James Levy

Images Dataset

1) Dataset details
    a) This dataset contains and draws images digits 0-9. They are represented as a vector of pixels where each pixel is represented by an integer to represent the lightness/darkness of the grayscale pixel. Each image has 784 pixels. Here is an example of how an image vector would look as a in a way that could produce the image:

```
000 001 002 003 ... 026 027
028 029 030 031 ... 054 055
056 057 058 059 ... 082 083
 |   |   |   |  ...  |   |
728 729 730 731 ... 754 755
756 757 758 759 ... 782 783
```

    b) For the project, I used the train.csv file. This file also contains a label with the "correct" answer for every row. This allows me to check the accuracy of my KNN algorithm after running it. This is an example of what the start of some data may look like. Note that most vector entries are 0 to denote white space:

pixel148	pixel149	pixel150	pixel151	pixel152	pixel153	pixel154	pixel155	pixel156	pixel157	pixel158	pixel159	pixel160
0	0	0	0	0	0	0	0	0	0	0	191	250
13	86	250	254	254	254	254	217	246	151	32	0	0
0	0	0	0	9	254	254	8	0	0	0	0	0
6	0	0	0	0	0	0	0	0	9	77	0	0
8	103	253	253	253	253	253	253	253	253	114	2	0
0	0	0	5	165	254	179	163	249	244	72	0	0

    c) For the knn split between test and training data, I made the assumption that the images listed as rows in the dataset were already in a random order. Due to the large amount of information, the algorithm was running extremely slowly. For this reason, I chose to use 50 elements (making sure there were at least 1 of every represented digit) as the training data. I chose another distinct 50 elements to do the testing on. These are the values that we would predict using the training data. Although a smaller sample, I believe it is enough to get the idea of the algorithm across.
    d) I chose to start with k equal to 3. This is because given the number of possible classifications, I did not want a large k to start to interfere with a possible correct majority. Likewise, a k = 1, offers too much room for error in case the closest neighbor happens to be incorrect or an outlier.

2) Algorithm description
    a) Load data
        i) Loading the data involved a simple loop that converted the first 100 rows in the .csv file into two matrices. Because of the long run time described in the run time section. The first 50 rows are saved as a training data matrix, and the second 50 rows are saved as a matrix for testing purposes.
    b) Calculate distances
        i) There is 1 distance formula used in the algorithm. The euclidean distance formula used is the traditional formula of sqrt((x1-x2)^2+(y1-y2)^2 + …). This sum and calculation is done by looping over the vector elements in the two rows being compared. The distance calculations ignore the label value.
    c) Get nearest neighbors

   i) Use my get_neighbors() function
   ii) Finding the nearest neighbors involves us having a given row. We go through a loop and find the distance from our row to every other row in the dataset. Once we have all rows matched to a distance, we sort the rows in ascending order. From there, we return a list of rows with the smallest k distances.

d) Make a prediction
   i) Use my predict_classification() function
   ii) In order to make a prediction we call the get neighbors function. Using those results we select the label value that occurs most often from our neighbors rows. This is done via a loop.

e) Run the K-NN program
   i) This function is a simple loop that calls the prediction() function on every row in the testset, while using the dataset to make the predictions. It adds the results of that function into a list of predictions.
   ii) We then have a function that builds a confusion matrix. It loops through all predicted values from the test set. For each prediction, it looks at the label for the test set. It then adds +1 at matrix[true value][prediction value]
   iii) We then have a function that analyses accuracy. It contains a loop that compares the predicted value to the label value for a row in the testset.

3) Algorithm results

Start KNN with k = 3

i |[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
0|[3, 1, 0, 0, 0, 0, 0, 0, 0, 0]
1|[0, 7, 0, 0, 0, 0, 0, 0, 1, 0]
2|[1, 2, 0, 1, 0, 0, 0, 1, 0, 1]
3|[0, 2, 0, 1, 0, 0, 0, 0, 1, 0]
4|[0, 0, 0, 0, 3, 0, 0, 1, 0, 1]
5|[0, 1, 1, 0, 0, 1, 0, 1, 0, 0]
6|[0, 2, 1, 0, 1, 0, 2, 0, 0, 0]
7|[0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
8|[0, 1, 1, 0, 0, 0, 0, 0, 1, 0]
9|[0, 1, 0, 0, 0, 0, 0, 0, 0, 7]

accuracy: 0.9
True Positive: 7
True Negative: 20
False Positive: 2
False Negative: 1

time to complete K-NN algorithms/predictions(s): 2.5850877000000003

how accuracy varies with k = 1
accuracy: 0.47368421052631576
True Positive: 1
True Negative: 8
False Positive: 5

False Negative: 5

how accuracy varies with k = 10
accuracy: 0.5
True Positive: 2
True Negative: 7
False Positive: 5
False Negative: 4

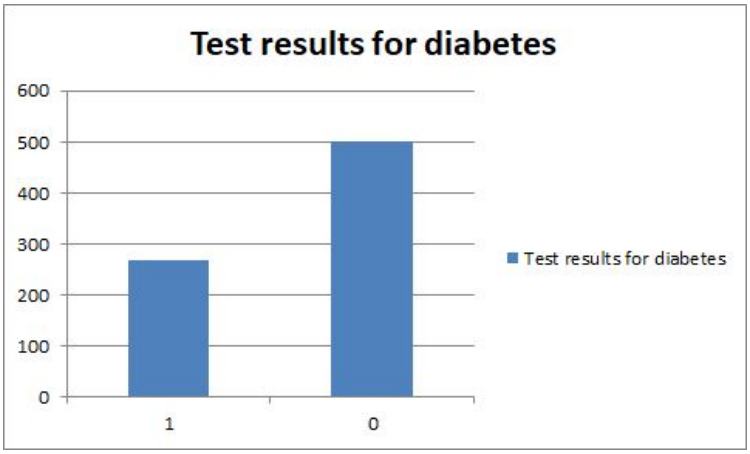
Accuracy with K seems to be parabolic with a maximum. We can see that at k=1 the accuracy is low because we may get an outlier on who is closest. It then rises as k = 3 and it can make better predictions. Once k=10 however, other incorrect labels may start overtaking and distracting from the correct solution.

4) Runtime
    a) It takes O(m) time to calculate the distance between two rows of data, where m is the number of features in a dataset vector.
    b) The algorithm itself works by finding the k closest neighbors for each row. This takes O(n) time where the n is the number of rows in a dataset.
    c) Finding the closest neighbors involves us going through all the rows in the data in order to calculate the distance formula. For this part of the algorithm, where n is the number of rows in the dataset. This section also involves a sort, bumping the min run-time to at least O(nlogn)
    d) Putting all of this together, along with doing the calculations on every row of the dataset, gives us a run-time of approximately O(mn^(2)log(n)) not counting how long python's max() function takes (this is used in the prediction step)
    e) Due to 42,000 entries, calculating the closest neighbors takes too long when using the entirety of the dataset given in the csv. This is why I cut the data set and test set in size in order to work on my machine. All data would require a GPU and an algorithm change to allow concurrency.
    f) The time it took to finish my scenario is 2.585 seconds

James Levy

PIMA Dataset

1) Dataset details
    a) The dataset given in this example deals with the numbers of people to test + for diabetes and what sets of features they have. The dataset contains 768 entries overall, each one representing a different person. The features that are checked in order to determine diabetes positivity are Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, and Age.



    b) From a quick look at the dataset, we can see that the majority of people who are represented do not test positive for diabetes. In fact only 268 do. This is known due to the label element of the data which states the correct result.

    c) The dataset also comes with a set of overall graphs for each variable. This allows us to see the big pictures of what the data looks like overall. As we can see, features such as pregnancies, insulin, the diabetes pedigree function and age are fairly heavily right

skewed. Other features seem to have a more normal distribution; glucose, blood pressure, and BMI.

d) For the testing/knn split, I decided to use the entirety of the dataset to train. Because of this, I also ran KNN on every vector in the dataset so that I could get a complete view of how the vectors were near each other. When running Knn, the label for the result (diabetes +/-) was not included in the distance calculation. This was done to make sure that it did not skew the effectiveness of feature closeness. I chose a K of 3 as it seemed like a fairly average number which was near the ½ point of how many features are in a vector.

2) Algorithm Description
   a) Load data
      i) Loading the data involved a simple loop that converted every row in the .csv file into a matrix. Each value in the matrix is converted to a float instead of string.
   b) Calculate distances
      i) There are 3 different distance formulas used in the algorithm. The euclidean distance formula used is the traditional formula of sqrt((x1-x2)^2+(y1-y2)^2 + …). The second function is the Manhattan distance. This function looks like sum(abs(x1-x2)+abs(y1-y2) + …). Lastly we use the Minkowski distance with a power of 3. This looks similar to the euclidean distance except we take the cubed root and and we cube all the differences.
      ii) All distance calculations ignore the label value.
   c) Get nearest neighbors
      i) Use my get_neighbors() function
      ii) Finding the nearest neighbors involves us having a given row. We go through a loop and find the distance from our row to every other row in the dataset. Once we have all rows matched to a distance, we sort the rows in ascending order. From there, we return a list of rows with the smallest k (3) distances.
   d) Make a prediction
      i) Use my predict_classification() function
      ii) In order to make a prediction we call the get neighbors function. Using those results we select the label value that occurs most often from our neighbors rows. This is done via a loop.
   e) Run the K-NN program
      i) This function is a simple loop that calls the prediction() function on every row in the dataset. It does this for all 3 distance types as well. It adds the results of that function into lists for each distance type.
      ii) We then have a function that analyses accuracy. It contains a loop that compares the predicted value to the label value for a row in the dataset.

3) Algorithm Results

Results for Euclidean distance
accuracy: 0.859375
True Positive: 201
True Negative: 459
False Positive: 41
False Negative: 67

Results for Manhattan Distance
accuracy: 0.8541666666666666
True Positive: 190
True Negative: 466
False Positive: 34
False Negative: 78

Results for Minkowski Distance
accuracy: 0.8450520833333334
True Positive: 195
True Negative: 454
False Positive: 46
False Negative: 73

time to complete all K-NN algorithms/predictions(s): 4.9712837

As we can see, in this situation, the 3 different distance metrics did fairly equally when it came to accuracy. However, all of their tables look slightly different. For example, manhattan was the worst providing true positives but did better when it came to not giving false positives.

4) Runtime
    a) It takes O(m) time to calculate the distance between two rows of data, where m is the number of features in a dataset vector.
    b) The algorithm itself works by finding the 3 closest neighbors for each row. However, it does this using 3 different distance measures. This takes O(n) time where the n is the number of rows in a dataset.
    c) Finding the closest neighbors involves us going through all the rows in the data in order to calculate the distance formula. For this part of the algorithm, where n is the number of rows in the dataset. This section also involves a sort, bumping the min run-time to at least O(nlogn)
    d) Putting all of this together, along with doing the calculations on every row of the dataset, gives us a run-time of approximately O(3mn^(2)log(n)) not counting how long python's max() function takes (this is used in the prediction step)
    e) When not running the algorithm with any concurrency, it took my computer a total of 4.971 seconds to compute all of the prediction lists for the 3 distance measures.