Machine Learning, Fall 2019: Project 1
out on 9/5/19 - due on 9/19/19 by noon on Blackboard

Guantao Zhao

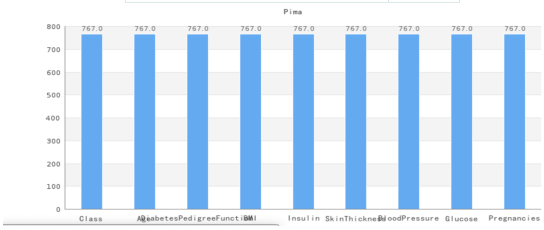
Header: Python 2.7; import csv; import random; import math; import operator; import time

Pima:

(20 points) Dataset details: Describe the data and some simple visualizations (for images, a few examples from each category; for other data, perhaps some scatter plots or histograms that show a big picture of the data). Describe your training/test split for K-NN and justify your choices.

I used a ratio of 3:1 and 4:1 to divide the data diabetes.csv into training data and test data. For project of diabetes. Train set: 469 and Test set: 298 (3:1), Train set: 526 and Test set: 241 (4:1), I used two different rates to prove my choices. This date included 9 category, Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age and Outcome. Meanwhile, we need to use the last column of data to validate our result, then I normalize that to 1 or 0. From this graph, we can check the valid number of each type is 767. So we can say this is a complete data set.

Valid Class	767.00
Valid Age	767.00
Valid DiabetesPedigreeFunction	767.00
Valid BMI	767.00
Valid Insulin	767.00
Valid SkinThickness	767.00
Valid BloodPressure	767.00
Valid Glucose	767.00
Valid Pregnancies	767.00



(10 points) Algorithm Description: K-NN is a very clear algorithm, so here describe any data pre-processing, feature scaling, distance metrics, or otherwise that you did.

To Implement k-Nearest Neighbors, we need to do those following step.

First of all, read the CSV format data.Second, we need to calculate the distance of each row of data, which will use euclidean Third, look for adjacent data. This is a straight forward process of calculating the distance for all instances and selecting a subset with the smallest distance values. Meanwhile, we still need to sort them to help next step, such as following picture.

```python
def euclideanDistance(ed1, ed2, length):
    distance = 0
    for x in range(length):
        distance += pow((ed1[x] - ed2[x]), 2)
    return math.sqrt(distance)


def getNeighbors(trainingSet, tested, k):
    distances = []
    length = len(tested)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(tested, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors
```

Fourth, the next task is to get the prediction results based on several recent examples. We can use these neighboring elements to predict the outcome of the vote and we choose the highest number of votes. Finally, evaluate the accuracy of the algorithm.

(40 points) Algorithm Results: Show the accuracy of your algorithm — in the case of the Pima Dataset, show accuracy with tables showing false positive, false negative, true positive and true negatives. For the Pima Dataset, use three different distance metrics and compare the results.

For the different distance metrics, I chose Manhattan Distance, Euclidean Distance and Cosine Distance

For the Manhattan Distance, it takes a longer time, and the accuracy rate is

```
--> Accuracy: 66.40625%
--> Train set: 511
--> Test set: 256
--> TP:0
--> FP:86
--> TN:0
--> FN:170
--> Run time: 1.4131278991699219
```

about 66.4 distance.png

For the Cosine Distance, it takes a short time, but the accuracy rate only has about 44.8

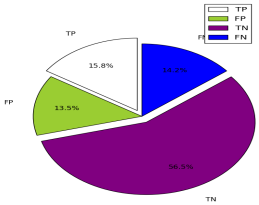
```
--> Accuracy: 44.81327800829876%
--> Train set: 526
--> Test set: 241
--> TP:34
--> FP:50
--> TN:83
--> FN:74
--> Run time: 0.3771829605102539
```

Therefore, after comparing these three methods, I think the Euclidean distance method can provide higher accuracy

In Pima Dataset and base on above computation, I chose k=3, k=5, k=7 and Euclidean distance method to compare the data.

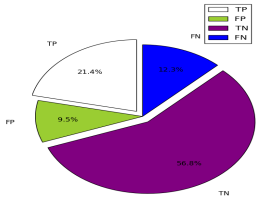
For k=3,

```
--------------------------------------------------
--> Accuracy: 72.30769230769231%
--------------------------------------------------
--> Train set: 507
--> Test set: 260
--------------------------------------------------
--> TP:41
--> FP:35
--> TN:147
--> FN:37
--------------------------------------------------
--> actualNegative:182
--> actualPositive:78
--> predictedNegative:184
--> predictedPositive:76
--------------------------------------------------
--> Run time: 0.4053499698638916
```



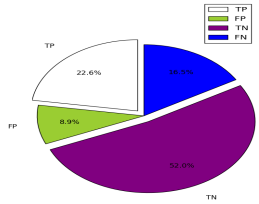
For k=5,

```
--------------------------------------------------
--> Accuracy: 78.18930041152264%
--------------------------------------------------
--> Train set: 524
--> Test set: 243
--------------------------------------------------
--> TP:52
--> FP:23
--> TN:138
--> FN:30
--------------------------------------------------
--> actualNegative:161
--> actualPositive:82
--> predictedNegative:168
--> predictedPositive:75
--------------------------------------------------
--> Run time: 0.37317991256713867
```



For k=7,

```
--------------------------------------------------
--> Accuracy: 74.59677419354838%
--------------------------------------------------
--> Train set: 519
--> Test set: 248
--------------------------------------------------
--> TP:56
--> FP:22
--> TN:129
--> FN:41
--------------------------------------------------
--> actualNegative:151
--> actualPositive:97
--> predictedNegative:170
--> predictedPositive:78
--------------------------------------------------
--> Run time: 0.37181997299194336
```



From the above data, we can find that the accuracy is highest when k=5

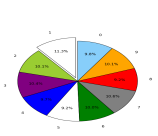
(10 points) Runtime: Describe the run-time of your algorithm and also share the actual "wall-clock" time that it took to compute your results.

We can see the running time in the last problem and with the k's increases, the running time is reduced.

MNIST

(20 points) Data set details: Describe the data and some simple visualizations (for images, a few examples from each category; for other data, perhaps some scatter plots or histograms that show a big picture of the data). Describe your training/test split for K-NN and justify your choices.

This data set stores a lot of data, including 42,000 handwritten digits in train.csv, represented by 738 pixels (each row). After simple statistics, we can find the distribution of Numbers, and got the largest number is '1'. However, in order to reduce running time and increase test efficiency of this huge data set, I will choose 0.9 train data and 0.1 test data. For the graph 2, I used Matlab to draw few examples to explain how is work.



(10 points) Algorithm Description: K-NN is a very clear algorithm, so here describe any data pre-processing, feature scaling, distance metrics, or otherwise that you did.

First of all, the method what I used is very similar to the previous data set, I still ignore the first row of the file, because those are headers. But, the label of this data set is in the first one, so we need to convert the location of the comparison to index 0. In addition, in the following part of distance metrics, I still choose Euclidean distance, because it can guarantee a good accuracy rate and relatively high efficiency.

(40 points) Algorithm Results: Show the accuracy of your algorithm. In the case of the MNIST digits show the complete confusion matrix. Choose a single digit to measure accuracy and show how that number varies as a function of K.

Based on the statistics of the first question, we can find that the number 1 is the most frequent, therefore, I will use the number 1 to test my program. I chose k=3, 5, and 7, because odd numbers can avoid the same votes at the same distance. In confusion matrix, I collected TP, FP, TN, FN, actualNegative, actualPositive, predictedNegative and predictedPositive to show a complete table.

For k=3, the total accuracy of test data is around 0.96, that's a satisfactory result. For the number '1', accuracy is 0.196, this is because TN is a large part of the result.

```
--> Total Accuracy: 96.07843137254902%
--> Train set: 41948
--> Test set: 51
--> TP:10
--> FP:0
--> TN:41
--> FN:0
--> actualNegative:41
--> actualPositive:10
--> predictedNegative:41
--> predictedPositive:10
--> accuracy of #119.07843137254903%
--> Run time: 2875.708692073822
```

For k=5,the total accuracy of test data is around 0.936. For the number '1' what I chose, accuracy is 0.253.

```
--> Total Accuracy: 93.61702127659575%
--> Train set: 41952
--> Test set: 47
--> TP:12
--> FP:1
--> TN:34
--> FN:0
--> actualNegative:35
--> actualPositive:12
--> predictedNegative:34
--> predictedPositive:13
--> accuracy of #125.53191489361702%
--> Run time: 728.9866369533539
```

For k=7, the total accuracy of test data is around 100. For the number '1' what I chose, accuracy is 0.7, the reason for this is that '1' appears only four times, but all four of these predictions are correct.

```
--> Total Accuracy: 100.0%
--> Train set: 41947
--> Test set: 52
--> TP:4
--> FP:0
--> TN:48
--> FN:0
--> actualNegative:48
--> actualPositive:4
--> predictedNegative:48
--> predictedPositive:4
--> accuracy of #117.69230769230769%
--> Run time: 803.608912229538
```

From the above data, we can find that the accuracy is highest when k=7

(10 points) Runtime: Describe the run-time of your algorithm and also share the actual "wall-clock" time that it took to compute your results.

We can see the running time in the last problem and it shown in result. After comparison, although k=7 takes more time, but the accuracy will be improved. On the contrary, when k=3, the time is the most and the accuracy is the least.

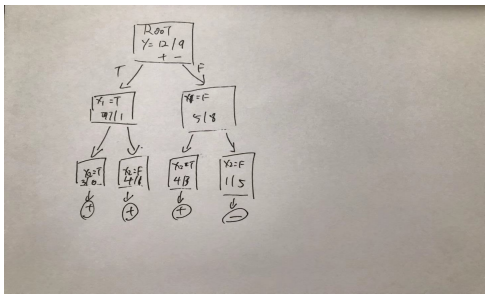
Decision Trees: Consider the following set of training examples for the unknown target function $< X_1, X_2 > \rightarrow Y$.

Y	X_1	X_2	Count
+	T	T	3
+	T	F	4
+	F	T	4
+	F	F	1
-	T	T	0
-	T	F	1
-	F	T	3
-	F	F	5

1. **(10 points)** What is the sample entropy $H(Y)$ for this training data?
   H(Y) = -$\sum p(Y) \log p(Y) = -\frac{3}{7}lg\frac{3}{7} - \frac{4}{7}lg\frac{4}{7} \approx 0.985228136040012$

2. **(10 points)** What are the information gains $IG(X_1) \equiv H(Y) - H(Y|X_1)$ and $IG(X_2) \equiv H(Y) - H(Y|X_2)$ for this sample of training data?
   H(Y—x1) = -$\sum p(X1 = xj) \sum\ p(Y = y1|X1 = xj) lgp(Y = yj|X1 = xj) = -\frac{8}{21}(\frac{7}{8}lg\frac{7}{8} + \frac{1}{8}lg\frac{1}{8}) - \frac{13}{21}(\frac{5}{13}lg\frac{5}{13} + \frac{8}{13}lg\frac{8}{13}) \approx 0.80212$
   $same as function above H(Y|X2) = -\frac{10}{21}(\frac{7}{10}lg\frac{7}{10} + \frac{3}{10}lg\frac{3}{10}) - \frac{11}{21}(\frac{5}{11}lg\frac{5}{11} + \frac{6}{11}lg\frac{6}{11}) \approx 0.9403448$
   $Information\ Gain\ I(X1) = H(Y) - H(Y|X1) = 0.98523 - 0.802123 = 0.183107$
   $Information\ Gain\ I(X2) = H(Y) - H(Y|X2) = 0.98523 - 0.9403448 = 0.044885$
   $So, I(X2) < I(X1)$

3. **(5 points)** Draw the decision tree that would be learned by ID3 (without postpruning) from this sample of training data.



Reference https://blog.csdn.net/wowotuo/article/details/38262057
https://numerics.mathdotnet.com/Distance.html
https://python.freelycode.com/