Report

Mustafa Ozan Alpay

June 11, 2021

1 Part 1: Decision Tree

1.1 Information Gain

Test results and referring to the tree diagram.

1.2 Average Gini Index

Test results and referring to the tree diagram.

1.3 Information Gain with Chi-squared Pre-pruning

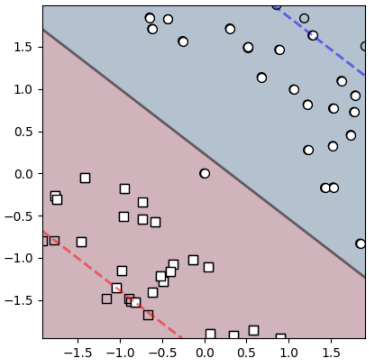
Test results and referring to the tree diagram.

1.4 Average Gini Index with Chi-squared Pre-pruning

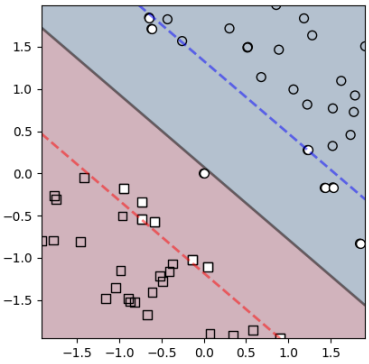
Test results and referring to the tree diagram.

2 Part 2: Support Vector Machine

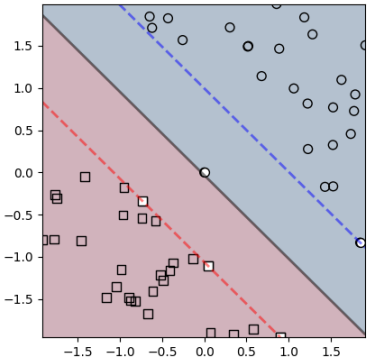
2.1 First Part



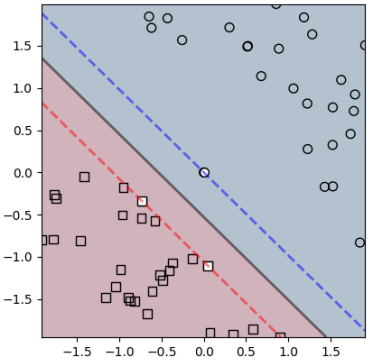
C = 0.01



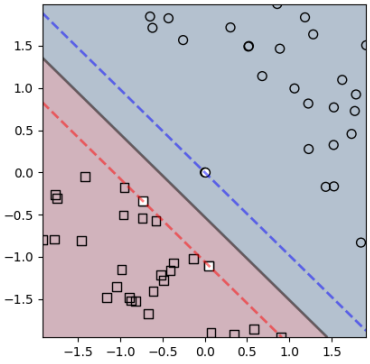
C = 0.1



C = 1.0



C = 10.0

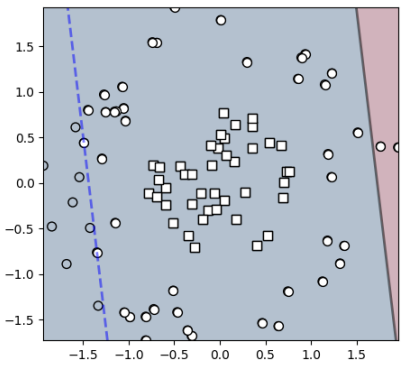


C = 100.0

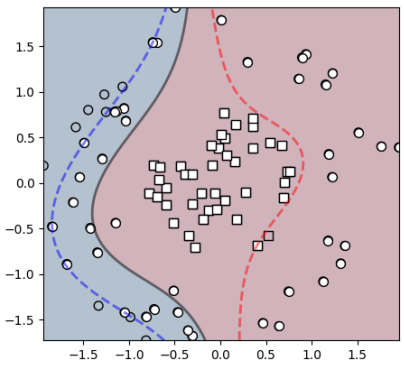
Figure 1: Linear SVM plots with different C values

As we increase the `C` values, the margins get smaller, which affects the support vectors. By changing the `C` values, the hyperplane might get affected from that change as well, which as a result affects the SVM. Using different `C` values might help us ignoring outliers in some cases, which may yield better performing SVM models. To understand if we have any outliers within the data, we might try plotting the data to understand the clusters and overall location of data points, and decide according to that result.

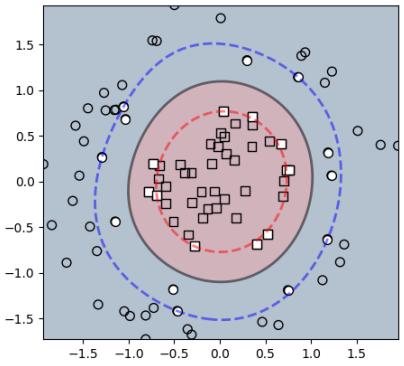
2.2 Second Part



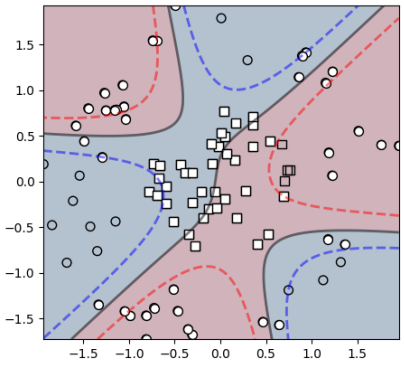
Linear Kernel



Polynomial Kernel



RBF Kernel



Sigmoid Kernel

Figure 2: SVM plots with different kernels

As we change the kernels, the classification changes a lot. We use different kernels to create the best performing SVM models that utilizes hyperplanes, and some datasets are not linearly separable. In order to create a model that can separate the data linearly, by using kernels, we increase the dimension of the data and we try to find a dimension that is useful for our model. Since the plots are in 2 dimensions, the support vectors and the hyperplane might appear wrong, but it their original dimensions they make perfect sense and perform accordingly. If we manage to find a

kernel that works properly for our model, we can create better performing SVM's and have better performing classifiers.

In our dataset, we can see a cluster of squares in the middle section, and circles surrounding it. In the original linear kernel, we cannot create a well performing classifier, since the data is not linearly separable. The polynomial kernel tries performs better than the linear kernel, but it fails to separate the data linearly as well. However the RBF kernel gets the job done perfectly due to its mountain-like shape, which allows the center of the data to be extracted properly. Sigmoid kernel would have been useful if the data had a shallow valley like structure in the center, but since our data is not in that distribution model, it fails as well.

2.3 Third Part

gamma	C				
	0.01	0.1	1	10	100
-	0.774	**0.811**	0.779	0.738	0.730

Table 1: Linear kernel

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.510	0.510	0.510	0.511	0.747
0.0001	0.510	0.510	0.511	0.749	0.796
0.001	0.510	0.511	0.751	0.828	0.857
0.01	0.510	0.761	0.869	0.894	**0.896**
0.1	0.510	0.510	0.883	0.885	0.885
1	0.510	0.510	0.510	0.510	0.510

Table 2: RBF kernel

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.510	0.510	0.510	0.510	0.510
0.0001	0.510	0.510	0.510	0.510	0.510
0.001	0.510	0.510	0.551	0.737	0.810
0.01	0.737	0.810	**0.875**	0.867	0.867
0.1	0.867	0.867	0.867	0.867	0.867
1	0.867	0.867	0.867	0.867	0.867

Table 3: Polynomial kernel

According to `GridSearchCV`, the best parameters are `{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}`, which can also be seen on the table. The accuracy of these parameters with the test data is `0.896`.

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.510	0.510	0.510	0.510	0.737
0.0001	0.510	0.510	0.510	0.737	0.774
0.001	0.510	0.510	0.736	0.768	**0.787**
0.01	0.510	0.511	0.337	0.687	0.681
0.1	0.510	0.510	0.510	0.510	0.510
1	0.510	0.510	0.510	0.510	0.510

Table 4: Sigmoid kernel

2.4 Fourth part

2.4.1 Without handling the imbalance problem

In the original dataset, I decided to do an analysis to determine the minority and majority group.

Values	Count of labels
0	121
1	1379

Table 5: Initial distribution of data for SVM Part 4

As it can be seen on the table, the data is heavily imbalanced. When the data is imbalanced, we cannot use accuracy as a good performance metric, since our model will have a tendency towards the majority group due to lack of enough data from the minority group. The initial test accuracy for this dataset is `0.950`. The confusion matrix for this dataset can be seen below.

Actual Class / Predicted Class	0	1
0	**3**	76
1	0	**1421**

Table 6: Confusion Matrix for Initial Data

Let's classify `0` as Positive, and `1` as Negative. This confusion matrix tells us that our SVM has no false positives, but it has **76** false negatives. When we look at the accuracy for the Positive class, we can see that our model has classified only 3 out of 79 positives, which gives us an accuracy of `0.037`. Similarly, for the Negative class, we have `1.000` accuracy. We can say that this model can accurately identify Negatives, but it fails identifying Positives.

2.4.2 Oversampling the minority class

After oversampling the minority class, the minority and majority groups have around 5% difference between them, which is acceptable.

The test accuracy for this dataset is `0.958`. The confusion matrix for this dataset can be seen below.

Values	Count of labels
0	1452
1	1379

Table 7: Distribution of data for SVM Part 4 after oversampling

Actual Class / Predicted Class	0	1
0	**31**	48
1	15	**1406**

Table 8: Confusion Matrix for Oversampled Data

Let's classify `0` as Positive, and `1` as Negative. This confusion matrix tells us that our SVM has **15** false positives, amd it has **48** false negatives. When we look at the accuracy for the Positive class, we can see that our model has classified 31 out of 79 positives, which gives us an accuracy of `0.392`. Similarly, for the Negative class, we have `0.989` accuracy. We can say that this model can accurately identify Negatives, but it performs poorly while identifying Positives. It still performed better than the original model though.

2.4.3 Undersampling the majority class

After undersampling the majority class, the minority and majority groups have equal numbers, which is acceptable.

Values	Count of labels
0	121
1	121

Table 9: Distribution of data for SVM Part 4 after undersampling

The test accuracy for this dataset is `0.817`. The confusion matrix for this dataset can be seen below.

Actual Class / Predicted Class	0	1
0	**50**	29
1	246	**1175**

Table 10: Confusion Matrix for Undersampled Data

Let's classify `0` as Positive, and `1` as Negative. This confusion matrix tells us that our SVM has **246** false positives, amd it has **29** false negatives. When we look at the accuracy for the Positive class, we can see that our model has classified 50 out of 79 positives, which gives us an accuracy of `0.632`. Similarly, for the Negative class, we have `0.826` accuracy. Even though the overall accuracy is lower than the previous two models, it performs twice as accurate while classifying Positives, while not losing much information regarding the Negatives.

2.4.4 Setting the class_weight to balanced

The test accuracy for this test is `0.959`. The confusion matrix can be seen below.

Actual Class / Predicted Class	0	1
0	**38**	41
1	21	**1400**

Table 11: Confusion Matrix for Automatically Balanced Data

Let's classify `0` as Positive, and `1` as Negative. This confusion matrix tells us that our SVM has **21** false positives, amd it has **41** false negatives. When we look at the accuracy for the Positive class, we can see that our model has classified 50 out of 79 positives, which gives us an accuracy of `0.481`. Similarly, for the Negative class, we have `0.985` accuracy. Even though the overall accuracy is higher than the undersampled example, it performs poorly while classifying Positives. However it is still very accurate while identifying Negatives.