1. Hands-on Exploration of the Income Dataset
    1.1. What percentage of the training data has a positive label (>50K)? What about the dev set? Does it make sense given your knowledge of the average US per capita income?
        1.1.1. Train = 25.01%, Dev = 23.58%. The data reflects that most people do not earn more than the national average which seems correct. The dev set has a smaller data set which could have skewed its data either higher or lower depending on which entries it had. Though the percentage was still similar to train data.

    1.2. What are the youngest and oldest ages in the training set? What are the least and most hours per week people work in the training set?
        1.2.1. Youngest = 17, Oldest = 90, Least Hr = 1, Most Hr = 99.

    1.3. Why do we need to binarize all categorical fields?
        1.3.1. We want to binarize categories to make sure there is no hierarchy for non-numerical values like id, edu, marriage.

    1.4. Why do we not want to binarize the two numerical fields, age and hours?
        1.4.1. Those values are already numbers and can stay as such.

    1.5. How should we deal with the two numerical fields? Use them as-is, or normalize them (e.g., age / 100)?
        1.5.1. You can use them as-is, but it may cause models to give more weight to larger values, which could distort their value. For algorithms like KNN and others, normalization is generally preferred to ensure that all features contribute equally and that the data points are properly scaled.

    1.6. How many features do you have in total (i.e., the dimensionality)? (Hint: around 90. If you binarize all fields, it would be around 230)
        1.6.1. Train.5k = 92, Dev = 80.

2. Data Preprocessing and Feature Extraction I: Naive Binarization
    2.1. Although pandas.get_dummies() is very handy for one-hot encoding, it’s absolutely impossible to be used in machine learning. Why?
        2.1.1. It can generate a large number of new columns, which may increase model complexity and distort feature relationships, potentially leading to overfitting. It may also cause problems with column representation and mismatch models.

    2.2. After implementing the naive binarization to the real training set (NOT the toy set), what is the feature dimension? Does it match with the result from Q1.6?
        2.2.1. Train.5k = 230, Dev = 195. Yes they match the estimated amounts (excluding id and target).

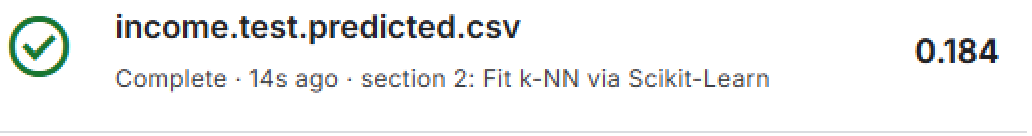
2.3. What's your best error rate on dev? Which k achieves this best error rate?
   2.3.1. Best Dev = 15.7% on k=67.

2.4. When k = 1, is training error 0%? Why or why not? Look at the training data to confirm your answer.
   2.4.1. k=1 train_err 1.5% | dev_err 23.2%. No, since it is not checking itself, there may be different labels between the comparing points. .

2.5. What trends (train and dev error rates and positive ratios, and running speed) do you observe with increasing k?
   2.5.1. The error rate of train tends to increase while dev decreases. The positive percentage both decrease as k increases.

2.6. What does k = ∞ actually do? Is it extreme overfitting or underfitting? What about k = 1?
   2.6.1. k = ∞ causes extreme underfitting because the model becomes too rigid. It treats all training points equally and doesn't learn meaningful boundaries, leading to poor predictions.
   k = 1 causes overfitting because the model relies too heavily on the closest neighbor. It's very sensitive to noise and small differences, which can lead to poor generalization.

2.7. Using your best model (in terms of dev error rate), predict the semi-blind test data, and submit it to Kaggle (follow instructions from Part 5). What are your error rate and ranking on the public leaderboard? Take a screenshot. Hint: your public error rate should be ~ 18.5%.

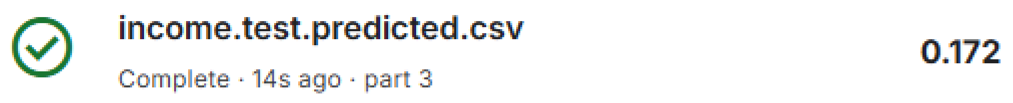

income.test.predicted.csv
Complete · 14s ago · section 2: Fit k-NN via Scikit-Learn
0.184

3. Data Preprocessing and Feature Extraction II: Smart Binarization
   3.1. Re-execute all experiments with varying values of k (Part 2, Q 4a) and report the new results. Do you notice any performance improvements compared to the initial results? If so, why? If not, why do you think that is?
      3.1.1. k= 1 train_err 1.5% | dev_err 23.1%. The dev error rate has slightly improved.

   3.2. Again, rerun all experiments with varying values of k and report the results (Part 2, Question 4a). Do you notice any performance improvements? If so, why? If not, why do you think that is?

3.2.1. k= 1 train_err 1.5% | dev_err 23.9% | k= 99 train_err 17.8% |dev_err 15.6%. Though the k=1 is about the same if not worse. The final k is a significant error improvement seeing around 15%.

3.3.



income.test.predicted.csv

Complete · 14s ago · part 3

0.172

4. Implement your own k-Nearest Neighbor Classifiers

4.1. Before implementing your k-NN classifier, try to verify the distances from your implementation with those from the sklearn implementation. What are the (Euclidean and Manhattan) distances between the query person above (the first in dev set) and the top-3 people listed above? Report results from both sklearn and your own implementation.

4.1.1. Euclidean:

id	sklearn	Own_Euc	Own_Man
4872	0.334419	0.334419	0.389992
4787	1.415275	1.415275	2.054795
2591	1.416747	1.416747	2.102600

Manhattan:

id	sklearn	Own_Euc	Own_Man
4872	0.389992	0.334419	0.389992
4787	2.054795	1.415275	2.054795
1084	2.102041	1.417890	2.102041

4.2. Is there any work in training after the feature map (i.e., after all fields become features)?

4.2.1. No, the field transformation into features happens and then is stored for later use. Therefore there is no training afterwards.

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

4.3.1. O(d) for each node * O(|D|) to find smallest k distance => O(d * |D|)

4.4. 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.

4.4.1. You never need to sort anything. It just makes things faster sometimes. If you are just looking for the top k then you can loop through them sequentially or use some other algorithm.

4.5. What numpy tricks did you use to speed up your program so that it can be fast enough to print the training error? Hint: (i) broadcasting (such as matrix - vector); (ii) np.linalg.norm(..., axis=1); (iii) np.argsort() or np.argpartition(); (iv) 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.

4.5.1. I have not fully customized my algorithm for best efficiency. But at this point, I have the basic linalg.norm and broadcasting vectors for distance. As well as argsort to find nearest indices.

4.5.2. Mine may have had some lag or traffic. I am VPN in from farther away. My time was 111.2 seconds.

4.6. Redo the evaluation using Manhattan distance (for k = 1 ~ 99). Better or worse?

4.6.1. Best k=41 train_err 17.3% (+:-3.2%) dev_err 14.1% (+:-3.2%) ++. This was a better outcome than euclidean and other versions I had.

5. Deployment

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

5.1.1. k=41, manhattan distance

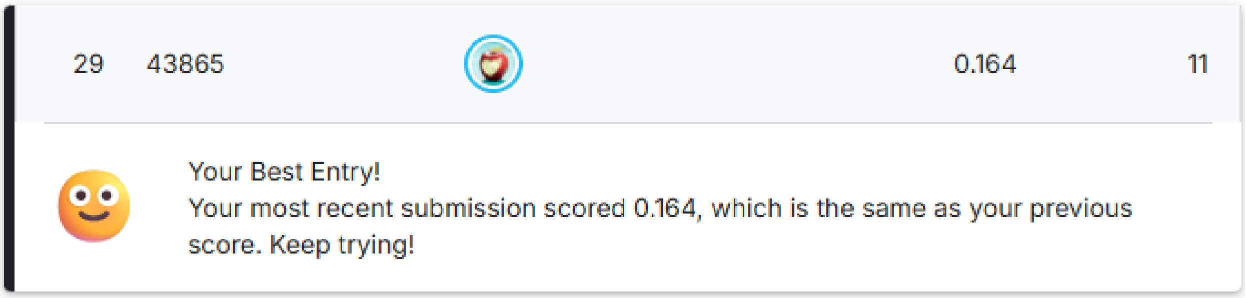
5.2. What's your best dev error rates and the corresponding positive ratios?

5.2.1. dev_err=14.1% (+:20.7%)

5.3. What's the positive ratio on test?

5.3.1. Test set positive prediction rate: 21.0%

5.4. What's your best rank on the public leaderboard? Take a screenshot. How many submissions did you use?



29 43865 0.164 11

Your Best Entry!
Your most recent submission scored 0.164, which is the same as your previous score. Keep trying!

5.4.1. There were two code errors in that.

6. Observations

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

6.1.1. The more k you try to sample the longer it takes. You are limited to the number of features and the more you have the more fitting through maybe over fitting but also the slower.

6.2. 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?

   6.2.1. Best models in this homework often hover near the edge of overfitting. They tend to closely follow patterns in the training data, including any biases, which can lead to exaggerated predictions on the dev and test sets. This behavior raises concerns about fairness, especially when biased training data influences decisions that affect real people.

7. Debriefing

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

   7.1.1. About 20 hours. I only had time to work on it after my job, and I ran into problems with helper functions, debug prints, and attempts to optimize that sometimes made the models worse.

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

   7.2.1. It should have been easy, but I made it more complicated. With my work schedule, it took more time than I would have liked.

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

   7.3.1. All alone

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

   7.4.1. Fairly well, though I was unable to improve the model much beyond what was shown in the homework notes.

7.5. Any other comments?

   7.5.1. None at this moment.