

Kaggle Competition: Do Not Overfit II

Machine Learning Project 20T2

Introduction

I attempted the Kaggle Competition: Do Not Overfit II where the target was to train a dataset in such a way that we avoid any overfitting when we test the model against an irregularly large test set. The training set had only 250 instances whereas the test set had 19500 instances. Also, the data set had 300 features which were values ranging from $[-2,2]$. The target feature was binary (0,1).

Overfitting is when we train our model a bit too well on our training set. This leads to problems when our model comes across entries it has not been trained for before so when you try to predict a value it is more than likely the model predicts a confused output.

Approach

I took a very experimental approach for this problem, I tried numerous classifiers, hyper-parameters, pre-processing techniques, feature-selection techniques and combination of classifiers to tackle this problem. The main classifiers that I could think of which would be best suited to solve a problem like this were:

1. Logistic Regression

A Logistic Regression model works best for a binary classification model as it tries to classify an entry either to the top of the logistic curve or to the bottom of the logistic curve. It works better than a linear regression model as a linear regression model is not bounded and can create a separation anywhere.

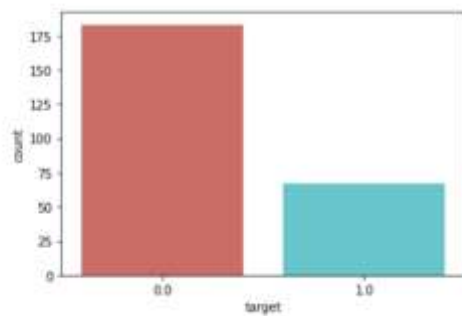
2. Decision Trees

Using DT for binary classification is useful as it creates binary recursive partitioning which is useful when you have only two classes. Also, DTs give us complete freedom to use however many splits and leaves we want to use for our model.

Implementation

I started out by investigating the training dataset. First, I looked at the size of the target of the data set as having an imbalanced target feature would mean the training accuracies would be heavily skewed in favour of the target with more instances.

```
sns.countplot(x = 'target', data = train, palette = 'hls')  
plt.show()  
plt.savefig('count')
```

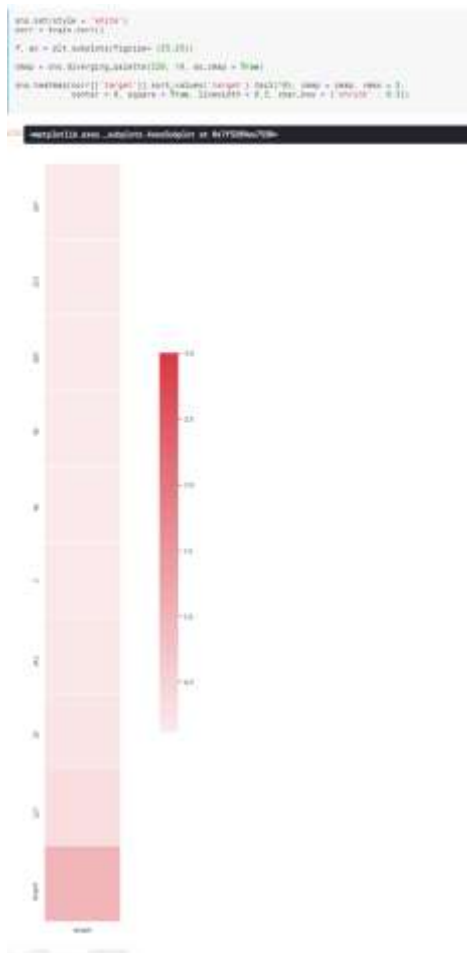


As we can see there were more '0' classes than '1' classes.

This tells us 2 things:

1. This is binary classification problem
2. We would need to balance the data when we are training the complete model to train it equally for both classes.

Then I checked if there are any correlations between the target and any of the features, to check if one or more features have more weight than the others.




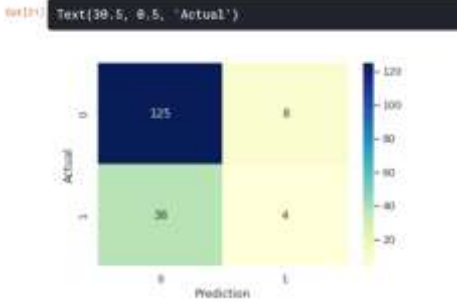


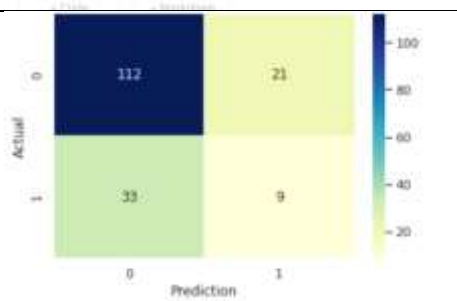
As we can see only feature '127' is somewhat important to the target class, which mean all the other features are almost equally import to target function.


As looking for correlations couldn't serve any purpose, I tried to look for features that are actually important to the target classes. For this I used ELI5 Permutation Importance but this time I broke up the it into training sets first to get a different feel for the data. I broke up the data 70:30(training, test).



As we can see there are few dominating features in the dataset and most other features do not carry enough weigh to affect the target too much. But that's just the training test.

So, after this I carried out an experiment to find the best classifier(s) for this problem. I trained them against only 30% of the training test and tested it against 70% of the training set. Below is the table of my results.

Classifier	Confusion Matrix	Accuracy									
Logistic Regression	 <p>A confusion matrix heatmap for Logistic Regression. The x-axis is 'Prediction' (0, 1) and the y-axis is 'Actual' (0, 1). The values are: True Positives (TP) = 111, True Negatives (TN) = 22, False Positives (FP) = 37, and False Negatives (FN) = 15. A color bar on the right indicates counts from 0 to 100.</p> <table border="1"> <thead> <tr> <th></th><th>Prediction 0</th><th>Prediction 1</th></tr> </thead> <tbody> <tr> <th>Actual 0</th><td>111</td><td>22</td></tr> <tr> <th>Actual 1</th><td>37</td><td>15</td></tr> </tbody> </table>		Prediction 0	Prediction 1	Actual 0	111	22	Actual 1	37	15	72%
	Prediction 0	Prediction 1									
Actual 0	111	22									
Actual 1	37	15									
K-Nearest Neighbours	 <p>A confusion matrix heatmap for K-Nearest Neighbours. The x-axis is 'Prediction' (0, 1) and the y-axis is 'Actual' (0, 1). The values are: TP = 125, TN = 8, FP = 36, and FN = 4. A color bar on the right indicates counts from 0 to 120. Above the plot is a text box: <code>Text(30.5, 0.5, 'Actual')</code>.</p> <table border="1"> <thead> <tr> <th></th><th>Prediction 0</th><th>Prediction 1</th></tr> </thead> <tbody> <tr> <th>Actual 0</th><td>125</td><td>8</td></tr> <tr> <th>Actual 1</th><td>36</td><td>4</td></tr> </tbody> </table>		Prediction 0	Prediction 1	Actual 0	125	8	Actual 1	36	4	74%
	Prediction 0	Prediction 1									
Actual 0	125	8									
Actual 1	36	4									
SVM	 <p>A confusion matrix heatmap for SVM. The x-axis is 'Prediction' (0, 1) and the y-axis is 'Actual' (0, 1). The values are: TP = 102, TN = 31, FP = 27, and FN = 15. A color bar on the right indicates counts from 0 to 100.</p> <table border="1"> <thead> <tr> <th></th><th>Prediction 0</th><th>Prediction 1</th></tr> </thead> <tbody> <tr> <th>Actual 0</th><td>102</td><td>31</td></tr> <tr> <th>Actual 1</th><td>27</td><td>15</td></tr> </tbody> </table>		Prediction 0	Prediction 1	Actual 0	102	31	Actual 1	27	15	67%
	Prediction 0	Prediction 1									
Actual 0	102	31									
Actual 1	27	15									
Decision Trees	 <p>A confusion matrix heatmap for Decision Trees. The x-axis is 'Prediction' (0, 1) and the y-axis is 'Actual' (0, 1). The values are: TP = 97, TN = 36, FP = 32, and FN = 10. A color bar on the right indicates counts from 0 to 90. Above the plot is a text box: <code>Text(30.5, 0.5, 'Actual')</code>.</p> <table border="1"> <thead> <tr> <th></th><th>Prediction 0</th><th>Prediction 1</th></tr> </thead> <tbody> <tr> <th>Actual 0</th><td>97</td><td>36</td></tr> <tr> <th>Actual 1</th><td>32</td><td>10</td></tr> </tbody> </table>		Prediction 0	Prediction 1	Actual 0	97	36	Actual 1	32	10	61%
	Prediction 0	Prediction 1									
Actual 0	97	36									
Actual 1	32	10									
XGB Classifier	 <p>A confusion matrix heatmap for XGB Classifier. The x-axis is 'Prediction' (0, 1) and the y-axis is 'Actual' (0, 1). The values are: TP = 112, TN = 21, FP = 33, and FN = 9. A color bar on the right indicates counts from 0 to 100.</p> <table border="1"> <thead> <tr> <th></th><th>Prediction 0</th><th>Prediction 1</th></tr> </thead> <tbody> <tr> <th>Actual 0</th><td>112</td><td>21</td></tr> <tr> <th>Actual 1</th><td>33</td><td>9</td></tr> </tbody> </table>		Prediction 0	Prediction 1	Actual 0	112	21	Actual 1	33	9	69%
	Prediction 0	Prediction 1									
Actual 0	112	21									
Actual 1	33	9									

Gaussian Naïve Bayes		73%
----------------------	------------------------------------------------------------------------------------	-----

From this experiment I could find even though having high accuracy GNB and KNN were confused about finding target '1' so probably they were already overfitting on the 30% training set. Decision Trees and XGB had lower accuracy but were less confused about '1' class. Logistic Regression performed the best with high accuracy and less confusion.

And after this I tried to do some feature selection using RFE which runs the model recursively and finds out top features that affect the outcome of a predict for a model.

It also predicts the minimum number of features required for a model to make an accurate prediction.

After this I removed the not so useful features from the dataset to prevent over fitting on the dataset I was already on.

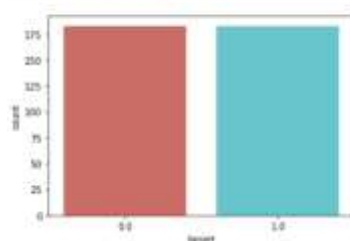
Next, I tried to add data to the data set by using SMOTE which is a tool used to balance datasets. Smote basically adds similar data to the dataset for the minority class to make equal instances for all classes.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy='minority', n_jobs=-1)
X_sm, y_sm = smote.fit_resample(X, Y)

df = pd.DataFrame(X_sm, columns = labels)
df['target'] = y_sm

sns.countplot(x = 'target', data = df, palette = 'hls')
plt.show
plt.savefig('count')
```



After this I used "StackingClassifier" to use multiple models for my prediction model. I also had the option to set Logistic Regression as the meta model to give it more weight than other models. This also meant I could not use SVM as I needed to use "predict_proba" for this feature.

I used LogisticRegression, XGBClassifier and DecisionTreesClassifier for my model.

Results

As this problem was a very challenging exercise due to the difference between training and test datasets, any model on its own was not able to get any sort of viable classification. Most models I tried were stuck on 0.500 score. Even after adjusting hyper-parameters for my models I could not break the 0.510 mark.

But after combining LogisticRegression with XGBClassifier and DTClassifier with the following hyper-parameters:

```
modelLR = LogisticRegression(solver = 'liblinear', C = 0.05, penalty = 'l2', class_weight = 'balanced', max_iter = 10)
modelDT = DecisionTreeClassifier(random_state = 0, max_depth = 5, min_samples_leaf = 8, min_samples_split = 4)
modelXGB = XGBClassifier(max_depth = 2, gamma = 2, eta = 0.8, reg_alpha = 0.5, reg_lambda = 0.5)
```

I could get a score of 0.530 which could be referenced here:

<https://www.kaggle.com/abdullahzaid/kernele2507ae491?scriptVersionId=40419646>

But as this model was not so consistent with its score, I came up another model which is also my final submission for this project which could be found in the Final_submission.py file. This could get a 0.520 consistently.

<https://www.kaggle.com/abdullahzaid/kernele2507ae491?scriptVersionId=40424681>

Conclusion:

In the end, this task proved out to be much more difficult than it looks, just the sheer imbalance makes it a difficult task for anyone.

The score I got for a binary classification is nothing to be proud of but as this was first attempt at making a model for a competition like this and also the magnitude of the task, the time spent trying and testing models was worth while and I got to learn a lot about numerous models, their hyper-parameters and techniques to pre-process data and selecting important features for a problem.

References

Dealing with very small datasets, Kaggle accessed 29/07/20

<<https://www.kaggle.com/rafjaa/dealing-with-very-small-datasets>>

Breaking the curse of small datasets in Machine Learning: Part 1 accessed 31/07/20

<<https://towardsdatascience.com/breaking-the-curse-of-small-datasets-in-machine-learning-part-1-36f28b0c044d>>