Harry Ammon 2 May 2019

Solution Explained

Overview

My solution was developed using Atom.io as a text editor and by installing SKlearn and Pandas on to my computer. The computer I used to come up with a solution had a dual core processor, this impacted the runtime of my solution and motivated me to find a solution that had as short a run time as possible.

My solution used the RandomForestClassifier with hyper-parameters that been calculated to best suited by using the GridSearchCV method. This was done in another file to decrease the runtime of my solution. I was able to achieve an accuracy score of 95.5%.

My solution was obtained by developing 2 separate scripts. The first (bestParameters.py) used GridSearchCV which I used to determine the best hyper-parameters for my chosen model. The second(solution.py) implemented these hyper-parameters and then found an accuracy score using ShuffleSplit.

Data Processing for GridSearchCV

The dataset was read into Python using the read_csv method provided by pandas. This data was then split into data(X) and answer(Y). The data consisted of the first 57 columns and answer was contained in the final column. To separate this data I first calculated the number of columns in the dataset. Then, using the values attribute provided by pandas that implements Python slice notation, I was able to separate the dataset as shown below.

```
data = pd.read_csv("dataset.csv")

numCol = len(data.columns)
X = data.values[:,0:numCol-1]
Y = data.values[:,-1]
```

When using this data with GridSearchCV the train test split method was used with a random state of 42 and a test size of 0.3. This test size was chosen as having a smaller test size caused the accuracy to decrease potentially due to an issue of overfitting. After this the data was scaled using the StandardScaler function so that the data will have a standard deviation of 1 and a mean value of 0.

Implementing GridSearchCV

First, I had to decide on what classifier to use. I first chose to use MLPClassifer as I knew how powerful Artificial Neural Networks can be. However, I ended up settling with using the RandomForestClassifier as it seemed to be able to predict an outcome much faster than MLPClassifer. Also, when using GridSearchCV RandomForestClassifier was quicker for searching through hyper-parameters due to how it is less complex than MLPClassifer. This reduced the runtime of my program significantly.

Then I had to settle on what hyper-parameters to use, I settled on these:

```
clf = RandomForestClassifier()

parametersRF = {
    'n_estimators': [10,100,500,1000],
    'bootstrap': [True,False],
    'criterion': ['gini','entropy'],
    'max_features':['auto','sqrt','log2'],
    'max_depth': [4,20,50]
}
```

I used the GridSearchCV method using a 5-fold Stratified K-Fold as the cross-validation method. This consistently returned the following parameters as the best:

- n_estimators: 100
- boostrap: false
- max_features: log2
- max_depth: 50
- critertion: entropy

These parameters where then used in the second script that calculated the accuracy.

Data Processing for ShuffleSplit

Like with GridSearch, I first used pandas to read in the dataset, and then split the data into X and Y using python slice notation.

When calculating the final accuracy, the data was split using the ShuffleSplit function with a test size of 0.3. This along side with a n_split of 5 was used to ensure that overfitting did not occur. The X data had also been scaled using the StandardScaler function.

```
X = data.values[:,0:numCol-1]
Y = data.values[:,-1]

scl = StandardScaler()
scl.fit(X)
scl.transform(X)
```

Implementing ShuffleSplit and cross_val_score

ShuffleSplit was chosen over KFold due to the method it used meaning that the runtime would be quicker. This is because KFold used 1 fold as the test set and then n-1 folds as the train set. However, Shufflesplit will only use the train and test split generated for each fold. This means that the same data could be used multiple

times. Cross_Val_score was used as a convenient way to combine both the cross_validation method used and the chosen classifier. This is then able to provide accuracy scores for each n split and then the average score. This is how I achieved an average accuracy score of 95.6% with the accuracy score for separate n split often being above 96%.

```
clf = RandomForestClassifier(n_estimators=100,bootstrap=False,max_features="log2",max_depth=50,criterion='entropy')

cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=42)

scores = cross_val_score(clf, X, Y, cv=cv,scoring='accuracy')
```