University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

CS6405 Data Mining Project Report

Assignment 2

The Boolean satisfiability (SAT) problem and hands-on experience in using a practical machine learning tool and analysing scientific results

James Henry Hehir

April 21st 2021

Table of Contents

Contents

Basic Model and Evaluation

This section deals with the KNN and Random Forest classifiers and how they are implemented on the Boolean Satisfiability Problem dataset. Initially the KNN classifier was implemented but both models have the same approach in reading in the data. The code below demonstrates the process:

```
df = pd.read_csv('dataset-sat.csv')

X = df.drop(columns=['ebglucose_solved'])
y = df['ebglucose_solved'].values
```

From scikit learn the classifiers are also imported.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

The data is split into features and a label is selected. The label is called 'ebglucose_solved'. X is all the columns bar the label which are the features and the Y is column called 'ebglucose_solved'. This data is then split into training and testing data which is 70% and 30% respectively. The code below illustrates this:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

KNN

The code below shows the KNN classifier in action.

```
# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors = 3)

# Fit the classifier to the data
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)

score = knn.score(X_test, y_test)
```

The score from this was 0.6314285714285715. This was expected as KNN is not suited to large data as it can be slow and also irrelevant features can affect the score. Below is the confusion matrix and recall.

```
[[ 64  29  11]
 [ 35  49  14]
 [ 30  10 108]]
              precision    recall  f1-score   support

           0       0.50      0.62      0.55       104
```

```
           1       0.56      0.50      0.53        98
           2       0.81      0.73      0.77       148

    accuracy                           0.63       350
   macro avg       0.62      0.62      0.61       350
weighted avg       0.65      0.63      0.64       350
```

Random Forest

The code below shows the Random Forest classifier in action.

```
RFC=RandomForestClassifier(n_estimators=1000)
RFC.fit(X_train,y_train)
y_pred=RFC.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

The accuracy was 0.8028571428571428. This was a high score as this model makes decisions based on a sequence of questions asked about the features vales. The model can sometimes overfit because of low bias and high variance. Below is the confusion matrix and recall.

```
[[ 76  19   8]
 [ 11  67  19]
 [ 12  11 127]]
              precision    recall  f1-score   support

           0       0.77      0.74      0.75       103
           1       0.69      0.69      0.69        97
           2       0.82      0.85      0.84       150

    accuracy                           0.77       350
   macro avg       0.76      0.76      0.76       350
weighted avg       0.77      0.77      0.77       350
```

Robust Evaluation

In this section I took the KNN from the first part of this report and initially tried the cross validation. The code for this is below:

```
#create a new KNN model
knn_cv = KNeighborsClassifier(n_neighbors=3)

#train model with cv of 5
cv_scores = cross_val_score(knn_cv, X, y, cv=5)

#print each cv score (accuracy) and average them
print(cv_scores)
print('cv_scores mean:{}'.format(np.mean(cv_scores)))
```

```
cv_scores mean:0.6227504493598914
```

The result from the cross validation compared to the hold-out score was similar but slightly lower. This was unusual as the cross validation gives the model multiple train-test splits. However the hold-out method can be more useful on large data sets.

Decision Tree, SVM, Linear Regression and Naïve Bayes

I tried implementing four other classifications which were Decision Tree, SVM, Linear Regression and Naïve Bayes. The table below shows the scores from these classifications.

Classification	**Score**
Decision Tree	0.7905982905982906
SVM	NA
Linear Regression	0.4955147427422846
Naïve Bayes	0.3142857142857143

As the table suggests that the Decision Tree classifier is the best as it forces the consideration of all possible outcomes of a decision and leads to a conclusion. Also the SVM classifier would not finish and seemed to run forever. This could have been a coding issue but it is known that large datasets perform poorly on SVM. Linear Regression and Naïve Bayes were poor performers. In Linear Regression I believe that data may have had several outliers. The linear regression classifier can be sensitive to these outliers. The lack of independence of the explanatory variables may be the issue for the poor performance in the Naïve Bayes classifier.

Hyper-parameter Tuning

In this section we experimented with the parameter values in KNN known as the best value for n neighbours and best value for leaf size. In both tests the best value was 1.

```
#List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors)
#Create new KNN object
knn_2 = KNeighborsClassifier()
```

```
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
best_model = clf.fit(X,y)
#best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

Feature selection and engineering

The feature selection was large for this dataset and it was hard to do much change to the output. The INSTANCE_ID column was the first suspect as it is just like an index. This column was removed, however due to the large dataset it was irrelevant to the overall score.

Performance Metrics

The performance metrics that were explored in this experiment include the confusion matrix, classification report and the accuracy score. In KNN the scores are listed below and show how the KNN classifier performs.

```
[[ 64  20  20]
 [ 33  44  21]
 [ 29   9 110]]
              precision    recall  f1-score   support

           0       0.51      0.62      0.56       104
           1       0.60      0.45      0.51        98
           2       0.73      0.74      0.74       148

    accuracy                           0.62       350
   macro avg       0.61      0.60      0.60       350
weighted avg       0.63      0.62      0.62       350

Accuracy: 0.6228571428571429
```

The confusion matrix gives us information on how well the classifier works. The precision column shows the ratio of correct data to the total predicted. Recall is positive results divided by relevant samples and the f1-score is the weighted average of precision and recall.

Feature Normalization

The KNN classifier is dependent on majority voting based on points nearest to the given test point. In this dataset it would be useful to normalize the label feature. The normalization scalar built into scikit learn does the hard work and scales the data. The results shows a small increase in overall score.

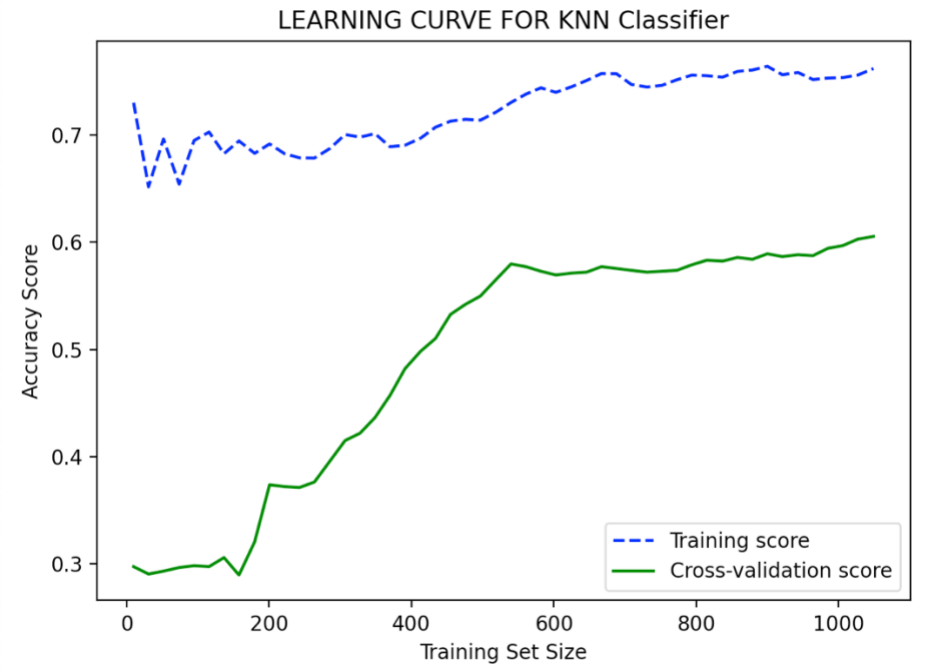
```
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
```

```
X_test = scaler.transform(X_test)
```

The score was 0.64. This was unusual as it was expected to give a higher score but this could be explained due to the large dataset.

Learning Curve Analysis



The learning curve shows that when the training size is increased that both the training score and cross-validation score increase in accuracy. Overall learning curves are ideal for finding bias and variance in machine learning algorithms. This plot indicates that the KNN classifier has low bias and high variance on the dataset.

Conclusion

Overall this project has shown the power and use of machine learning to understand large datasets. The Boolean satisfiability (SAT) is an unique problem and the KNN classifier was useful to understand this dataset. In the robust evaluation the methods used to evaluate the classifier and dataset were challenging but illustrate the unique features of scikit learn and how it can be used to better understand large datasets.