HW2 : Drug Prediction

Details:

Name : Siddhanth Kalyanpur
Miner username : mb13
Miner Score : 0.67

Approach:

1. **Importing Libraries and read the Train and Test File.**

2. **Data Preprocessing :**

    - Replace all NaN with 0

    - On Converting the Data to a DataFrame I looped over the length of features in each record and appended 1 where the feature was present for that record and 0 otherwise . This gave us the expected Sparse Matrix for further preprocessing.

    - Smote- Tomec Link tried for under and over sampling but the performance was poor.

3. **Feature Selection :**

    - **TruncatedSVD , PCA :**

      TruncatedSVD does not centre the data before computing the SVD, making it more fruitful to use with sparse data. This is based on 2 algorithms , namely 'Arpack' and Randomized , I have used the later . I have selected the number of components to be 50 as this captures the variability of the dataset as seen below :

```
var_explained = svd.explained_variance_ratio_.sum()
```

```
print(var_explained)
0.999999999999998
```

These factors helped me select TruncatedSVD over PCA which used a more Linear approach for feature selection and hence a poor throughput.

4. Split the Data :

Split Train set to Train(80%) and Test(20%) to model our classifiers performance.

5. Classifier :

Name	Description	Performance Metrics	F1-Score on Train Split
Decision Tree	To Manage the class imbalance assigned weights so the active class labels are oversampled. Allowed the classifier to decide the depth so it could terminate the tree once pure leaves are achieved. Ensured this process doesn't not over fit the data. Also used parameter Hyper tuning using RandomizedSearchCV which identifies the best criterion : gini or entropy for the split	[53]: print(classification_report(y_test,y_hat)) precision recall f1-score support 0 0.95 0.96 0.95 144 1 0.57 0.50 0.53 16 accuracy 0.91 160 macro avg 0.76 0.73 0.74 160 weighted avg 0.91 0.91 0.91 160 Confusion Matrix : [[138,6],[8,8]]	**0.91**
SVM	Used Linear SVM and managed the class weights to counter the imbalance	print(classification_report(y_test,y_hat)) precision recall f1-score support 0 0.92 0.99 0.96 144 1 0.80 0.25 0.38 16 accuracy 0.92 160 macro avg 0.86 0.62 0.67 160 weighted avg 0.91 0.92 0.90 160 Confusion Matrix : [[143,1],[12,4]]	**0.92**

<table>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>Logistic Regression</td><td>Performs the best with penalty to avoid overfitting</td><td><table>
<tr><td></td><td>precision</td><td>recall</td><td>f1-score</td><td>support</td></tr>
<tr><td>0</td><td>0.93</td><td>0.99</td><td>0.96</td><td>144</td></tr>
<tr><td>1</td><td>0.83</td><td>0.31</td><td>0.45</td><td>16</td></tr>
<tr><td>accuracy</td><td></td><td></td><td>0.93</td><td>160</td></tr>
<tr><td>macro avg</td><td>0.88</td><td>0.65</td><td>0.71</td><td>160</td></tr>
<tr><td>weighted avg</td><td>0.92</td><td>0.93</td><td>0.91</td><td>160</td></tr>
</table>
Confusion Matrix : [[143,1],[11,5]]</td><td>0.93</td></tr>
</table>

I have tried more classifiers for prediction but stuck to the above 3 for reporting purpose. The model performance for the remaining can we found out in the code.

6. Conclusion

To review and compare the model performance of all the classifier I have used the ROC_AUC score and the model with the highest area under curve is adjudged the best classifier. An efficient model will be close to the top left corner i.e. (FPR,TPR) = (0,1) and any classifier randomly classifying data will be along the line passing through (0,0) and (1,1). Based on the comparisons made Logistic performs the best .

```
y_predict_proba = logreg.predict_proba(X_test)[:,-1]
roc_auc_score(y_test,y_predict_proba)
```

```
0.9444444444444444
```

```
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show()
```