Report

Ufuk Cem Birbiri- e2171379

January 8, 2021

1 Part 1: Decision Tree

1.1 Information Gain

I only completed the functions in dt.py file.

1.2 Gain Ratio

Test results and referring to the tree diagram.

1.3 Average Gini Index

Test results and referring to the tree diagram.

1.4 Gain Ratio with Chi-squared Pre-pruning

Test results and referring to the tree diagram.

1.5 Gain Ratio with Reduced Error Post-pruning

Test results and referring to the tree diagram.

2 Part 2: Support Vector Machine

2.1 First Part

Small C values will make the optimizer to consider a hyperplane with larger margin. But with small C values and larger margin, it is more possible to miss-classify data points. With large C values,the optimizer will choose a hyperplane with small margin. Small-margin hyperplane is better for classifying all data points correctly, however it may not be good for testing data. Different C values is shown in Figure 1.

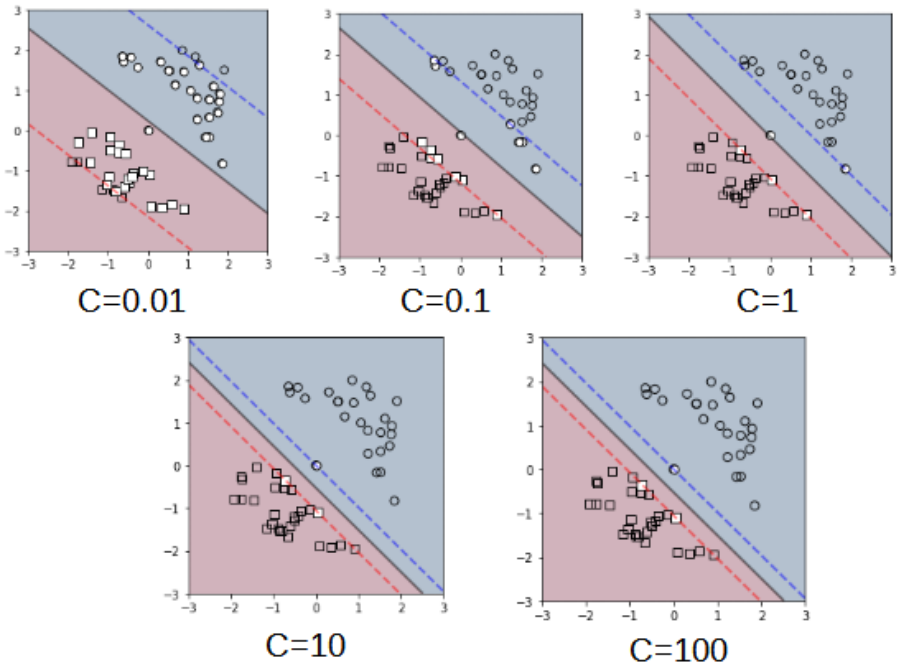

Figure 1: SVM results with different C values.

2.2 Second Part

I leave C parameter as default ie. C=1. In this part, the data is not linearly separable. So the "linear" kernel could not classify the data points. "Poly" and "rbf" kernels are for nonlinear data. Changing the kernel parameter affects the behavior of the classifier such as it tries to classify data points according to their kernel characteristics. The most suitable kernel for this dataset is ""rbf kernel which is shown in Figure 2.

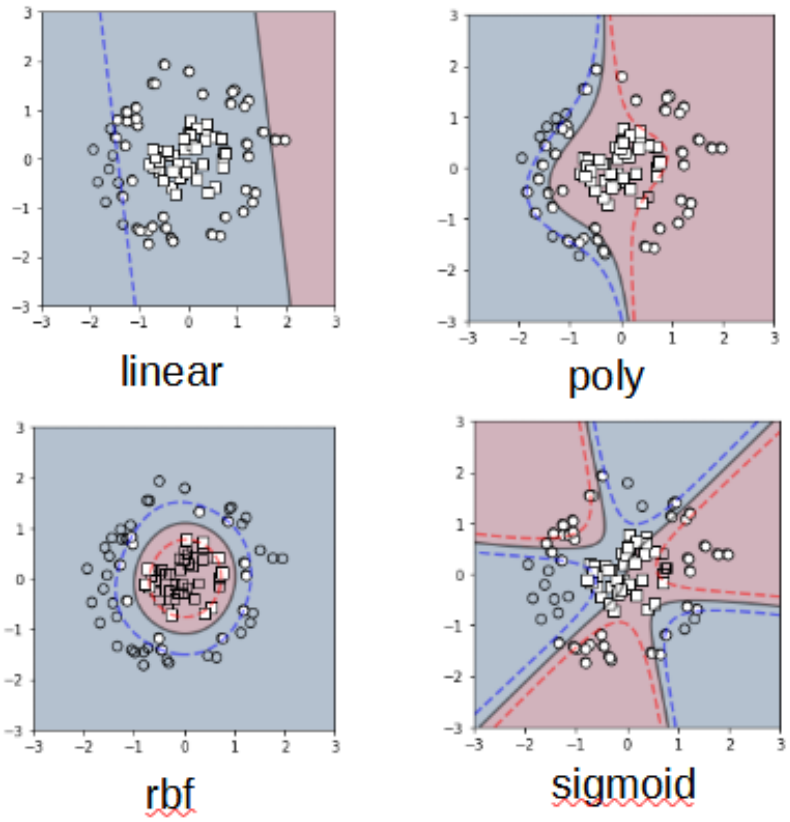

Figure 2: SVM results with different kernels.

2.3 Third Part

The tables below have the example hyperparameters after the normalizations **train_data** = **preprocessing.normalize(train**$_data, norm =' l2')$
$test\_data = preprocessing.normalize(test_data, norm =' l2')$
$The best results occur when kernel = polynomial, gamma = 1 and C = 100.$

gamma	C				
	0.01	0.1	1	10	100
-	0.636444	0.647730	0.686107	**0.701982**	**0.701982**

Table 1: **Linear** kernel where GridSearchCV scoring parameter is **"accuracy"**

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.537635	0.537635	0.537635	0.537635	0.537635
0.0001	0.537635	0.537635	0.537635	0.537635	0.537635
0.001	0.537635	0.537635	0.537635	0.537635	0.537635
0.01	0.537635	0.537635	0.537635	0.537635	0.537635
0.1	0.537635	0.537635	0.537635	0.537635	0.594998
1	0.537635	0.537635	0.537635	0.592950	**0.631346**

Table 2: **RBF** kernel where GridSearchCV scoring parameter is **"accuracy"**

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.537635	0.537635	0.537635	0.537635	0.537635
0.0001	0.537635	0.537635	0.537635	0.537635	0.537635
0.001	0.537635	0.537635	0.537635l	0.537635	0.537635
0.01	0.537635	0.537635	0.537635	0.537635	0.537635
0.1	0.537635	0.537635	0.537635	0.537635	0.581666
1	0.537635	0.581666	0.662571	0.716337	**0.748075**

Table 3: **Polynomial** kernel

2.4 Fourth part

2.4.1 Without handling the imbalance problem

Report test accuracy. Can accuracy be a good performance metric? Report confusion matrix and comment on it. Report additional metrics here if you want.

The test accuracy is 0.83333. Accuracy is (of correctly classified data points) / (total number of data points). It is also (TP+TN)/(Total of examples) if we want to use confusion matrix It is not a good performance metric actually because it works correctly when the number of each samples are same for each class.
sklearn.metrics.confusion_matrix result:

$$\begin{vmatrix} 1 & 190 \\ 0 & 949 \end{vmatrix}$$

If we assume that it is a binary classification and the class labels are "yes" or "no", then:
True Positives(TP) : Prediction is yes and actual class label is yes.
True Negatives(TN) : Prediction is no and actual class label is no.
False Positives(FP) : Prediction is yes and actual class label is no.
False Negatives(FN) : Prediction is no and actual class label is yes.
Recall is out of all the positive classes, how much we predicted correctly. It should be high as possible. Recall= TP/(TP+FN)
Precision is out of all the positive classes we have predicted correctly, how many are actually positive. Precision= TP/(TP+FP)
For this experiment, precision= 0.8331870 and recall=1.0. These results are good

gamma	C				
	0.01	0.1	1	10	100
0.00001	0.537635	0.537635	0.537635	0.537635	0.537635
0.0001	0.537635	0.537635	0.537635	0.537635	0.537635
0.001	0.537635	0.537635	0.537635	0.537635	0.537635
0.01	0.537635	0.537635	0.537635	0.537635	0.601115
0.1	0.537635	0.537635	0.537635	0.602139	**0.629780**
1	0.537635	0.537635	0.585238	0.520220	0.508957

Table 4: **Sigmoid** kernel

since recall is 1. As I can see the precision is similar to accuracy.

2.4.2 Oversampling the minority class

Report your test accuracy, confusion matrix and comment on them.
The number of data points in training dataset whose label is 0 = 860
The number of data points in training dataset whose label is 1 = 925, and the total = 1785.
Now, the test accuracy is 0.796491. When the number of each class gets nearly equal, the accuracy metric ie. (of correctly classified data points) / (total number of data points) decreased. This is not a bad result since the accuracy metric works more correct.
sklearn.metrics.confusion_matrix result:
$\begin{vmatrix} 70 & 121 \\ 111 & 838 \end{vmatrix}$ **Precision= 0.87382**
Recall= 0.883034
Recall has decreased. This can be explained as there occur same data points in the dataset while multiplying the minority class. If the optimizer missclassify the same data points over and over again, the accuracy decreases and recall as as a result. Also, although we copy the data points and increase the training data size, the discriminative samples in the dataset did not increase because the data points have same features.

2.4.3 Undersampling the majority class

Report your test accuracy, confusion matrix and comment on them. The number of data points whose label is 0 = 215
The number of data points whose label is 1 = 215, and the total = 430.
Now, the test accuracy is 0.614912280. Confusion matrix :
$\begin{vmatrix} 106 & 85 \\ 354 & 595 \end{vmatrix}$ **Precision= 0.875**
Recall= 0.62697
The accuracy and recall has decreased compared to previous steps. The reason for that could be the number of data points in the dataset was reduced, as a result there is not much discriminative data points exists anymore and the machine learning algorithm couldn't learn to classify data points well. In ML algorithms training dataset is very

important. Although the number of datapoints for each class is equal, 430 is a very bad quantity for SVM network to classify data points.

2.4.4 Setting the class_weight to balanced

Report your test accuracy, confusion matrix and comment on them.
When we use class_weight="balanced" then we use weights that are inversely proportional to class frequencies. For imbalanced datasets this method could work to get a better accuracy. Confusion matrix :

$$\begin{vmatrix} 87 & 104 \\ 173 & 776 \end{vmatrix}$$

Precision= 0.881818
Recall-4= 0.817702
Accuracy = 0.757017. This accuracy is less than the svm result without class_weight="balanced" (0.833333). I expected to get a higher accuracy since I think the dataset is imbalanced. Also recall is less than the first part(recall=1).