CS519 HW #1

Matthew Frantz

April 10, 2019

1 Data (Pre-)Processing (Feature Map)

1.1

The positive % of the training data is **25.02%**. For the dev set, the positive % is **23.6%**.

According the video this data is from 1994 to these numbers make sense for the time given the average per capita income level was roughly $27K per year. This percentage has gone up since then and according to the IRS Statistics of Income, as of 2014 37.6% of Americans make >$50K per year [1].

1.2

1. Youngest: 17
2. Oldest: 90
3. Least amount of hours: 1 hour
4. Most amount of hours: 99 hours

1.3

We need to binarize all the categorical fields because we need a means of conducting numerical analysis (i.e. distance calculations) on the labeled data. Although integer encoding could be used, some of the data doesn't have an ordered relationship like country of origin. Thus, using a binary approach we can overcome this lack or ordered relationship and still do numerical analysis.

1.4

There are 7 input fields not counting age and hours. If each of these fields is converted into a binary value then it can either equal 1 or 0. Equations (1) and (2) show the Euclidean and Manhattan distances respectively. Since we are binary, the only possible values for each difference pair is either 1 or 0.

$$L_2(x, y) = \sqrt{|x_1 - y_1|^2 + \cdots + |x_d - y_d|^2} \tag{1}$$

$$L_1(x, y) = |x_1 - y_1| + \cdots + |x_d - y_d| \tag{2}$$

The furthest distance would be when two person are exact opposite meaning for each input field, they do not have the same binary feature. An example is if the first person is: White, Male, and Separated, while the other is: Asian-Pac-Islander, Female, Married, they would be really far apart.

For a single input fields the furthest Euclidean Distance would be:

$$L_2(x, y) = \sqrt{|1 - 0|^2 + |0 - 1|^2} = \sqrt{2} \approx 1.414$$

This would repeat for all 7 inputs meaning the furthest Euclidean Distance would be:

$$FurthestEuclidean = 7 * \sqrt{2} \approx 9.9$$

For a single input fields the furthest Manhattan Distance would be:

$$L_2(x, y) = |1 - 0| + |0 - 1| = 2$$

This would repeat for all 7 inputs meaning the furthest Manhattan Distance would be:

$$FurthestManhattan = 7 * 2 = 14$$

1.5

We do not want to binarize the two numerical fields (age and hours) for several reasons. The first would be a large number of additional categories needed to capture the large number range which would add to processing time. Second, we would lose the ability to compare values that are *near* each other such as 20 and 21 year old which we expect to have similar characteristics. It would make it much more difficult to compare people close in age and working hours if they were just binary numbers. The distance between someone who was 60 and 17 for example would be the same as for a 60 and 59 year old if they were binary. However, we expect those two groups of people to be in very different parts of their life leading to a not so great comparison.

If we were to binarize all the fields such that each field has equal weight, we would need to normalize the feature vectors. This would ensure that the highest value would be bound to 1 for the largest distance.

1.6

From Table 1, we have a total of 92 features assuming age and Hours-Per-Week are not binary. The number of features allocated per field is shown in the second column of Table 1.

Table 1: Field and number of Feature for each field.

Field	Features
Age	1 (67 if Binary)
Sector	7
Education	16
Marital-Status	7
Occupation	14
Race	5
Gender	2
Hours-per-Week	1 (73 if Binary)
Country-of-Origin	39
Total	92 (231 if all Binary)

2 k-Nearest Neighbor Classification

2.1

From Table 1, we can see that we would have 231 features if we were to binarize every field.

- Q: Is there any work in training after finishing the feature map?
  - No, once the training data is properly vectorized the "training" is complete and no further action is required.
- Q: Whats the time complexity of k-NN to test one example (dimensionality d, size of training set $|D|$)?
  - Computing the feature vector will depend on the size of d. Calculating each distance function would be the size of the training data or $O(|D|)$. Finding the time complexity for k-NN on one example will depend mostly on the sorting algorithm used in additional to the previous two mentioned. Using standard sorting techniques will give you $O(|D|^2)$ time complexity. There are better techniques that can achieve $O(nlogn)$. So our time complexity would be the summation of all three elements.
- Q: Do you really need to sort the distances first and then choose the top k?
  - No, there are other algorithms out there like Quickselect that can operate in $O(n)$. This is what the Numpy Argpartition function does this.

2.2

Q: Why the k in k-NN has to be an odd number?

k-NN must be an odd number otherwise you could have cases where there is no clear winner. Since k-NN returns +1 if $\hat{y} > 0$ and -1 for $\hat{y} < 0$, having an even number for k could return 0, which is not useful. If k is always odd then we will always get a useful return.

2.3

Evaluations of k-NN on the Dev set are shown in Table 2 for $k = 1, 3, 5, 7, 9, 99, 999, 9999$.

Table 2: Dev set error rate and predicted positive % with Euclidean distance.

K	Error Rate	Positive %
1	26.1	26.9
3	25.3	24.7
5	22.7	24.3
7	23.5	23.7
9	22.5	23.1
99	23.1	11.3
999	23.6	0.0
9999	23.6	0.0

Q: whats your best error rate on dev, and where did you get it?

My best error rate on the Dev set was 22.5% for $k = 9$. This makes sense as we expect the test error to drop at first, find a minimum, then rise again as k gets larger.

2.4

Training and Dev set errors and positive % rate are shown in Table 3.

Table 3: Training set error rate and predicted positive %.

K	Training Error Rate (%)	Training Positive %	Dev Error	Dev Positive
1	0.0	25.02	26.1	26.9
3	2.14	24.08	25.3	24.7
5	3.12	23.66	22.7	24.3
7	3.72	22.98	23.5	23.7
9	4.48	22.38	22.5	23.1
99	14.18	16.92	23.1	11.3
999	24.5	0.52	23.6	0.0
9999	25.02	0.09	23.6	0.0

Q: When k = 1, is training error 0%? Why or why not? Look at the training data to confirm your answer

Looking at the training data in Table 3, when $k = 1$ the training error is 0%. This is because if the training data were to find which point it was closest to, it would be closest to itself resulting in 0% classification error.

2.5

- Q: What trends (train and dev error rates and positive ratios, and running speed) do you observe with increasing k? Do they relate to underfitting and overfitting?
  - For the training data the error rate increased as k increased which makes sense as you would expect the smallest error in the training data to be when it only compares itself to itself and is overfitted. For the training set the positive % rate went down as k increased which relates the to misclassification previously discussed. At k approached and exceeded the size of the training data, we see the error rate approach the positive % value. Since we know roughly 75% of the data set is $< 50K$, once k becomes large basically the majority label wins and the data is severely underfitted. In term of running speed the most expensive part of this algorithm is the sort. To help with this I only performed the sort once, then calculated my prediction labels for all required k values. Although the sort is expensive the rest of the calculations were quick.
  - For the Dev data, the error initially goes down to a minimum and then rises again. This follows the normal trend you expect our of testing data as it goes from over to underfitted. You see a similar pattern in dev error that you see in training error in that as k gets really large pretty much all values think they are $>= 50K$ leading to complete misclassification of all the $> 50K$ values. Time complexity was a shorter due to the number of distances and sorting required.
- Q: What does k = actually do? Is it extreme overfitting or underfitting? What about k = 1?
  - When $k = \text{inf}$ it means that each new point will take on the label of majority set meaning all labels will think they are the same. This will cause severe underfitting of the data. When $k = 1$ each data point will only look to the nearest neighbor and will form numerous tiny islands and complex edges leading to extreme overfitting of the data.

2.6

Table 4 shows the results for using the Manhattan distance as well as the Euclidean distance. It appears than Manhattan distance has a lower error rate of about for most k with similar positive % rates.

Although it appears Manhattan distance is slightly better I would not say that it is overwhelmingly better and either distance metric would be fine to use. An advantage to Manhattan might be less computation resources but it is unclear if taking the absolute value is more expensive than squares and square roots required for Euclidean.

Table 4: Manhattan versus Euclidean distance error rate and predicted positive %.

K	Manhattan Error Rate (%)	Manhattan Positive %	Euclidean Error	Euclidean Positive
1	25.9	27.05	26.1	26.9
3	22.6	26.2	25.3	24.7
5	22.2	24.6	22.7	24.3
7	21.8	25.6	23.5	23.7
9	22.7	24.3	22.5	23.1
99	19.9	17.5	23.1	11.3
999	21.8	2.4	23.6	0.0
9999	23.6	0.0	23.6	0.0

2.7

Table 5 shows the results of all the data being binarized compared to the age and hours worked not being binarized. What we see is much lower error rate from the all binary data for $k <$ size of the data set. Although this appears better I don't believe it makes sense. What this formulation does is allows the distance between someone who is 17 and 45 to be the same as between someone who is 17 and 18. Clearly the 17 and 18 would have more in common and more likely to be in the same income bracket where as the 45 year old would have a better chance at making more money (has had more time for education, career advancement, ect.). Therefore, I would say that this formulation is not useful in this case.

Table 5: All binarized data compared with age and hours worked not binarized error rate and predicted positive %.

K	All binary Error Rate (%)	All binary Positive %	Not all binary Error	Not all binary Positive
1	1.8	23.0	26.1	26.9
3	0.4	23.4	25.3	24.7
5	0.6	23.2	22.7	24.3
7	0.5	23.1	23.5	23.7
9	0.5	23.1	22.5	23.1
99	0.8	22.8	23.1	11.3
999	1.9	21.7	23.6	0.0
9999	23.6	0.0	23.6	0.0

3 Deployment

- Q: At which k and with which distance did you achieve the best dev results?
  - Figure 1 shows the comparison of Euclidean and Manhattan distance for the dev set from $k = 1$ to $k = 199$. From this we can see that Manhattan distance has a lower error rate for almost every k value and has the lowest error rate of 18.5% at $k - 61$.
- Q: Whats your best dev error rates and the corresponding positive ratios?
  - From Figure 1 we can see that the best dev error rate is when $k = 61$ and is 18.5%. This has a corresponding positive % of 17.9%.

- Q: Whats the positive ratio on test?
  - Using the above parameters on test, I found a positive ratio of 18.0%. This seems a little low compared to the other 2 sets (25.02% and 23.6%) but is within reason.

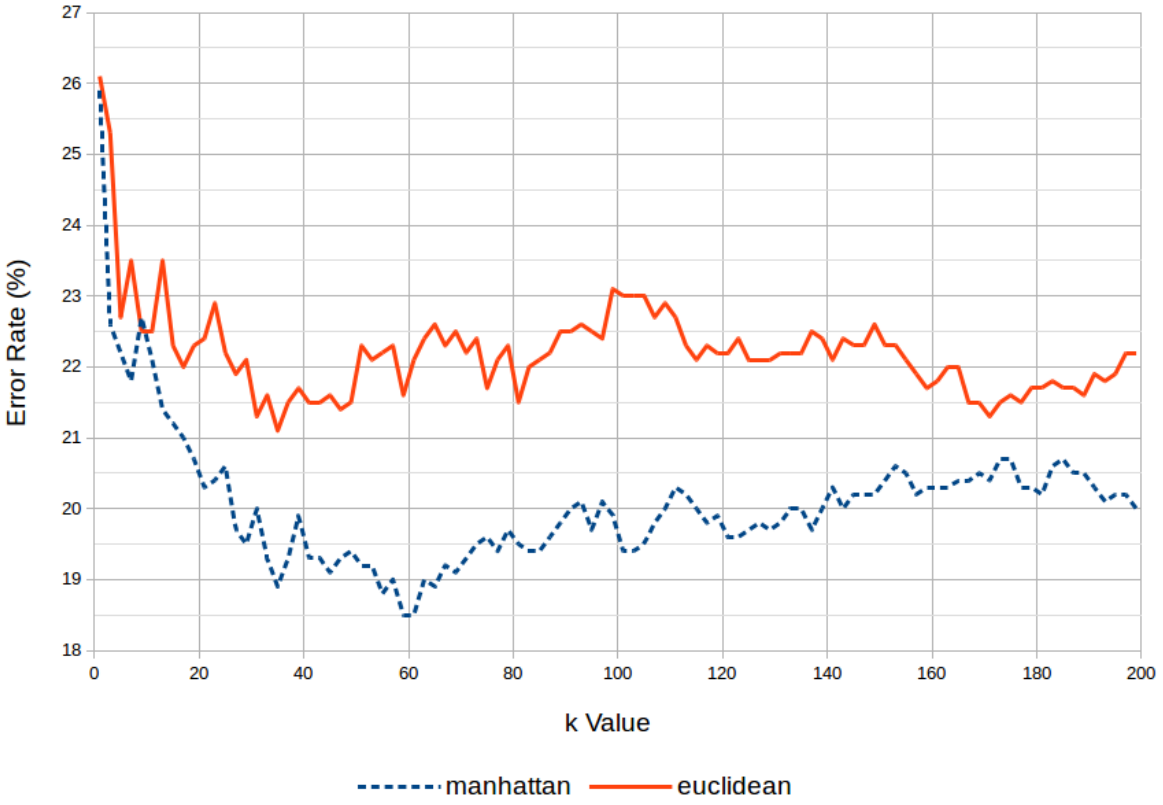

Figure 1: Comparison of k values and error rates for Euclidean and Manhattan distance from $k = 1$ to $k = 199$. The best error rate occurred at $k = 61$ using the Manhattan distance.

4 Observations

1. Q: Summarize the major drawbacks of k-NN that you observed by doing this HW.
   (a) The first one that stands out is having to sort all the data for each example. Since many sort algorithms have a worse case $O(n^2)$ performance, this could become really time intensive for much larger datasets.

   (b) I had to create some rules for dealing with new test and dev values that didn't come up during training. This seemed a bit informal to me, but k-NN doesn't really seem to have a nice way of handling new unknowns.

   (c) Having to tune the k parameter is an obvious drawback. For instance we knew the actual positive % rate of the dev set was 23% but for the lowest error rate I got a positive % of 17.5%. This makes it difficult as a designer because the intuition is to use a smaller error rate, but if you know what the actual output is and it doesn't match it makes you questions your method.

   (d) One of the biggest bottlenecks I found was calculating all the distances. I used np.sort() to conduct the sorting and it took on average about 2ms. Calculation of all the distances using the np.linalg.norm() method for Euclidean or np.abs(training - datapoint).sum() method for

Manhattan, was about 20ms or 10x more expensive. Since you have to run this over every data point it really adds up!

2. . Q: Do you observe in this HW that best-performing models tend to exaggerate the existing bias in the training data? Is it due to overfitting or underfitting? Is this a potentially social issue?

   (a) The data has a much large number of data points for white, from the United States, males, in their 40's, then it does other groups. This means that for the smaller groups the data will be heavily biased by the small number of examples. Many of the above 50K examples I found were biased to the above group for the dev and test set results. I would say its underfitting since the data is heavily biased towards a subset of people, meaning outliers will only be mostly compared to the above subset which is likely not representative of the true values. Although it could potentially overfit in some cases if the number of data points it compared to only represented a small subset of the whole population. Yes this does have potential social issues because it could lead you to make decisions on people (like how much credit you should give them) based on heavily biased data. There have been real life examples such as who to give parole to in prison systems, that have fallen victim to this. It is vitally important for people who use data science or machine learning to actually question the output of their work as well as their training data.

3. Q: What numpy tricks did you use to speed up your program so that it can be fast enough to print the training error?

   (a) I used quite a few actually. I used broadcasting when subtracting the data point from the training array. For Euclidean distance I used np.linalg.norm(...,axis=1). For sorting I used np.argsort() and used that array for finding the indexes of the nearest neighbors. I also used slicing. I pretty used most of the techniques you recommended and saw huge performance boosts.

4. How many seconds does it take to print the training and dev errors for k = 99 on ENGR servers?

   (a) Mine took 14.951u seconds which is pretty good I think.

5. What is a Voronoi diagram (shown in k-NN slides)? How does it relate to k-NN?

   (a) A Voronoi diagram is way to partition a space into regions such that the distance from any point in a sub-region is closest to the point that defines the sub-region. It has some really useful properties such as each region being convex. It relates to k-NN because a Voronoi diagram of a set of data points would be equivalent to a 1-NN representation. This would be useful because it would greatly speed up the classification process in this case since all you would have to check would be the label for the Voronoi cell the test point falls in. No need to calculate all the distances.

5 Debriefing

1. Approximately how many hours did you spend on this assignment?

   (a) I spent approximately 15-20 hours on this assignment.

2. Would you rate it as easy, moderate, or difficult?

   (a) Moderate. There were a few command line, python, and numpy skills I never used before so it involved looking into them.

3. . Did you work on it mostly alone, or mostly with other people?

   (a) Alone with the help of the internet.

4. How deeply do you feel you understand the material it covers (0%100%)?

(a) 85%. I would have liked to speed my code up more but everything was at least functioning in reasonable time limits. I would say my gaps come from some of the programming aspects rather than understanding k-NN which is a pretty straight forward algorithm.

5. Any other comments?

   (a) This project taught me a lot about what feature maps really are and how data quality is really a driving force for Machine Learning. I was a little unsure of how the error rate was being derived so perhaps next time a slide on error rates calculations would be useful.

References

[1] M. Frankel, "Here's the Average American Household Income – How Do You Compare?," 2016.