captured in the idea of maximizing the "*between class covariance*". However, as we can see that this causes a lot of overlap between the projected classes. We want to minimize this overlap as well. To handle this, Fisher's LDA tries to minimize the "*within-class covariance*" of each class. Minimizing this covariance leads us to the projection in the figure on the right

hand side, which has minimal overlap. Formalizing this, we can represent the objective as follows.

$$J(w) = \frac{w^\top S_b w}{w^\top S_w w}$$

where Sb ∈ R (d×d)and Sw ∈ R (d×d)are the between-class and within-class covariance matrices, respectively. They are calculated as

$$S_b = \sum_{k=1}^{K} (m_k - m) N_k (m_k - m)^\top$$

$$S_w = \sum_{k=1}^{K} \sum_{n=1}^{N_k} (X_{nk} - m_k)(X_{nk} - m_k)^\top$$

where Xnk is the nth data example in the kth class, Nk is the number of examples in class k, m is the overall mean of the entire data and mk is the mean of the kth class. Now using Lagrangian dual and the KKT conditions, the problem of maximizing J can be transformed into the solution :

$$S_w^{-1} S_b w = \lambda w$$

which is an eigenvalue problem for the matrix inv(Sw).Sb . Thus our final solution for w will be the eigenvectors of the above equation, corresponding to the largest eigenvalues. For reduction to d′ dimensions, we take the d′ largest eigenvalues as they will contain the most information. Also, note that if we have K classes, the maximum value of d′ can be K−1. That is, we cannot project K class data to a dimension greater than K−1. (Of course, d′ cannot be greater than the original data dimension d). This is because of the following reason. Note that the between-class scatter matrix, Sb was a sum of K matrices, each of which is of rank 1, being an outer product of two vectors. Also, because the overall mean and the individual class means are related, only (K−1) of these K matrices are independent. Thus Sb has a maximum rank of K−1 and hence there are only K−1 non-zero eigenvalues. Thus we are unable to project the data to more than K−1 dimensions.

HOW ?

In short , following are the key steps to be followed :

1. Group the training data into it’s respective classes . i.e Form a dictionary and save data grouped by classes it belongs to.
2. Calculate mean vector of given training data of K-dimensions excluding the target class and also calculate class-wise mean vector for the given training data.
3. Calculate SB and SW matrices needed to maximize the difference between means of given classes and minimize the variance of given classes.

4. After that calculate the matrix M , where M = (inv(SW)).(SB)
5. Calculate eigenvalues of M and get eigenvector pairs for first n (needed) dimensions.
6. These eigen vectors give us the value of ‘w’ used to transform points to the needed number of dimensions.
7. After applying the transformation to every point i.e. Transformed_Point = w.dot(given_point) , we calculate the threshold value by which we will classify points into positive and negative class.
8. To calculate Threshold use Gaussian Normal Distribution.

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

where: y = vertical height of a point on the normal distribution

x = distance along the horizontal axis

σ = standard deviation of the data distribution

μ = mean of the data distribution

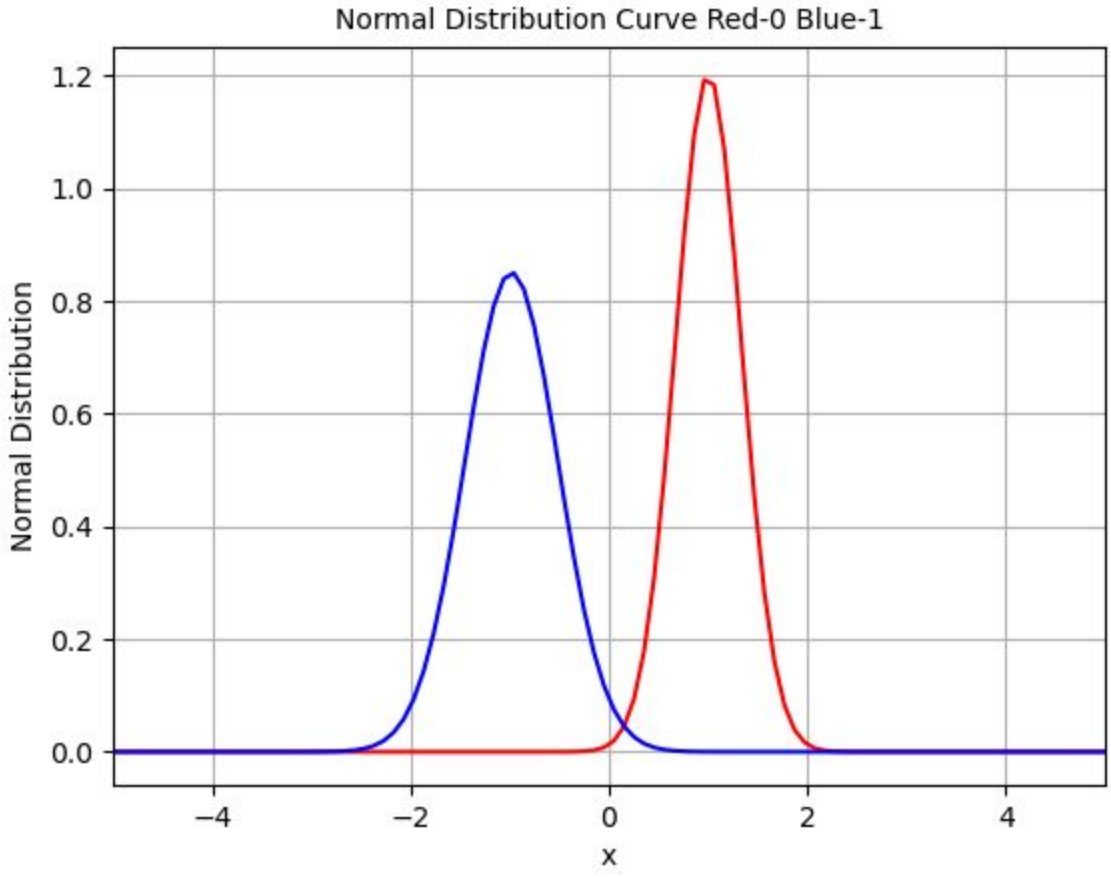
e = exponential constant = 2.71828...

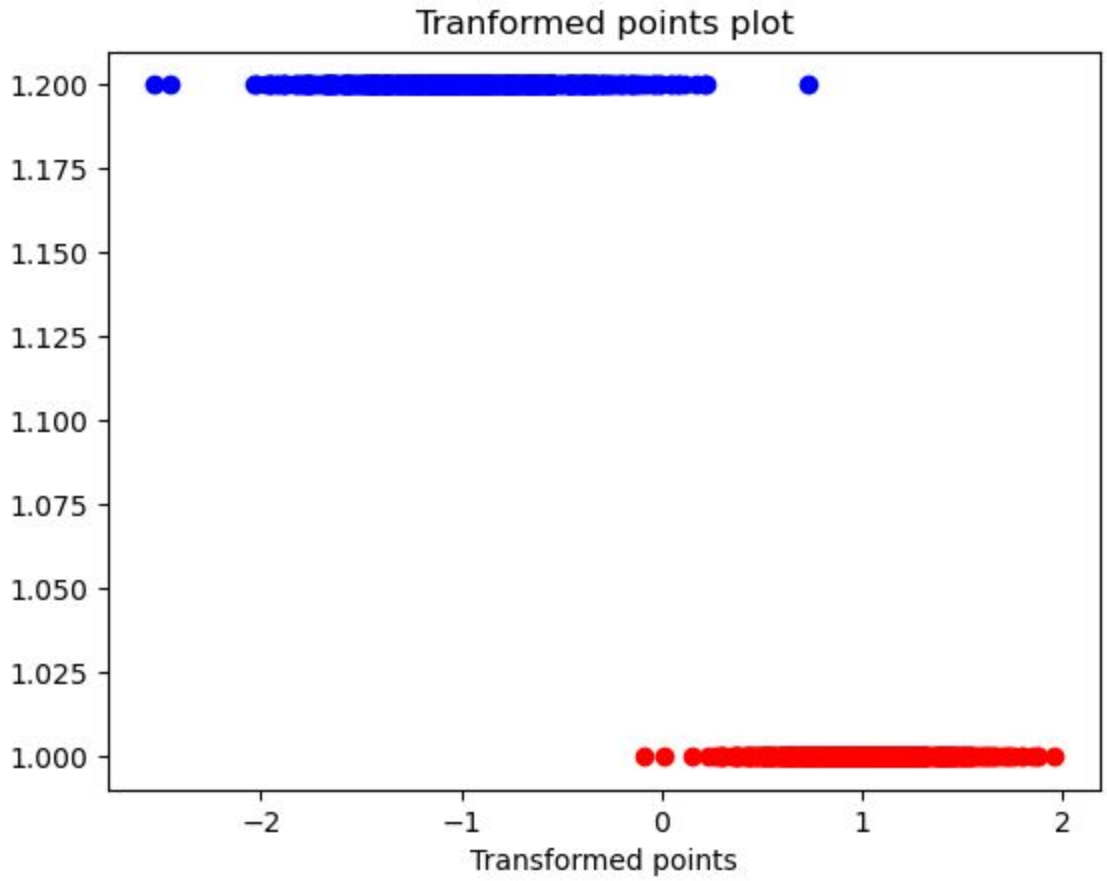
π = pi = 3.14159....

9. After calculating Normal Equation of both classes , we get threshold value and then classify points by threshold.

EXPERIMENTS :

1. Dataset a1_d1.csv



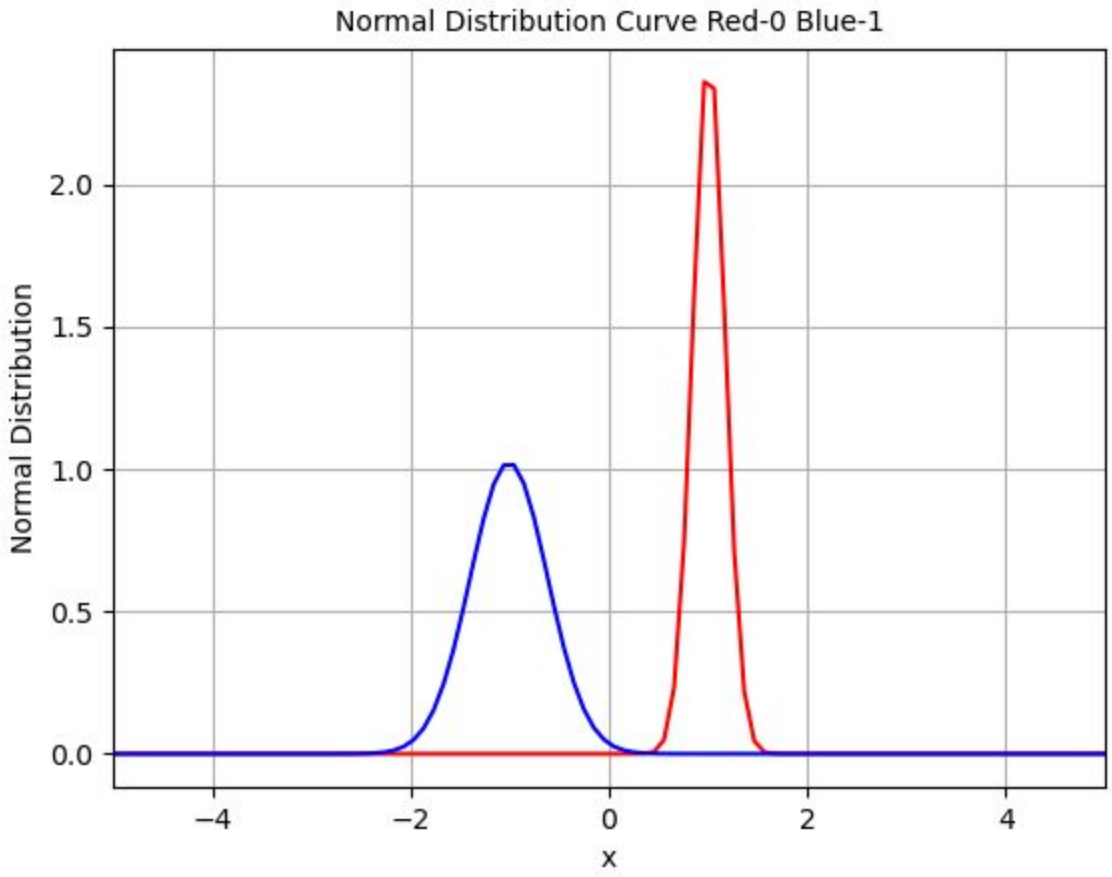


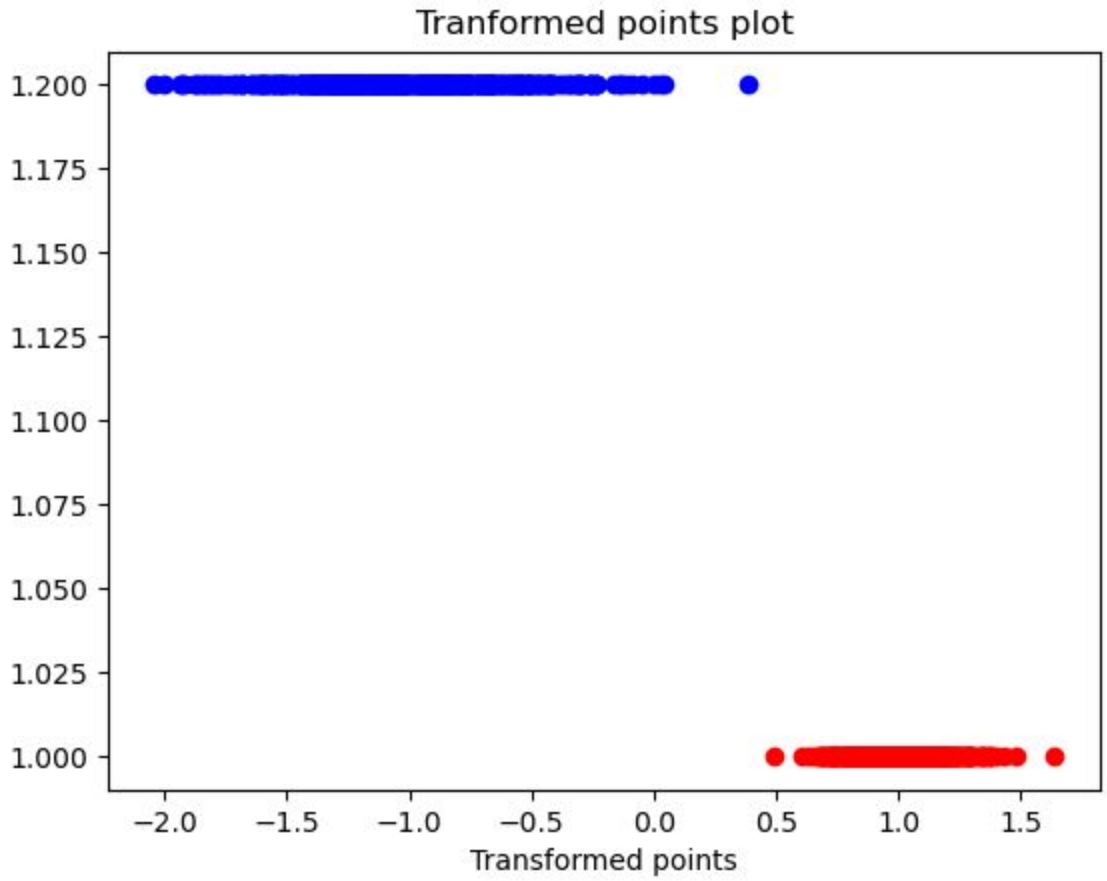
Threshold after transformation = 0.15207088287130205

Accuracy = 0.993

F1-Score = 0.9929929929929929

2. Dataset a1_d2.csv





Threshold after transformation = 0.38930280209937634

Accuracy = 1.0

F1-Score = 1.0

Sentiment Analysis Using Naive Bayes

We use Naive Bayes classifier to correctly classify a review as **positive(1)** or **negative(0)**. In our data ,these are the two **classes** to which each document belongs.

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|d)$$

In more mathematical terms, we want to find the **most probable class given a document**, which is exactly what the above formula conveys. **C** is the set of all possible classes, **c** one of these classes and **d** the document that we are currently classifying. We read **P(c|d)** as the probability of class c, given document d.

We can rewrite this equation using the well known Bayes' Rule, one of the most fundamental rules in machine learning. Since we want to maximize the equation we can drop the denominator, which doesn't depend on class c.

The rewritten form of our classifier's goal naturally splits it into two parts, the **likelihood** and the **prior.** You can think of the latter as "the probability that given a class c, document d belongs to it" and the former as "the probability of having a document from class c". To go a step further we need to introduce the assumption that gives this model its name.

Naive Bayes assumption: given a class c, the presence of an individual feature of our document is **independent** on the others.

We consider each individual word of our document to be a feature. If we write this formally we obtain:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{f \in F} P(f|c)$$

The Naive Bayes assumption lets us substitute P(d|c) by the product of the probability of each feature conditioned on the class because it assumes their independence.

We can make one more change: maximize the log of our function instead. The reason for this is purely computational, since the log space tends to be less prone to underflow and more efficient. We arrive at the final formulation of the goal of the classifier.

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

P(c) is simply the probability of encountering a document of a certain class within our data. This is easily calculated by just dividing the number of occurrences of class c by the total number of documents.

P(w_i|c) is the probability of word w_i occurring in a document of class c. We can calculate this as Number of occurrences of Word_i in the text of class c divide by number of words in data of class c.

After training the model, the next step comes is testing.

In testing we take one review, and multiply the probability of the word stored in the dictionary of class Ci. The result is predicted as : After multiplying the values whichever class gets the maximum probability is assigned.

Of course there may be words present in text which are not in the **vocabulary** build using training data.

We can use smoothing in that case :

$$\hat{P}(w_i|c) = \frac{count(w_i,c)+1}{\sum_{w\in V}(count(w,c)+1)} = \frac{count(w_i,c)+1}{\left(\sum_{w\in V} count(w,c)\right)+|V|}$$

|V| = denotes the size of the vocabulary.

Implementation :

Step1 :Data Preprocessing

Tokenizing : We split all words separated by space(‘ ‘) so that we can perform Naive Bayes.

Next Step was to remove punctuations and numbers since they don't constitute much to sentiment analysis.

```
(base) D:\Sem 6\ML\Assignment\Assign1>python main_code.py
Accuracy :  [0.73, 0.75, 0.775, 0.79, 0.765] 0.762 +- 0.02063976744055031
F-score :  [0.7476635514018691, 0.7663551401869159, 0.7804878048780487, 0.8073394495412846, 0.7766990291262136] 0.7757089950268663 +- 0.019485093545528947
Recoil :  [0.851063829787234, 0.8282828282828283, 0.7843137254901961, 0.8, 0.8421052631578947] 0.8211531293436305 +- 0.02525498397016405
Precision :  [0.6666666666666666, 0.7130434782608696, 0.7766990291262136, 0.8148148148148148, 0.7207207207207207] 0.7383889419178571 +- 0.051782661510277654
```

Result with keeping punctuations

```
(base) D:\Sem 6\ML\Assignment\Assign1>python main_code.py
Accuracy :  [0.795, 0.825, 0.815, 0.8, 0.82] 0.8110000000000002 +- 0.011575836902790184
F-score :  [0.7960199004975126, 0.8416289592760181, 0.8195121951219513, 0.8113207547169812, 0.8309178743961353] 0.8198799368017197 +- 0.01573246458293537
Recoil :  [0.8, 0.9117647058823529, 0.8484848484848485, 0.8349514563106796, 0.8958333333333334] 0.858206868802243 +- 0.040761658972066184
Precision :  [0.7920792079207921, 0.7815126050420168, 0.792452830188679 3, 0.7889908256880734, 0.7747747747747747] 0.7859620487228673 +- 0.006836315060811325
5
```

Result without keeping punctuations

Step 2: Implementing 5 fold Cross validation

We randomly shuffle data , and divide the data into 5 sets of 200 examples each. Because of randomized shuffling we ensure that every file has got nearly equal number of examples of each class.

We then use 4 files as train set and remaining file as test set , after doing this for every file we print average accuracy and average F_score.