

Hands-on Exploration of the Income Dataset

- 1) Percentage of training data with positive label (>50K)?
 - a) training data: 25.02 %
 - b) dev dataset: 23.60%
 - c) The percentages were 25.02% and 23.60% respectively. Given that the average US income is ~ \$41,000 (according to latest US census) the percentages in the dataset are not surprising.
- 2) Q: What are the youngest and oldest ages in the training set? Youngest is 17 and oldest is 90.
- 3) Most hours worked: 99. Least amount of hours worked: 1.
- 4) Q Why do we need to binarize all categorical fields? Most algorithms cannot effectively handle text based representation of data, so we need to convert to binary for each text value.
Q: Why do we not want to binarize the two numerical fields, age and hours?
Age and hours are already numerical, they could be rescaled but treating them as binary fundamentally mistreats the data.
- 5) Q: How should we deal with the two numerical fields? Just as is, or normalize them (say, age / 100)? I would normalize them, otherwise their respective numerical ranges may have too strong (or too weak) of an influence on the model.

Q: How many features do you have in total (i.e., the dimensionality)? There are 92 features after binarizing the categorical fields.

Data Preprocessing and Feature Extraction I: Naive Binarization

Although `pandas.get_dummies()` is very handy for one-hot encoding, it's absolutely impossible to be used in machine learning. Why?

`get_dummies()` would not be able to handle previously unseen values in the new data / blind data that we are trying to predict.

- 6) Q 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 Part 1 Q5?

There are 230 features after naive binarization of the real training set, which is different from the result in Q1 where I only binarized the categorical fields.

7) Q: what's your best error rate on dev? Which k achieves this best error rate? (Hint: 1-NN dev error should be ~23% and the best dev error should be ~16%). (1 pt)

- The best dev error is 15.6% for k=69 and 1-NN dev error is 23.2%

8) When $k = 1$, training error ~1.5%. This is overfitting but the error is not exactly 0% due to the presence of some nearly identical data points with different target labels.


9) Trends that can be observed. Dev error rate is highest at $k = 1$, then decreases until $k = 9$. After that it stays relatively flat. Train error increases from $k = 1$ to $k = 9$, then stays relatively flat afterwards.

For both datasets, the positive ratio appears to increase as k increases.

10) For $k = 1$, training error is minimized at (~1.5%) but dev error is high (~23.2%). This looks like a case of overfitting. For very large k (infinity), intuitively this seems to be extreme underfitting. Error rate for both datasets is high $> 20\%$. When k is very large, the algorithm attempts to separate the data into many different clusters.

11) 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?

Error rate is 0.194 and ranking is 102 as of Oct 21, 2024

Submission and Description		Public Score ⓘ	Select
	output_n.csv Complete · 31s ago	0.194	<input type="checkbox"/>


12) 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? (1.5 pts)

The best dev error is 20.7% at $k = 33$ and 1-NN dev error is 26.9%. The performance is actually worse than the performance with only naive binarization. I believe this is because the numerical features are not normalized yet and have too exaggerated of an effect on the prediction.

13) 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? (1.5 pts) (Hint: 1-NN dev error should be ~24% and the best dev error should be ~14–15%)

After applying smart binarization AND numerical feature scaling, I am seeing improved prediction performance. The best dev error is 14.3%, for $k=41$ and 1-NN dev error is 23.7%.

14) New error rate is 17% and public ranking is ~71.

 **output_new.csv**
Complete · 3h ago

0.170

☐

Implement your own k-Nearest Neighbor Classifiers

15) 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.

- Sklearn (index, distance): (4872, 0.334), (4787, 1.415), (2591, 1.417)
- Own Implementation (index, distance): (4872, 0.33440), (4787, 1.41528), (2591, 1.41674)

16) (a) Q: Is there any work in training after the feature map (i.e., after all fields become features)? (0.25 pts)

Yes, training data needs to be stored in memory, i.e. `knn.fit()`

(b) Q: What's the time complexity of k-NN to test one example (dimensionality d ,

size of training set $|D|$)? (0.75 pt) If you do not have a CS background, please state it here.

$O(d * |D| + |D| + k)$

(c) Q: 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. (0.5 pts)

Not necessarily. It's possible to perform top k using quick select.

(d) Q: 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. (1 pt)

Broadcasting (matrix - vector), `np.linalg.norm`, and `np.argpartition` (for selecting k smallest distances)

(e) Q: 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 real time. (Mine was 14 seconds). (0.5 pts)

Took ~11.2 seconds for $k = 99$.

Redo the evaluation using Manhattan distance (for $k = 1 \sim 99$). Better or worse?

$k = 3$, produced 12.1% error rate on dev. The positive ratio is 23.8%. Slightly worse than Euclidean.

Deployment

Let's try more k 's and take your best model (according to dev error rate) and run it on the semi-blind test data, and produce `income.test.predicted.csv`, which has the same format as the training and dev files.

Q1: At which k and with which distance did you achieve the best dev results? $k = 3$, euclidean

Q2: What's your best dev error rates and the corresponding positive ratios? 10.9% error rate and 24.0% positive ratio.

Q3: What's the positive ratio on test? 26.6%

What's your best rank on the public leaderboard? How many submissions did you use?

Best rank is 2 and I used 28 submissions in total.

Observations:

Q Summarize the major drawbacks of k-NN that you observed by doing this HW. There are a lot! (1 pt) Hint: The most obvious one is: Why are all fields treated equally? Which field should be more important?

Drawbacks: - All features are equally important in making the prediction, which goes against the intuition that in making a prediction some factors will have a higher "weight" than others.

- Storing the entire dataset in memory may not be feasible for large datasets.

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? (1.5 pts) Hint: for example, compare the true positive % vs. your predicted positive % on the dev set. What about the positive % given gender? (e.g., for all females in the dev set, compare the true positive % and your predicted one.) Or race?

It does appear that the better performing models exaggerate existing bias in the training data. I think this would be due to overfitting. This is definitely a potential social issue. If these models were somehow used in making public policy decisions, they could reinforce existing societal problems instead of guiding decision makers to make better decisions that would bring more fairness. More specifically, there may be a tendency to pay females less than males for the same job and years of experience if these models were involved in constructing pay bands.

We can see in the training data that there is a greater percentage of males with income > 50k compared to females (~28% vs ~13%)

Debriefing

I spent around 8 hours on this homework.

I would rate it as moderate in difficulty.

I worked on it alone.

I understand the material 85%.

Had a lot of fun with this assignment. My only complaint is I felt like I am dealing with a black box and refined my model through blind trial and error.