Winton Wong

Midterm

ISE 364

Introduction

The objective is to create a model with data from project_data.csv and be used to predict target value with new_obs.csv.

Summary

The data set are trained for a Supporting Vector Machine, without applying balancing techniques and with overbalancing and under-balancing techniques, and Linear SVC, K Neighbor Classifier, and Random Forest Classifier. The penalty scores were then compared, and the data set that produced the most accurate results was used to predict the new_obs data.

Experimental Details

- Load the project_data.csv as df
- Conduct Exploratory Data Analysis for project_data.csv
- Convert the V0 to binary data
- Apply Train Test Split to df
- Train and evaluate the model with Linear SVC
- Train and evaluate the model with K Neighbor Classifier
- Train a Support Vector Machine Classifier Model (SVM), in grid, and test it with test data
- Evaluate the model with confusion matrix and classification report
- Resample data for underbalanced case with imblearn.under_sampling
- Train a Support Vector Machine Classifier Model (SVM), in grid, and test it with test data
- Evaluate the model with confusion matrix and classification report
- Resample data for overbalanced case with from imblearn.over_sampling
- Train a Support Vector Machine Classifier Model (SVM), in grid, and test it with test data
- Evaluate the model with confusion matrix and classification report
- Train and evaluate the model with Random Forest in Grid
- Used the best model to predict the targets from new_obs.csv

Results and discussions

The metric used to evaluate this project is the penalty score = **50 (number of False Positives) + 10 (number of False Negatives),** as outlined in the descriptions. It is the last number in each result (Some results don't have it because I forgot to add them, and it takes too long to retrain).

Supporting Vector Machine without balancing

```
{'C': 10, 'gamma': 1, 'kernel': 'rbf'}
[[  78  255]
 [   6 1311]]
              precision    recall  f1-score   support

           0       0.93      0.23      0.37       333
           1       0.84      1.00      0.91      1317

    accuracy                           0.84      1650
   macro avg       0.88      0.61      0.64      1650
weighted avg       0.86      0.84      0.80      1650

12810
```

Supporting Vector Machine with under-balancing

```
{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
[[218 115]
 [373 944]]
              precision    recall  f1-score   support

           0       0.37      0.65      0.47       333
           1       0.89      0.72      0.79      1317

    accuracy                           0.70      1650
   macro avg       0.63      0.69      0.63      1650
weighted avg       0.79      0.70      0.73      1650
```

Supporting Vector Machine with overbalancing

```
{'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
[[ 154  179]
 [ 199 1118]]
              precision    recall  f1-score   support

           0       0.44      0.46      0.45       333
           1       0.86      0.85      0.86      1317

    accuracy                           0.77      1650
   macro avg       0.65      0.66      0.65      1650
weighted avg       0.78      0.77      0.77      1650
```

K Neighbor Classifier

```
[[  83  250]
 [ 100 1217]]
              precision    recall  f1-score   support

           0       0.45      0.25      0.32       333
           1       0.83      0.92      0.87      1317

    accuracy                           0.79      1650
   macro avg       0.64      0.59      0.60      1650
weighted avg       0.75      0.79      0.76      1650

13500
```

Linear SVC

```
[[   4  329]
 [   5 1312]]
              precision    recall  f1-score   support

           0       0.44      0.01      0.02       333
           1       0.80      1.00      0.89      1317

    accuracy                           0.80      1650
   macro avg       0.62      0.50      0.46      1650
weighted avg       0.73      0.80      0.71      1650

16500
```

Random Forest

```
[9] print(rf_random.best_params_)

    {'n_estimators': 1000, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': None, 'bootstrap': False}
```

```
[10] predictions = rf_random.predict(X_Test)
     cm = confusion_matrix(y_test, predictions)
     print(cm)
     print(classification_report(y_test, predictions))

     print(penaltyScore(cm))
```

```
     [[ 147  186]
      [  35 1282]]
                   precision    recall  f1-score   support

                0       0.81      0.44      0.57       333
                1       0.87      0.97      0.92      1317

         accuracy                           0.87      1650
        macro avg       0.84      0.71      0.75      1650
     weighted avg       0.86      0.87      0.85      1650

     9650
```

Body

After the data was loaded into data frames, Exploratory Data Analysis was conducted to attempt to discover trends, to check for nulls, data types, etc. The data has 12 different type of characteristics, and the target was binary of 1s and 0s. However, one of the features contains unique values of As and Bs, and it had to be converted to was binary of 1s and 0s using dummies variables.

The data set was then split with 30% for testing after training the model.

The logical model of choice is Supporting Vector Machine, after consulting the scikit-learn cheat sheet (1). This set of data contains more than 50 samples, we are predicting a category of 1 or 0, the data are labeled. The information gives a path that first led to Linear SVC, the K Neighbor Classifier, and then SVC. As shown in the results, both Linear SVC and K Neighbor Classifier did not perform as well as SVC. (Both models can be retained using linearSVC.py and k_neighbors.py)

Following the data clean up and setting up for training. Then the grid was paired with the Supporting Vector Machine to find the best combinations the parameter C and gamma that would produce the best result of all SVM with a penalty score of 12810, C and gamma that produced the best results are 10, and 1.

It was discovered that the ratio of target 1s are 4 times more than 0s in target. Although the model was retrained with under balancing and overbalancing in mind afterwards, it appears such data engineering methods did not produce a better result pre outlined in the results sections.

In an exploratory attempt, the data was then again trained using random forest models using random grid search and random parameters. Two of such was done, one with less tries and one with more tries, and the reason one took less time to train. The random forest with the following parameters produced the lowest penalty score of 9650 out of all the models: {'n_estimators': 1400, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 90, 'bootstrap': False}. And therefore, this model used to predict the new_obs data.

Conclusions

Although the cheat sheet suggested using the SVM and it did not disappoint, the random forest model through random grid search produced a slightly better penalty score, therefore the parameter was preserved and used to produce the prediction for the new_obs.csv data.

Recommendations

It would be nice if there is a description of the different features, then each feature can be deeply analyzed and to be used to the advantage. Models can be further tuned with more and different parameters if time allows.

Files

random_forest.py

- A python script for a random forest model

Midterm_ISE364_WintonWong.ipynb

- An ipython notebook where the SVM models, without balancing are contained, as well as my thoughts and logic

svm.py

- A python script for the unbalanced svm model

linearSVC.py

- A python script for the linear SVC model

k_neighbors.py

- A python script for the k neighbors classifier model

finalized_svm_model.sav

- This is the saved model of SVM without any balancing

RandomForest_SVM_Grid_Midterm.ipynb

- An ipython notebook where I compared the SVM models and the 2 random forest with random grid search, and determined that random forest is the better candidate

make_prediction.py

- This script uses the specified parameters to predict new_obs.csv data and produce the predictions.csv

Reference

1. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html