CS412 Project: Don’t get kicked!

By Guangyu Zhou and Qi Zhang

Introduction

The challenge of this competition is to predict if the car purchased at the Auction is a Kick (bad buy). We use two algorithms: K-nearest neighbor and Naïve Bayes.

K-Nearest Neighbor classification:

1. Description of algorithm:

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function.

Since the three distance measures (Manhattan, Euclidean and Minkowski) are only valid for continuous variables, in the instance of categorical variables the Hamming distance must be used.

It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset.

2. Data Pre-processing:

Based on the above constraints, the car auction data has to be processed to satisfy the requirement. Firstly, we use Weka to smoothing the missing values by either assigning the average if it is a numeric attribute or assigning the mode of the value if it is a category value. Afterwards, those invalid or NULL value of the numeric data are filled with average of the current attribute to minimize the prediction error.

3. Classification Implementation:

The KNN algorithm is implemented in Python 2.7, with the external import of “numpy” library only.

The whole algorithm is broken into four parts as below:

1) Handle Data: Open the dataset from CSV to store into test/train datasets (list)
2) Similarity: Calculate the distance between two data instances.
3) Neighbors: Locate k most similar data instances.

4) Response: Generate a response (Classify by vote from k neighbors) from a set of data instances.

For the 1st part, the key thing is to differentiate the numerical data and categorical data (nominal data). For numerical data, it is converted to float; otherwise, they are stored as string. Also, since the distances between both numerical and nominal data have to be compatible, the normalization of the numeric data is utilized as below:

```
for y in num_idx:
    for x in range(1, len(dataset)):
            dataset[x][y] = (dataset[x][y]-min_x)/(max_x-min_x)
```

For the similarity part, I define 2 distance calculating functions, namely Euclidean Distance and Hamming Distance functions:

```
for x in num_idx:
    #print("x",x, testInstance[x-1], trainingSet[x])
    distance += pow((testInstance[x-1] - trainingSet[x]), 2)

for x in cat_idx:
    if testInstance[x-1] == trainingSet[x]:
        distance += 0
    else:
        distance += 1
```

As for finding k nearest neighbors, a heapsort is used to find the k minimal distances with the dataset entry as a form of tuple and return a list of such tuples.

```
def heapSearch( tupArray, k ):
    heap = []
    for item in tupArray:
        #print "item",item[1]
        if len(heap) < k or item[1] > heap[0][1]:
            if len(heap) == k: heapq.heappop( heap )
            heapq.heappush( heap, item )
    return heap
```

After that, the getResponse function will count the vote of classification labels from the k neighbors and out put the result.

4. Chosen of Parameters:

KNN algorithm is known as a non-parametric algorithm. The only parameter that need to be set is k. After comparing different results based on Weka, we choose to use k = 3.

5. Improvement Analysis:

The main problem of KNN is the efficiency with big size of training data and memory consuming. As the memory is not the focus for our improvement, we paid great attention on how to improve time complexity.

One thing we have improved is the sorting of distance. Since we only need to find the k nearest neighbor, we replace the original O(nlogn) sort with heap sort with running time O(nlogk). We also eliminate noise data by doing data preprocessing.

Based on research, we find that a K-D tree data structure can be utilized due to the curse of dimensionality. With that, the distance in 32 dimensions can be calculated faster by pruning the tree.

Another potential improvement that we yet have time to do is do PCA on the training data. It can on the one hand improve the predication accuracy and on the other hand reduce time complexity.

Naïve Bayes Algorithm

1. Description of algorithm

   Naïve Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

   Abstractly, the probability model for a classifier is a conditional model $p(C|F_1,\dots,F_n)$ over a dependent class variable C with a small number of outcomes or *classes*, conditional on several feature variables F_1 through F_n. The problem is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable.

   Using Bayes' theorem, this can be written

$$p(C|F_1,\dots,F_n) = \frac{p(C)\ p(F_1,\dots,F_n|C)}{p(F_1,\dots,F_n)}.$$

   Therefore, by comparing probabilities of different classes we can decide prediction.

2. Data Pre-processing

   For data pre-processing, I fill missing values with mode and NULL values with means.

3. Classification Implementation

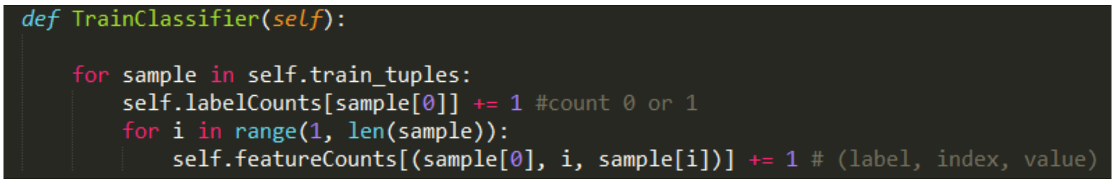
   Naïve Bayes is implemented in Python 2.7.

   Main steps are:

   i. Training Classifier

   ii. Classifying Test Data

   i. Training Classifier

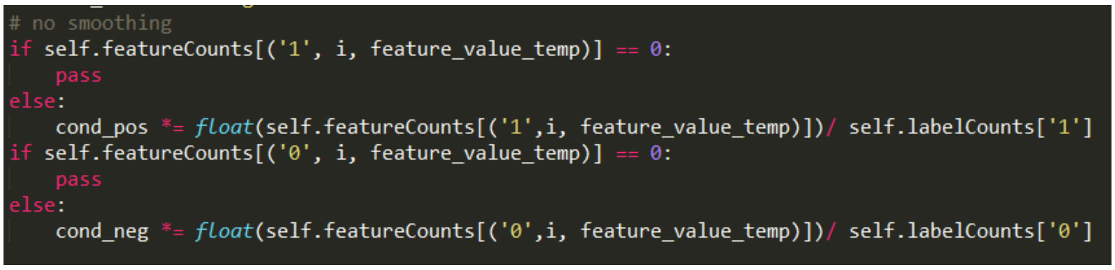


```python
def TrainClassifier(self):

    for sample in self.train_tuples:
        self.labelCounts[sample[0]] += 1 #count 0 or 1
        for i in range(1, len(sample)):
            self.featureCounts[(sample[0], i, sample[i])] += 1 # (label, index, value)
```

   Read in training data and count the numbers of labels and (label, attribute, value) tuples.

   ii. Classifying Test Data



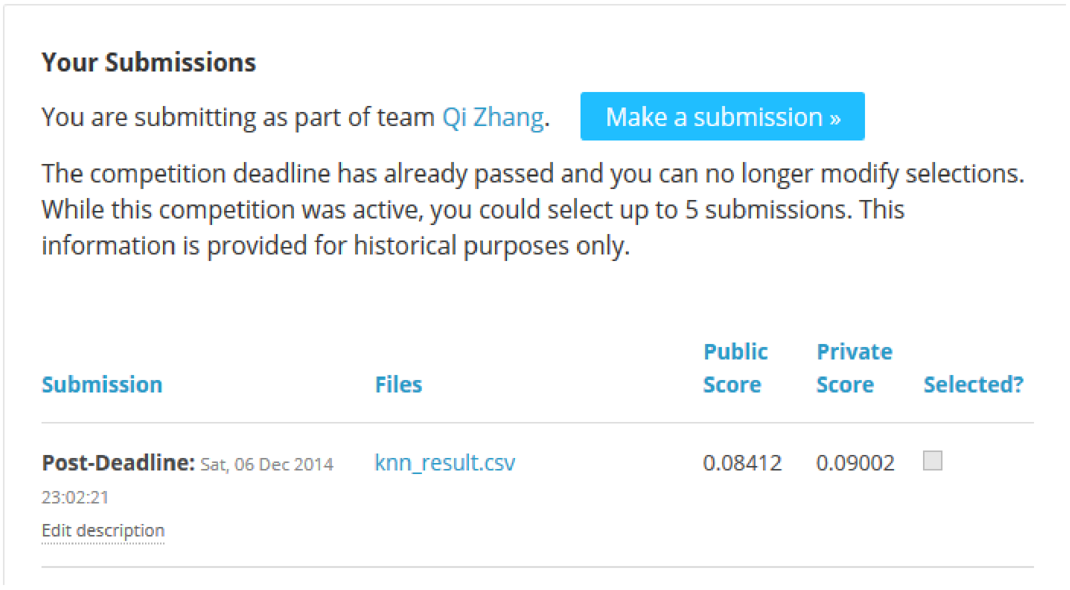
```python
# no smoothing
if self.featureCounts[('1', i, feature_value_temp)] == 0:
    pass
else:
    cond_pos *= float(self.featureCounts[('1',i, feature_value_temp)])/ self.labelCounts['1']
if self.featureCounts[('0', i, feature_value_temp)] == 0:
    pass
else:
    cond_neg *= float(self.featureCounts[('0',i, feature_value_temp)])/ self.labelCounts['0']
```

   Calculating posterior for each class (0 or 1) and predicting it as the class with larger probability. Notice that I do not do any smoothing here. When a new unseen value of an attribute occur, to avoid zero probability being generated, I simply ignore that term.

4. Improvement Analysis:
   No improvement was done on this algorithm because the algorithm we focus on is KNN, which generates much better results than our Naïve Bayes does.

Highest Score

Our K-nearest neighbor algorithm gives the best result



Your Submissions

You are submitting as part of team Qi Zhang.

The competition deadline has already passed and you can no longer modify selections. While this competition was active, you could select up to 5 submissions. This information is provided for historical purposes only.

Submission	Files	Public Score	Private Score	Selected?
Post-Deadline: Sat, 06 Dec 2014 23:02:21 Edit description	knn_result.csv	0.08412	0.09002	☐

Distribution of labor:

We have a team of 2 people, Guangyu Zhou (gzhou6) and Qi Zhang (qizhang4). We worked together on discuss and analyze how to smooth the data and which algorithm to pick up.
After that, each of us implemented an algorithm. Qi implemented Naïve Bayes and Guangyu implemented K-Nearest Neighbor, which includes data preprocessing, algorithm learning and coding, evaluation based on Kaggle and improvement.
The report is collaborated with both of us.