CIS 572 Assignment 3

Steven Walton

May 5, 2019

Problem 1

An analyst wants to classify a number of customers based on some given attributes: total number of accounts, credit utilization (amount of used credit divided by total credit amount), percentage of on-time payments, age of credit history, and inquiries (number of time that the customer requested a new credit account, whether accepted or not). The analyst acquired some labeled information as shown in the following table:

ID	Total Accounts	Utilization	Payment History	Age of History (days)	Inquiries	Label
1	8	15%	100%	1000	5	GOOD
2	15	19%	90%	2500	8	BAD
3	10	35%	100%	500	8	BAD
4	11	40%	95%	2000	6	BAD
5	12	10%	99%	3000	6	GOOD
6	18	15%	100%	2000	5	GOOD
7	3	21%	100%	1500	7	BAD
8	14	4%	100%	3500	5	GOOD
9	13	5%	100%	3000	3	GOOD
10	6	25%	94%	2800	9	BAD

Consider the following three accounts to be labeled:

Total Accounts	Utilization	Payment History	Age of History (days)	Inquiries	Label
20	50%	90%	4500	12	P1
8	10%	100%	550	4	P2
9	13%	99%	3000	6	P3

(a) Before using nearest neighbor methods to make predictions, how would you recommend processing or transforming the data? Why? Make any changes you think appropriate to the data before continuing on to the next two parts.

(b) What are the predicted labels P1, P2, and P3 using 1-NN with L 1 distance? Assume that percentages are represented as their corresponding decimal numbers, so 95% = 0.95. Show your work.

(c) Keep the information of customers 7, 8, 9, and 10 as validation data, and find the best K value for the K-NN algorithm. If the best value of K is not equal to 1, find the new predictions for P1, P2, and P3. Show your work.

Part a)

Before doing nearest neighbors I would standardize Total Accounts, Age of History, and Inquiries. Utilization and Payment history do not need to be standardized because they are already statistical values. I would also convert the percentages to their decimal form and relabel "GOOD" = 1 and "BAD" = 0
This gives us the following table

ID	Total Accounts	Utilization	Payment History	Age of History (days)	Inquiries	Label
1	1.79887605	0.15	1.	1.04031297	2.35993791	1
2	3.3728926	0.19	0.9	2.60078243	3.77590065	0
3	2.24859507	0.35	1.	0.52015649	4.71987582	0
4	2.47345457	0.4	0.95	2.08062595	2.83192549	0
5	2.69831408	0.1	0.99	3.12093892	2.83192549	1
6	4.04747112	0.15	1.	2.08062595	2.35993791	1
7	0.67457852	0.21	1.	1.56046946	3.30391307	0
8	3.14803309	0.04	1.	3.64109541	2.35993791	1
9	2.92317359	0.05	1.	3.12093892	1.41596274	1
10	1.34915704	0.25	0.94	2.91287633	4.24788823	0

This puts everything in a way that a computer can understand and we are manipulating numbers in a consistent way that will not bake in a bias into the data.
An example of baking in bias would be to normalize any column, because we would then add information, where we know what the largest number is. This would be a problem because Age of History for P1 would be > 1. So let's not do that.
Essentially we want all preprocessing transformations to be affine.

Part b)

The easiest thing to do is put everything into arrays and calculate the distance in

each metric. We can do this simply with python my doing the following:

```
for i in range(len(historical_data)):
    print(np.linalg.norm(historical_data[i][:-1] - p1,1))
173.90056753592185
>>>169.21011878426268
171.41106710090637
171.41368846095935
170.48851598104
170.6116594957137
173.50073341951403
170.06062806267784
171.7396192192782
170.42977287093967

for i in range(len(historical_data)):
    print(np.linalg.norm(historical_data[i][:-1]-p2,1))
66.51302641069294
68.05544206856952
68.2071201162355
67.65074844546294
68.12620191198478
>>>65.30474431692596
69.16145559408837
67.7746518035519
67.46935482364034
70.8832591024702


for i in range(len(historical_data)):
    print(np.linalg.norm(historical_data[i][:-1]-p3,1))
76.52073327477089
73.97823484969435
78.20713349292065
74.61385419980834
>>> 73.06868171988899
73.23182523456269
77.18487432165438
73.68110677474745
74.31978495812722
76.18186409966265
```

We use numpy's linear algebra package to call upon norm. Norm defaults to the Euclidean norm, but by passing 1 to the second parameter we can get the Manhattan distance.
We can easily see that the closest label (marked with >>>) is that of ID=2, which

is labeled BAD, therefore we should label P1 as BAD.
Doing this again we see that P2 is closest to ID=1, so GOOD.
P3 is closest to ID=5, so GOOD
Giving us

Label	Total Accounts	Utilization	Payment History	Age of History (days)	Inquiries	Label
p1	168.304435	0.5	0.9	4.6814	5.6638	BAD
p2	67.3217740	0.1	1	0.5721	1.8879	GOOD
p3	75.7369958	0.13	0.99	3.1209	2.8319	GOOD

Part c)

The easy way to do this is create a table of the nearest neighbors to the training set from the validation set. That is

ID	1	2	3	4	5	6
7	2.64842918	4.33061464	4.17029227	3.03102012	4.1761926	4.89702425
8	4.05993947	2.93113522	6.69031485	3.11703556	1.51186308	2.56990749
9	4.24889864	3.56981341	6.87927402	3.30599473	1.70082225	3.20858567
10	4.37023269	2.90781703	3.92414545	3.53251066	3.17318238	5.57851479

Figure 1: Distance between validation set and testing data points

With indexing of sorted being

1	4	3	5	2	6
5	6	2	4	1	3
5	6	4	2	1	3
2	5	4	3	1	6

Figure 2: ID of sorted distance

This would suggest that $k = 3$ is the best choice.

k	1	2	3	4	5	6
7	GOOD	GOOD	BAD	GOOD	BAD	GOOD
8	GOOD	GOOD	GOOD	GOOD	GOOD	GOOD
9	GOOD	GOOD	GOOD	GOOD	GOOD	GOOD
10	BAD	GOOD	BAD	BAD	BAD	GOOD

Figure 3: Classification using kth nearest neighbor, break ties towards GOOD

Referring back to part b we can look at the classifications of the p's and we get

p1	BAD
p2	GOOD
p3	GOOD

Problem 2

This exercise demonstrates how the performance of nearest neighbor degrades as you add more dimensions. Please do the following in a programming language of your choice:

1. Generate a training dataset with 100 points in 10 dimensions. For 50 of the points, the first dimension (x 1) should be 0; for the other 50, the first dimension should be 1. All other dimensions (x 2 through x 10) should be either 0 or 1, selected randomly with equal probability.

2. Follow the same procedure from step 1 to generate a random test dataset.

3. Count the fraction of test points that would be correctly labeled using 1-nearest neighbor with this training data. (One way to do this is by checking if the closest training point to each test point has the same value for the first dimension, x 1 .) This is the accuracy of nearest neighbor on this synthetic data.

4. Repeat steps 1-3 10 times, and average the results. This is the average accuracy of nearest neighbor in 10 binary dimensions, with 1 perfectly informative attribute and 9 random ones.

5. Repeat step 4 with 5, 10, 20, 50, and 100 dimensions. As before, every dimension should be random except for the first (x 1). Report the average accuracy for each number of dimensions.

If youre familiar with numpy and Python list comprehensions, its possible to do this exercise with just 30 lines of well-structured code. If you do everything from scratch or use a more verbose language, its a few more lines of code but it still shouldnt be too difficult. For your answer to this question: please include:

(a) the average accuracies obtained in step 5,

(b) a graph showing the accuracy versus the number of dimensions, and

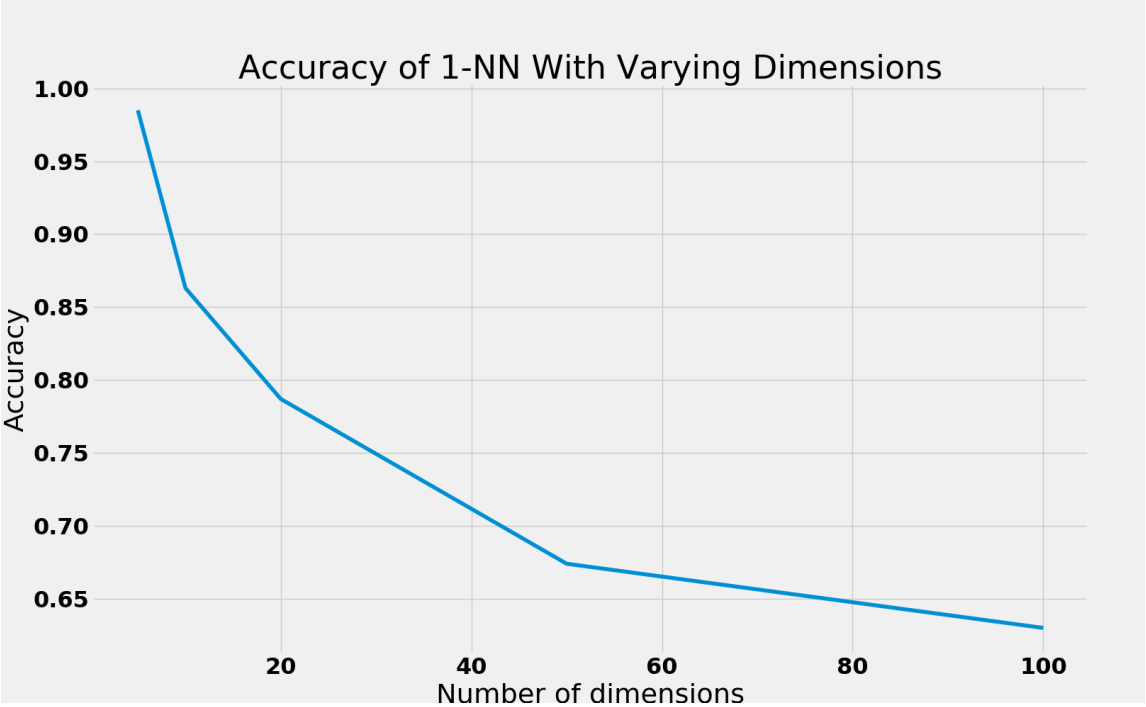
(c) a printout of the code you used to help answer this question.

(Your code must be your own work.)

Solution)

```
import numpy as np
import statistics as stat
import matplotlib.pyplot as plt
import matplotlib
# Plotting style
plt.style.use('fivethirtyeight')
font = {'family': 'normal',
        'weight': 'bold',
        'size': 23}
matplotlib.rc('font',**font)


# Number of points
N = 100
# Dimensions that we want to test
dim_set = [5,10,20,50,100]

def gen_data(dims):
    r"""
    Generates the training data where the first 50 points
    have 0 in the first dimension and 1 for the next 50
    points
    """
    training_data = np.random.randint(2,size=(N,dims))
    training_data[:50,0] = np.ones(len(training_data[:50]))
    training_data[51:,0] = np.zeros(len(training_data[51:]))
    return training_data

def get_nn(train,validation,lp=1):
    r"""
    Get the nearest neighbors
    """
    dist = [np.linalg.norm(validation[0]- train[i],lp) for i in
        range(len(train) )]
    sorted_dist = np.argsort(dist)
    for i in range(1,len(validation)):
        d = [np.linalg.norm(validation[i]-train[j],lp) for j in
            range(len(train)) ]
        dist = np.vstack((dist,d))
```

```
        sorted_dist = np.vstack((sorted_dist,np.argsort(d)))
    return dist[:,0], sorted_dist[:,0]

def average_nn(dims,times=10):
    r"""
    Average the results of the nearest neighbor classification
    """
    acc = np.zeros(N)
    for t in range(times):
        train = gen_data(dims)
        test  = gen_data(dims)
        _, s_dist = get_nn(train,test)
        for i in range(N):
            if test[i][0] == train[s_dist[i]][0]:
                acc[i] += 1
    acc = [acc[i]/times for i in range(N)]
    acc = np.average(acc)
    return acc
```



Problem 3

Suppose we have some binary input data, $x_i \in \{0, 1\}$. The training data is shown in the above table. Note that we have two outputs per training example.
Let us first embed each x_i into 2d using the following basis function:

$$\phi(0) = (1, 0)^T, \phi(1) = (0, 1)^T$$

x	y
0	(-1,-1)^T
0	(-1,-2)^T
0	(-2,-1)^T
1	(1,1)^T
1	(1,2)^T
1	(2,1)^T

Consider a linear regression model over ϕ:

$$\hat{y} = W^T \phi(x)$$

where W is a 2x2 matrix. Compute W that has the maximum likelihood on the training data (called the maximum likelihood estimation). Show your calculation details.

Solution)

We can simplify this problem and take some nice mathematical shortcuts if we rewrite the matrix form

$$\begin{pmatrix} -1 & -1 \\ -1 & -2 \\ -2 & -1 \\ 1 & 1 \\ 1 & 2 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

We notice that we have solutions

$$w_{11} = -1$$
$$w_{11} = -1$$
$$w_{11} = -2$$

If we're versed in linear regularization we know that we get the solution that is the same as the averaged solution (as this case it will minimize the residual error). In this case $w_{11} = -\frac{4}{3}$
If we exploit the nice pattern here we can write W as

$$W = \begin{pmatrix} -\frac{4}{3} & -\frac{4}{3} \\ \frac{4}{3} & \frac{4}{3} \end{pmatrix}$$

Problem 4

The multiclass perceptron maintains a weight vector and bias for each class. When it makes an incorrect prediction, it adjusts the weight vector of both the predicted class $\hat{y}$ and the correct class y

Prove that the standard perceptron is equivalent to the multiclass perceptron when there are only 2 classes.

Proof)

We note that the activation function for the standard perceptron is

$$a = \langle w, x \rangle + b$$

For the multiclass perceptron it is

$$a_w = \langle w_y, f(x) \rangle$$

Where

$$f(x) \begin{cases} 1, & \langle w, x \rangle + b > 0 \\ 0, & \text{else} \end{cases}$$

and we say that

$$y = \underset{x}{\text{argmax}}(a_w(x, y))$$

If there are two classes then this also means that there is only one binary classification. When we do $f(x) = \underset{i}{\text{argmax}}\, f_i(x)$ we only have one i and therefore we always yield the argmax result. So can show that the multiclass perceptron trivially reduces to the preceptron:

$$\begin{aligned} y &= \underset{x}{\text{argmax}}(a_w(x, y)) \\ &= a_w(a, y) \\ &= \langle w_y, f(x) \rangle \end{aligned}$$

this is the same as the activation of the standard perceptron.

Problem 5

Given a set of data points, we can define the convex hull to be the set of all points x given by

$$x = \sum \alpha_i x_i$$

where $\alpha_i \geq 0$ and $\sum \alpha_i = 1$. Consider a second set of points together with their corresponding convex hull. By definition, the two sets of points will be linearly seperable if there exists a vector, w, and a scalar, b, such that $w^T x_i + b > 0 \; \forall x_i$ and $w^T z_i + b < 0 \; \forall z_i$

Proof)

We'll let C be convex hulls and H be half spaces.
Suppose that the two convex hulls are linearly seperable. That means that $\exists\, C_1 \in H$ and $C_2 \in H^C$. By the definition of half spaces we have $H \cap H^T = \oslash \therefore C_1 \cap C_2 = \oslash$

Suppose there is a point that belongs to two convex hulls, that is $\exists p \in C_1 \cap C_2$. If we suppose that C_1 and C_2 are linearly sepearble, that is

$$w^T x_i + b > 0$$
$$w^T z_i + b < 0$$

This implies that when $x_i = p = z_i$, the result is both > 0 and < 0. Since this cannot exist, we can say that no such point p can exist, and $\therefore C_1 \cap C_2 = \oslash$