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 Preliminary Steps - Feature Selection

feature selection
cross correlation
low variance

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()

Algorithm 1 Split Data in K folds

Require: kfolds {kfolds = 5 was used}
$data \leftarrow dataframe(DataFile.tsv)$
$class0 \leftarrow$ data.where(class=0).ShuffleRows()
$class0partition \leftarrow RowsInclass0/kfolds$
$class1 \leftarrow$ data.where(class=1).ShuffleRows()
$class1partition \leftarrow RowsInclass1/kfolds$
$leftOvers0 \leftarrow$class0 rows after(class0partition * cvfolds)]
$leftOvers1 \leftarrow$class1 rows after(class1partition * cvfolds)]
$leftOvers \leftarrow concatenation(leftOvers0, leftOvers1)$
$theFolds \leftarrow newPythonDictionary$
for i=0 **to** kfolds-1 **do**
    $createfold$**i** {i = corresponding iteration in the for cycle}
    $class0Range \leftarrow$class0[**from** i * class0partition **to** (i * class0partition)+class0partition
    $class1Range \leftarrow$ class1[**from** i * class1partition **to** (i * class1partition)+class1partition
    $fold$**i** $\leftarrow class0range + class1range$
    if $i = kfolds - 1$ **then**
        $fold$**i** $\leftarrow fold$**i** $+ leftOvers$
    end if
    $theFolds[fold$**i**$] = TempData$
end for
return theFolds

3.2 SplitData()

Algorithm 2 Create Learning and Training Data

Require: DictionaryOfFolds , iterNum

$testFold \leftarrow fold\{iterNum\}$ {iterNum refers to the iteration number, such that testFold will get fold1,fold2,...,fold5}
$testDF \leftarrow DataFrame(testFold)$
$trainDF \leftarrow NewDataFrame$
for foldKey, foldValue **in** DictionaryOfFolds **do**
  if $foldkey = testFold$ **then**
    Skip this Iteration
  end if
  $trainDF.append(foldValue)$
end for
$xtraining \leftarrow$ trainDF without the last column
$ytraining \leftarrow$ testDF without the last column
$xtesting \leftarrow$ trainDF with only the last column
$ytesting \leftarrow$ testDF with only the last column
return xtraining, xtesting, ytraining, ytesting

3.3 Calculate_Nearest_neighbour()

Algorithm 3 Calculate Nearest Neighbor Function

Require: topMatched , K

$TieBreaker \leftarrow NewDataframe()$
$TieBreaker \leftarrow UniqueRandomNumbers()$
$topMatched.CreateColumn('Count') \leftarrow Count('Class')$
$concatenate(topMatched, TieBreaker)$
$topMatched = topMatched.Filter('Count' = Count.Max())$
$topMatched.sortBy('tieBreaker')$
$TopMatch \leftarrow topMatched.row(0)$
$Probability \leftarrow \frac{topMatch[Count]}{K}$
$FinalOutput \leftarrow [TopMatch[Class], Probability)]$
$returnFinalOutput$

4 Deciding on Performance

Since there are several different performance metrics that can be used to determine how good an algorithm will perform, it is important to make an effort to chose an adequate performance metric for the task at hand. Upon inspection of the training data, it became obvious that the amount of zeros drastically outnumbered the amount of ones by ratio of about 10 to 1. This meant that blindly using accuracy as the factor to determine the best model would be insufficient, since an algorithm that always predicts the output class to be zero would actually be correct 90% of the time. Therefore, to determine which model was best, we decided to focus on identifying instances of class 1. In other words, we considered ones as positive, and zeros as negatives to build the ROC curves, determine our loss function and select our best model

5 Results

To validate the accuracy of our algorithm, we used the sklearn metrics library, which allowed us to generate image 1 and image 2

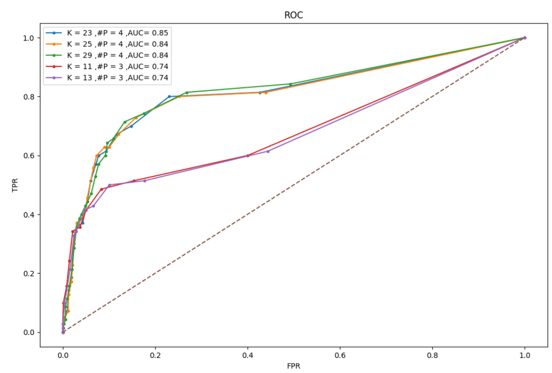

Fig 1 ROC Curve for the 3 best models and the 2 worst models

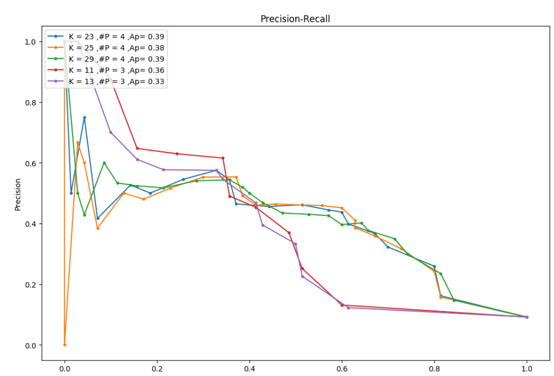

Fig 2 Precision-Recall Curve for the 3 best models and the 2 worst models

6 Conclusion

KNN is a powerful and useful machine learning classifier. However, just like any other classifier, without the proper parameters such as an adequate feature selection, number of neighbors, and correct training set, the model can easily become skewed or flawed.

For this reason, it is important to select the proper performance metrics and to run cross validation tests with at least 5 folds. In our particular case, using 5 fold cross validation, and selecting the features based on a correlation matrix and variance, consistent models with AUCs of up to 0.85 were obtained.