Report: Assignment 3

In this assignment we will explore how dataset of bill authenticity is learned by different classifiers. The idea to use binary classification or half space classifiers is because of separating the dataset into two parts of the input space. Then there will be some datasets which might be separable and some are completely non separable. If we are able to achieve a hyperplane dividing the input space into 2 halves, then we say we have achieved linear separability.

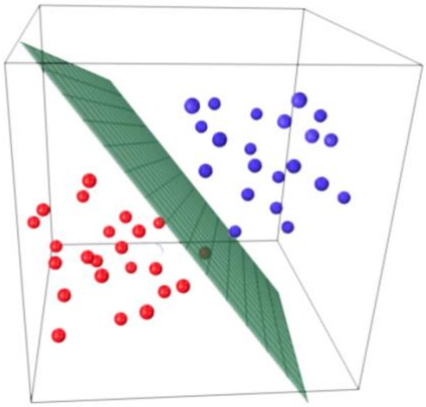

Figure 1 Hyperplane dividing the input space

Understanding Dataset:

Bill authentication dataset contains the information of the pixels of the bank notes 400x400 pixels. It contains 5 variables (columns).
Attribute Information:
1. Variance of Wavelet Transformed image (continuous)
2. skewness of Wavelet Transformed image (continuous)
3. Curtosis of Wavelet Transformed image (continuous)
4. entropy of image (continuous)
5. class (integer) (Target variable) (1: genuine, 0: Forged)

The dataset was created manually by capturing images of fake (forged) and real (genuine) notes. The data was created for checking the authenticity of the bank notes. The greyscale images were then used to harness the features of the notes. Wavelet transform tool was used to extract accurate figures of the bank notes. *This data set is recommended for learning and practicing your skills in exploratory data analysis, data visualization, and classification modelling techniques*.

Understanding the assignment:

The assignment covers the 5 different methods for binary classification tasks, moreover, since the variable to be predicted (*class*) is also binary which is identifying the bank note is legal or illegal. The goal is to identify and classify the bank notes as a function of features, using different proven methods along with some changes in the application.

Figure 2 Pie chart of class variable

Further we divide the dataset into two halves, that is training set and test set. The training set is on which we build our model/classifier and the test set is on which we check our trained model. We check accuracy of the trained model by applying the test set. Since we don't have much information and lack of resources it is better to skip validation set for simplicity. Additionally, we apply pair-plot for visualizing the data to check how the variables are co-related, and plotting variance vs skewness.

Task 1: Half Space classifier:

This type of classifier is the simplest form of classification problem. We come with 2 class labels that is {+,-}. The real numbers are mapped to either {+,-} which is done by checking the sign. Sign can be checked using sign function as well. The hyperplane straight line we get the value 0. On one side of the straight line where <w,x> > 0 which is positive and other <w,x> < 0 which is negative. When we realize it, we become interested in best possible line which could divide the space into two halves. Predictor $h_w(x)$ will result in {+1, -1}

$$a(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}),$$

where

- $\mathbf{x}$ – is a feature vector (along with identity);
- $\mathbf{w}$ – is a vector of weights in the linear model (with bias w_0);
- $\text{sign}(\bullet)$ – is the signum function that returns the sign of its argument;
- $a(\mathbf{x})$ – is a classifier response for $\mathbf{x}$.

Figure 3 Binary Classification problem

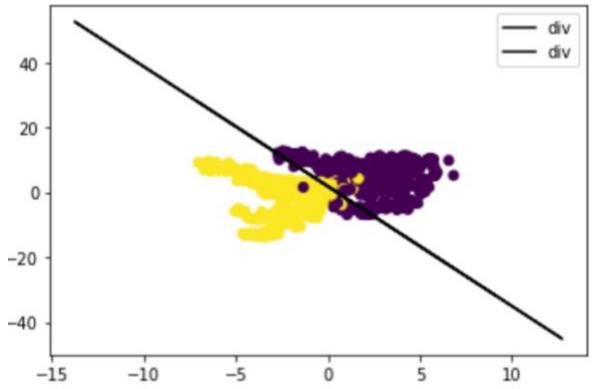

Figure 4 Variance vs Skewness

On looking for the solution to get the hyperplane/line we check 2 things. First is correct classification of the data point, that is $y_i < w_i, x_i > \ > 0$ or (actual label).(value returned by the hypothesis) > 0 for correct classification.
Single layer perceptron algorithm is usually used for binary classification where we divide the input space into two halves in the separable case. Hence in our case since the single layer perceptron algorithm did not converge, which means our data is linearly in-separable. Moreover by looking the plot as well, we can say that the data is linearly in-separable. It will not classify correctly if the data set is not linearly separable. For our testing purpose, this is exactly what we need. The linear separability test can be carried out without splitting the dataset into test/train. Thus we can easily tell we have to apply a better algorithm for converging the complex dataset.
The algorithm will run iteratively and at every step make small updates to the parameters. The process works when we detect incorrectly labelled example and our equation becomes Y ($<w^Tx>$) < 0. Then our objective is to adjust w such that parameter is adjusted and crosses zero to become positive. It is guaranteed to converge for realizable case.

Task 2: Logistic Regression:

The ability to provide probabilities and classify new samples using continuous and discrete measurements makes it a popular ML method. The likelihood is the key factor in Logistic regression.
Likelihood: Pick a probability from the distribution curve, and use it to calculate the likelihood of 0/1. Once all likelihood is found, we multiply all. Once we get all likelihoods, we get one line. The same way we get other lines by repeating the likelihood process for all other lines by shifting the line. Finally the curve with maximum likelihood is selected. The important feature of Logistic regression is that it can also be used to assess what variables are useful for classifying examples. Added benefit of predicting the probability of the example x_i.
For example in our case we can chose a threshold which could determine the level of confidence of classifying the examples. Points near boundary will always have a confusion. In such case our half space classifier, is unable to distinguish between points which are likely to get confused about their class labels and points which are very sure of their class labels.
Hypothesis class for Logistic Regression:
$H_{sigmoid}$ = {X ➔ Sigmoid(<w,x>): w}

Loss function of logistic regression is convex thus we can minimize the loss using Gradient Descent procedure as it is convex function. Compute the derivative and move small steps.

```
[95] #LOGISTIC REGRESSION
     from sklearn.linear_model import LogisticRegression
[95] lr=LogisticRegression()
     lr.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
[96] from sklearn.metrics import accuracy_score
     y_pred=lr.predict(X_test)
     score=accuracy_score(y_test,y_pred)
     score
```

```
0.9902912621359223
```

Figure 5 snapshot from notebook

Using LogisitcRegression function, we can easily operate on modelling our dataset. Once we are trained with the lr.fit(Training Dataset) we can predict any classification corresponding the example. We can further calculate the accuracy of the model, that is how did it perform on test set, using the classifier which was trained on train set.

Task 3+4: SVM classifier (using a linear kernel) + SVM classifier (using a Polynomial kernel and a Gaussian kernel)

We run a linear kernel SVM classifier over the variables Variance and Skewness. Since we want to check if our dataset is getting separated by the line by SVM, we are strict to the margin of Hard SVM formulation. Margin is the maximum distance between the closest point to the hyperplane. Since we are checking for the separability we are strict to our conditions of hard margin such that it results something else if does not get converged. Since figure 4 tells our data is non-separable because there is no perfect classification or separation. The support vector classifier class, which is written as SVC in the Scikit-Learn's svm library. The function takes input as the type of kernel. This is very important. In the case of a simple SVM we simply set this parameter as "linear" since simple SVMs can only classify linearly separable data. We will see non-linear kernels in the next Task 4. The hard SVM learning rule says – "select the hyperplane that separates the training set with largest possible margin. The hyperplane where margin is very close, and maximize the minimum distance to the closest data point."

SVM needs to find the optimal line with the constraint of correctly classifying either class: Follow the constraint: only look into the separate hyperplanes(e.g. separate lines), hyperplanes that classify classes correctly Conduct optimization: pick up the one that maximizes the margin

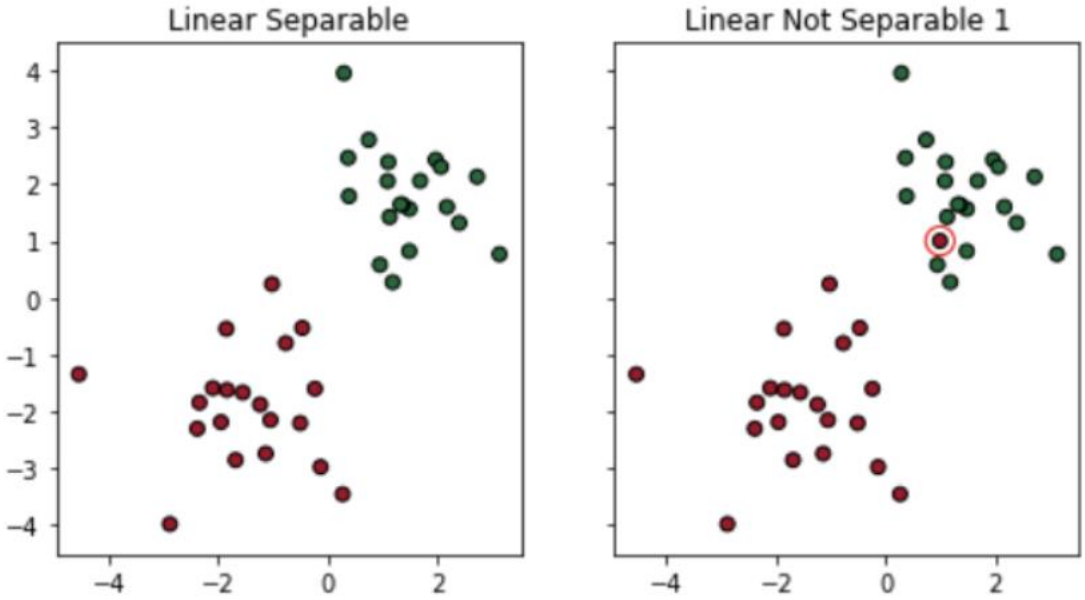

Figure 6 Linear separable and non-separable example

Soft Margin: try to find a line to separate, but tolerate one or few misclassified dots (e.g. the dots circled in black) and Kernel Trick: try to find a non-linear decision boundary.

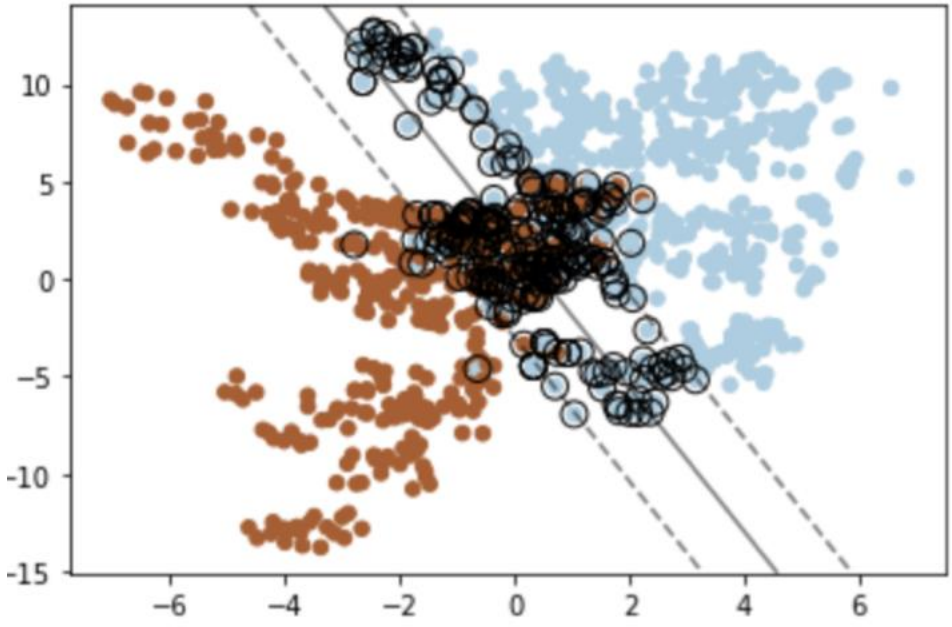

Figure 7 Soft Margin

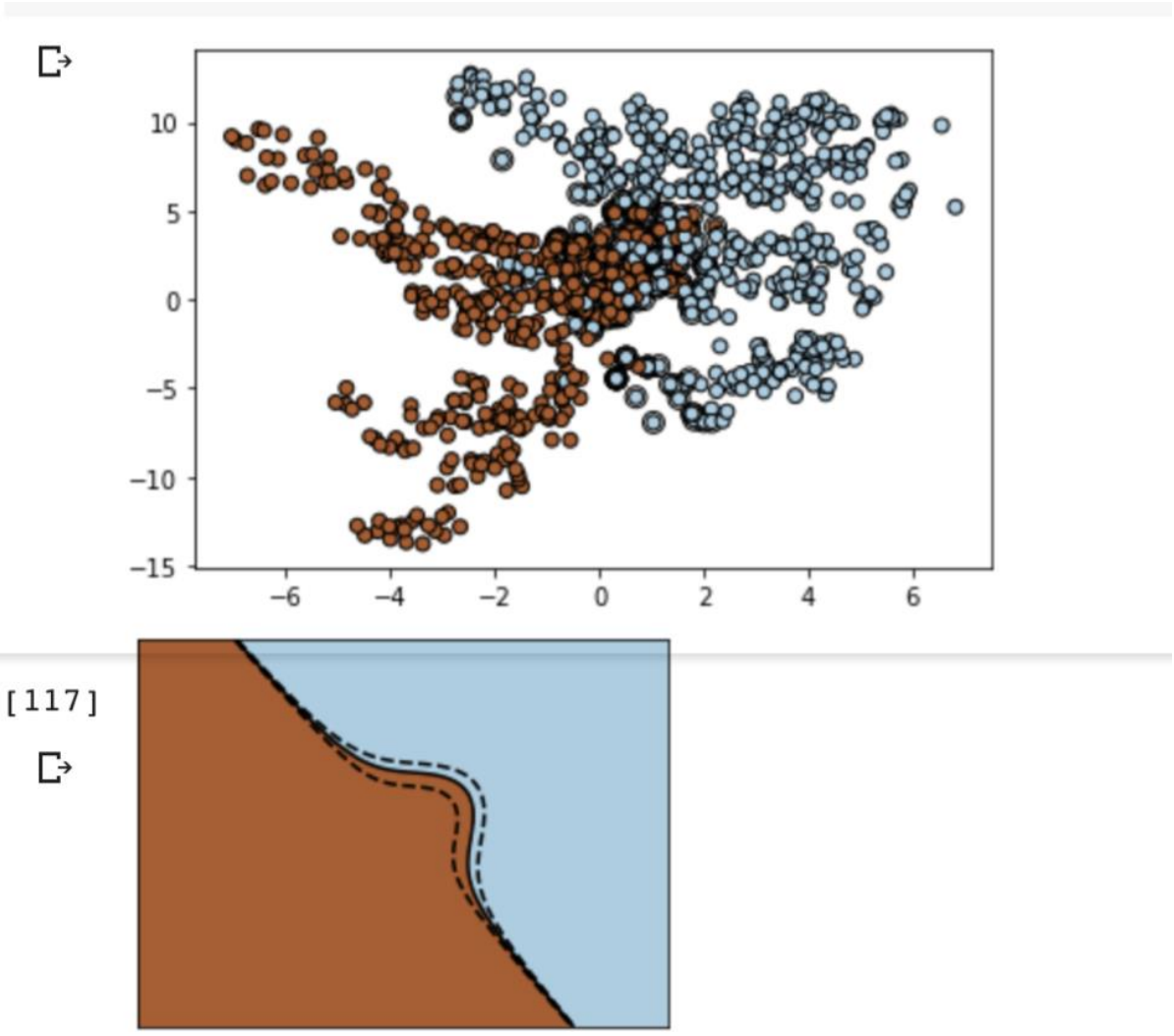

Figure 8 Non-linear Kernel

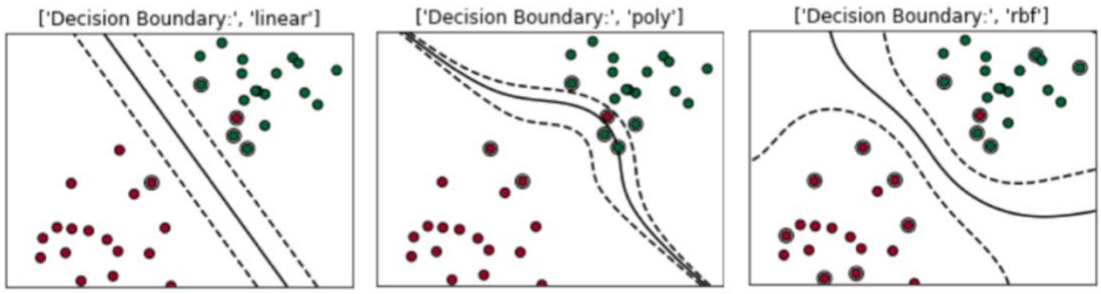

Figure 9 Example of linear, polynomial and gaussian/radial bias function

Soft SVM learning rule: Sum of all penalty terms for all the examples, total cost which is included as part of minimization, our optimizer will try to minimize the number of points which have penalty associated with it. Best practice is to minimize the number of points getting misclassified and minimum number of points happens to be inside the margin. We

also have a hyperparameter which controls the tradeoff between the minimization terms. It evaluates the weight you want to give to the penalty and hyperplane.

When regularization parameter / hyperparameter is large: Separation is good between + and – points even after misclassification. Implies larger margin.
When regularization parameter / hyperparameter is small: Hyperplane will be in such way that minimum number of points gets misclassified. Generalization will be poor.

SVM becomes stronger with the kernel applications. We can push ahead our normal SVM formulation ahead and get interesting polynomial graphs which could separate our input space more accurately. Scalability and robustness comes with SVM because as you add/remove more points in the hyperplane will not get affected, unless you add/remove some points in the margin. The parameters of the hyperplane are linear combination of support vectors. Even if the data exists in high D, we can converge it with very few coefficients alphas corresponding to examples.As compared with Half Space classifier, we have saved a lot of computation, because half space uses all parameters and x_is to get the classifier. Instead of storing w components, for all examples explicitly, we store alpha values and indices for training examples. Implementing Soft SVM using SGD: If f is lambda-strongly convex function (when the curve always stays upside). When we get a convex function, gradient descent can be applied easily. Moreover, with lambda-strongly convex function, we can learn the gap as well and can converge faster. Changing step sizes will help in converging to global minimum faster. $w^{(t)} = w^{(t-1)} – (\text{StepSize}* \triangle)$. The data which was not separable in smaller D can be separable in higher D.

Kernel function approximate the inner product in the high dimensional space. That inner product will be approximated using Kernel function, computationally frugal function to approximate the inner product in high dimensional space.
Using this kernelized support vector machine, we learn a suitable nonlinear decision boundary. This kernel transformation strategy is used often in machine learning to turn fast linear methods into fast nonlinear methods, especially for models in which the kernel trick can be used.

References:

1. Class Lectures
2. https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496
3. https://www.tarekatwan.com/index.php/2017/12/methods-for-testing-linear-separability-in-python/
4. https://education.rstudio.com/blog/2020/07/palmerpenguins-cran/
5. https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html
6. https://towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d
7. https://sdsawtelle.github.io/blog/output/week7-andrew-ng-machine-learning-with-python.html
8. https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/