Homework 1

# Part 1: Data (Pre-)Processing (Feature Map)

1. What are the positive % of training data?

In the training data, there are 5000 total observations. Of those, 1251 people make over 50k. Therefore, the percent positive in the training set is 25.02%.

What about the dev set?

In the dev set, there are 1000 total observations. Of those, 236 people make over 50k. Therefore, the percent positive in the dev set is 23.60%.

Does it make sense given your knowledge of the average per capita income in the US?

Yes, this makes sense, because the data is fairly old. It dates back to 1994, and the course lecture mentioned that the average per capita income that year was about 27K. This is consistent with data from the Social Security Administration, which shows that the average net compensation in 1993 was USD22.786.73.

Reference: <https://www.ssa.gov/OACT/COLA/central.html>

2. What are the youngest and oldest ages in the training set?

The youngest age in the training set is 17 and the oldest age is 90 years old.

What are the least and most amounts of hours per week people in this set work?

The minimum number of hours worked in the training dataset is 1 hour (1 person), and the maximum is 99 hours (5 people).

3. Why do we need to binarize all categorical fields?

Humans can interpret and compare categorical fields like Male/Female or Bachelor/Associate/Doctorate in relation to each other. However, a machine cannot do this. The field names mean nothing to the computer. By binarizing, we translate each feature into the absence/presence (0/1) of a certain characteristic, which can then be used to calculate distances. More efficient, less memory

Harder to work with when not binarized

factorization

4. If we do not count *age* and *hours*, what is the maximum possible Euclidean and Manhattan distances between two training examples? Explain.

vector of all cat - 1

Square each individual cat

5. Why do we **not** want to binarize the two numeric fields, *age* and *hours*?

What if we did?

How should we define the distances on these two dimensions so that each field has equal weight? (In other words, the distance induced by each field should be bounded by **2** (N.B.: not 1! why?)).

## Hint: numerical fields / 50, e.g., “age” / 50 so that the max distance on “age” is also 2.

…

6. How many features do you have in total?

Without binarizing age and hours worked per week, there would be 92 features (90 for the categorical variables and 2 for the numerical variables).

How many features do you allocate for each of the 9 fields?

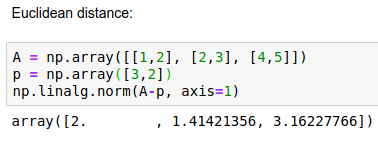
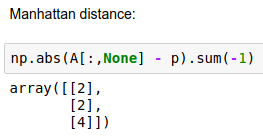
I would allocate the features as follows:

* Age: 1
* Workclass: 7
* Education: 16
* Relationship status: 7
* Occupation: 14
* Ethnicity: 5
* Sex: 2
* Hours worked per week: 1
* Country of origin: 39

7. How many features would you have in total if you binarize all fields?

If we binarized all fields, including age and hours worked per week, there would be 230 features (90 for the categorical variables, + 67 for age, + 73 for hours worked per week).

# Part 2: Calculating Manhattan and Euclidean Distances



What does axis=1 mean?

…

1. Find the five (5) people closest to the last person (in Euclidean distance) in dev, and report their distances.

…

2. Redo the above using Manhattan distance.

…

3. What are the 5-NN predictions for this person (Euclidean and Manhattan)?

…

Are these predictions correct?

…

4. Redo all the above using 9-NN (i.e., find top-9 people closest to this person first).

…

**Part 3: *k*-Nearest Neighbor Classification**

1. Implement the basic *k*-NN classifier (with the default Euclidean distance).

Is there any work in training after finishing the feature map?

No, once the feature map is complete, the training set is left aside. The algorithm then moves on to the dev/test set to predict the new labels.

What’s the time complexity of *k*-NN to test one example (dimensionality *d*, size of training set |*D*|)?

…

Do you really need to sort the distances first and then choose the top *k*? Hint: there is a faster way to choose top *k* without sorting.

…

2. Why does the *k* in *k*-NN have to be an odd number?

It is best to use an odd *k* to serve as a tiebreaker.

3. Evaluate *k*-NN on the dev set and report the error rate and predicted positive rate for *k* = 1*,* 3*,* 5*,* 7*,* 9*,* 99*,* 999*,* 9999.

What’s your best error rate on dev, and where did you get it? (Hint: 1-NN dev error should e ∼23% and its positive % should be ∼27%).

4. Now report both training and testing errors (your code needs to run a lot faster! See Question 4.3 for hints. See also week 2 videos for numpy and linear algebra tutorials, in case you’re not familiar with the “Matlab”-style of thinking which is inherited by numpy):

k=1 train\_err xx.x (+:xx.x) dev\_err xx.x (+:xx.x)

k=3 …

…

k=9999 …

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

…

5. 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?

…

What does *k* = ∞ actually do? Is it extreme overfitting or underfitting? What about *k* = 1?

…

6. Redo the evaluation using Manhattan distance. Better or worse? Any advantage of Manhattan

distance?

…

7. Redo the evaluation using all-binarized features (with Euclidean). Better or worse? Does it make sense?

…

**Part 4: Deployment**

Now try more *k*’s and take your best model and run it on the semi-blind test data, and produce income.test.predicted, which has the same format as the training and dev files.

At which *k* and with which distance did you achieve the best dev results?

…

What’s your best dev error rates and the corresponding positive ratios?

…

What’s the positive ratio on the test data?

…

Part of your grade will depend on the accuracy of income.test.predicted.

**Part 5: Observations**

1. Summarize the major drawbacks of *k*-NN that you observed by doing this HW. There are a lot!

…

2. 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?

…

3. What numpy tricks did you use to speed up your program so that it can be fast enough to print the training error? Hint: (a) broadcasting (such as matrix - vector); (b) np.linalg.norm(..., axis=1); (c) np.argsort() or np.argpartition(); (d) slicing. The main idea is to do as much computation in the vector-matrix format as possible (i.e., the Matlab philosophy), and as little in Python as possible.

…

4. How many seconds does it take to print the training and dev errors for *k* = 99 on ENGR servers? Hint: use time python ... and report the user time instead of the real time. (Mine was about 14 seconds).

…

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

…

**Part 6: Debriefing**

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

…

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

Overall, the whole homework was moderate. Part 1 was easy. Part 2 was confusing, because

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

I worked on my own.

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

…

5. Any other comments?

…