Assignment 2: Choosing the Best Parameters to Use for a Binary KNN classifier using on 5-fold cross-validation

Dalia Ibrahim[1] and Carlos Dasaed Salcedo[2]

Studnet ID: [1]201893217, [2]201892008

February 15, 2019

1 Introduction

For this assignment, we have implemented a cross-fold validation algorithm from scratch using Python and the following libraries: pandas, numpy, math, random, and sys. To improve the efficiency of the algorithm, unsupervised filtering was also used based on the correlation matrix and variances of the data using sklearn libraries. The other sklearn libraries included in the final program were used to calculate KNN, and related precision metrics. To run the program, the following line must be executed from the command line in Linux:

```
$python3 A2_t2.py [DataFile.tsv]
```

2 Dimensionality reduction

Dimensionality reduction is the process of reducing the number of predictor variables (features) included in a model by eliminating some features, and this step is recommended for multiple reasons. Firstly, making learning algorithm faster, so fewer features mean high improvement in terms of speed. Secondly, according to Occam's razor, he mentioned A simpler model is preferable.

The given data has 348 features, so the unsupervised feature selection is made to reduce the number of features by performing two consecutive steps which are removing correlated features then removing features which have low variance.

2.1 Drop Highly Correlated Features

The correlation matrix is calculated to show too correlated features. The correlated features mean that they bring the same information.

The threshold equal to 0.80 is used, and it reduces the features from 348 to 305 features.

2.2 Removing low-variance features

The features with low variance mean the values across all observation does not change a lot. For example, if the variance equals zero means, this feature has the same value, so it does not add any new information to the model. So the feature does not meet the varaince threshold which equals 0.9 will be removed.

3 Main Pseudo-code

This section only includes the functions that are relevant and currently being used by the algorithm. Functions, such as the ones from sklearn, will only be mentioned in the pseudo-code, but will not be individually described.

3.1 FoldSplitter()

3.2 Cross Validation ()

Algorithm 1 Euclidean Distance Function

```
Require: TrainingData, Kfolds
1: $NumofselectedFeatures \leftarrow Dimensionality_Reduction_step$
2: $n_neighbors = [3, 5, 7, ..., 30]$
3: Divided data into Kfold Using Split Function
4: for iteration from 1 to Kfold do
5:   $X_train, X_test, y_train, y_test \leftarrow SplitData(foldDict, iteration)$
6:   for $P in range(NumofselectedFeatures)$ do
7:     $NewX_train \leftarrow X_train[:, 0 : P + 1]$
8:     $NewX_test \leftarrow X_test[:, 0 : P + 1]$
9:     for $P in n_neighbors$ do
10:      $knn \leftarrow KNeighborsClassifier(n_neighbors = K)$
11:      $knn.fit(NewX_train, y_train)$
12:      $PredictedOutput \leftarrow knn.predict(NewX_test)$
13:      $probs \leftarrow knn.predict_proba(NewX_test)$
14:      $probs \leftarrow probs[:, 1]$ {use probabilities for class=1}
15:      $aucValue \leftarrow roc_auc_score(y_test, probs)$
16:      $row_index \leftarrow n_neighbors.index(K)$
17:      $col_index \leftarrow P$
18:      $ResultGrid[row_index, col_index] \leftarrow ResultGrid[row_index, col_index] + aucValue$
19:    end for
20:  end for
21: end for
22: $ResultGrid \leftarrow ResultGrid/Kfold$ { calculating average AUC}
    { Find top 3 AUC values and their index}
23: $top_n = 2$
24: $topbestmodels \leftarrow [[-1, -1, -1], [-1, -1, -1], [-1, -1, -1]]$
25: for $K in range(3)$ do
26:   for $i, row in range(ResultGrid)$ do
27:     $top = row.nlargest(top_n).index$
28:     for $topCol in top$ do
29:       if $ResultGrid[i, topCol] > topbestmodels[k][0]$ then
30:         $topbestmodels[k][0] = ResultGrid[i, topCol]$
31:         $topbestmodels[k][1] = i$
32:         $topbestmodels[k][2] = topCol$
33:         $ResultGrid.loc[topbestmodels[k][1], topbestmodels[k][2]] = np.NAN$
34:       end if
35:     end for
36:   end for
37: end for
    { Find worest 2 models }
38: $lowest_n = 2$
39: $Worestmodels = [[1000, -1, -1], [1000, -1, -1]]$
40: for $K in range(2)$ do
```

4 Loss Function

The Area Under the Curve (AUC) is calculated, and the Model with the highest AUC will be considered as the best model. and the model with lowest AUC will be the worst model.

5 Deciding on Performance

Since there are several different performance metrics that can be used to determine how good an algorithm will perform, we decided to use the AUC of the ROC curve

To prevent possible ties, random numbers without repetition are generated and assigned to each of the selected closest neighbors. In the case of a tie in the number of votes, the class of the neighbor with the smallest random number would be selected.
The following modifications were implemented and tested in an effort to improve the algorithm.

1. Normalization of the training and testing data
2. Addition of weights to the neighbors to improve voting in the selection of the closest neighbor
3. Normalization and Addition of weights together.

6 Results

To validate the accuracy of our algorithm, we used crossed validation with the help of the sklearn library. These are the steps we followed to implement Cross Validation and test our code:

1. Shuffle the data using *trainingdata.sample*() function from sklearn Library
2. Split the data using *trainTestSplit* function in the sklearn Library and make the testing part equal to 40% of the total training data.
3. Calculate average accuracy by comparing actual data with the predicated data, and then applying the formula in (1)

$$Accuracy = \frac{CorrectPredicted}{AllOutputs} \quad (1)$$

4. Generate a report with Precision and Recall using the *classificationReport* function from the sklearn Library
5. Repeat all pervious steps for Kfold = 5

7 Conclusion

Table 1 shows the results of running KNN with serveral modifications, and K=3.

Method	**Accuracy**
Classical KNN	85.00%
Classical KNN with TieBreaker	**100%**
KNN Weighted	84%
Normalized KNN	91.00%
Normalize KNN Weighted	79.00%

Table 1: Accuracy calculated using sklearn.

8 Conclusion

Since we were using the sklearn to validate our results, we were able to generate a table to compare the accuracy of the results of the algorithm as we made and applied the different modifications. However, to our surprise, Classical KNN with a tie breaker function added yielded accuracy reaches up to 100%. Normalizing the data and adding weights actually reduced our accuracy to 79%.