Hands-on Exploration of the Income Dataset

1. Q: What percentage of the training data has a positive label (>50K)? (This is known as the positive %). What about the dev set? Does it make sense given your knowledge of the average US per capita income?
Answer:
25.02% of the training data has a positive label (>50K).
23.60% of the dev set has a positive label (>50K).

If we take a look at US 2021 per capita income, 75% of people earn less than the US per capita income. The result I got concludes that 76% of people earn less than the US per capita income.

2. Q: What are the youngest and oldest ages in the training set? What are the least and most amounts of hours per week do people in this set work?
Answer:
The youngest age:17
The Oldest:90.
The least hours per week: 1
The most hours per week: 99.

3. Q: Why do we need to binarize all categorical fields?
Q: Why we do not want to binarize the two numerical fields, age and hours?
Answer: Most machine learning models only take numerical variables, categorical variables must be pre-processed. We must transform these categorical variables to integers in order for the model to interpret and extract useful information.
It's generally advisable to retain the continuity of numerical attributes like "age" and "hours per week." These continuous variables hold valuable information that can be leveraged effectively. To work with them, we employ techniques like regression models, decision trees, or, when required, feature scaling to maintain their integrity. Binarization, on the other hand, is typically reserved for categorical variables. It's a process of converting categorical data into a binary format, commonly used for algorithms like logistic regression. This approach allows us to adapt categorical data for specific modelling needs, while preserving the rich nuances within continuous numerical attributes.
4. Q: How should we deal with the two numerical fields? Just as is, or normalize them (say, age / 100)?

Normalization standardizes numerical attributes like "age" and "hours per week" to a common range, often 0 to 1 or -1 to 1. This allows for direct comparison with categorical fields, ensuring that both types of data share a consistent scale. Categorical variables have discrete values, so normalizing numeric attributes aligns their scaling for fair analysis.

5. Q: How many features do you have in total (i.e., the dimensionality)?

Answer:

We have 90 features in total.

Data Preprocessing and Feature Extraction I: Naive Binarization

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

Answer:

The get_dummies function in Pandas lacks handling for unseen data, a crucial consideration in machine learning. It encodes features solely based on observed values, potentially causing issues when the number of unique values in the test data differs from training data. This inconsistency can impact model generalization. Moreover, Pandas may lack stability and robustness in some scenarios.

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?

Answer:

The feature dimension after the naive binarization is 230. No, it does not match the results from Part 1 Q5.

Q: What's your best error rate on dev? Which k achieves this best error rate?

Answer:

Best error rate = 16.00%

k = 54

Q: When k = 1, is training error 0%? Why or why not?

Answer:

No, it is not zero there is always overfitting.

Q: What trends (train and dev error rates and positive ratios, and running speed) do you observe with increasing k?

Answer:

I observed that with increasing k train, dev error rates and positive ratios decreases while running speed does not.

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

Answer:

In the K-Nearest Neighbours (KNN) algorithm, the choice of 'k' is crucial. A large 'k' (very large number) means that the model considers all data points as neighbours, effectively underfitting the data. It relies heavily on the overall dataset, resulting in a high-bias, low-variance model that makes predictions close to the dataset's mean or mode.
On the other hand, setting 'k' to 1 means the model only looks at the nearest neighbour, which can lead to overfitting. It's extremely sensitive to noise and outliers, as it often predicts based on the closest point, which might not represent the true data pattern. This results in a low-bias, high-variance model. Selecting the right 'k' value is vital to balance bias and variance in KNN.

Data Preprocessing and Feature Extraction II: Smart Binarization

Q. 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?
Answer:
No, I noticed there was no improvement in performance comparing Smart encoding to Naïve Encoding. . Smart encoding did not transform whereas Naïve encoding binarized numerical fields leaving less room for error.

Q. 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?
Answer:
Yes, I noticed that Smart-Scaled encoding gives less error rate when compared to Naïve encoding. Smart scaled encoding scales(MinMaxScaler) the numerical values hence performing better.

Implement your own k-Nearest Neighbor Classifiers

Q. 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 Results

```
Euclidean Distances:
Neighbor 1: 0.33
Neighbor 2: 1.42
Neighbor 3: 1.42

Manhattan Distances:
```

```
Neighbor 1: 0.39
Neighbor 2: 2.05
Neighbor 3: 2.10
```

My KNN(Euclidean)

```
Top 3 closest data points for Euclidean distance:
Index 1: Index 2182, Distance: 0.33
Index 2: Index 794, Distance: 1.42
Index 3: Index 898, Distance: 1.42

Top 3 closest data points for Manhattan distance:
Index 1: Index 898, Distance: 2.05
Index 2: Index 2182, Distance: 0.39
Index 3: Index 1151, Distance: 2.10
```

My KNN(Manhattan)

```
Top 3 closest data points for Euclidean distance:
Index 1: Index 3225, Distance: 1.42
Index 2: Index 3120, Distance: 0.33
Index 3: Index 1336, Distance: 1.42

Top 3 closest data points for Manhattan distance:
Index 1: Index 3120, Distance: 0.39
Index 2: Index 1336, Distance: 2.10
Index 3: Index 3252, Distance: 2.10
```

2. Implement your own k-NN classifier (with the default Euclidean distance).
Q: Is there any work in training after the feature map (i.e., after all fields become features)?
Answer:
Following feature transformation, the data is divided into training and testing sets. A loop cycles through various k values to optimize the KNN model. This involves training the model with different k values, making predictions on the test set, and assessing prediction accuracy to identify the best hyperparameter.

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

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.
Answer:
To only find the top-k elements it is better to use np.argpartition before opting for sorting the distances array. Using the np.argpartition I can find the indices of the k-smallest distance in a faster way.

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.
Answer:
Calculating the Euclidean distances between a single data point 'x' and all the data points in 'X_train' efficiently, without needing explicit loops. I do this using NumPy's 'np.linalg.norm' function with 'axis=1' to compute distances along the specified axis, resulting in a one-dimensional array of distances.

To find the indices of the k-nearest neighbours, I utilize 'np.argpartition.' This function efficiently performs a partial sort on the distances array and gives us the indices of the k smallest distances. It's a quicker way to identify the nearest neighbours compared to a full sort, which can save computational resources and speed up the process of finding the k-nearest neighbours.

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 the real time. (Mine was about 14 seconds).
Answer:
It takes 53 seconds

3. Redo the evaluation using Manhattan distance. Better or worse?
Answer:
Comparing them and looking at the dev dataset both of them have the same error rates.

Deployment

Q: At which k and with which distance did you achieve the best dev results?
Answer:
They have very similar error rates on the dev dataset. At k: 81 and using Euclidean distance I got a best accuracy of 84.5% and a best error rate of 15.50%.

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

My positive ratio was around 19-20%.

Q: What’s the positive ratio on test?
Answer:

Around 20-21%.

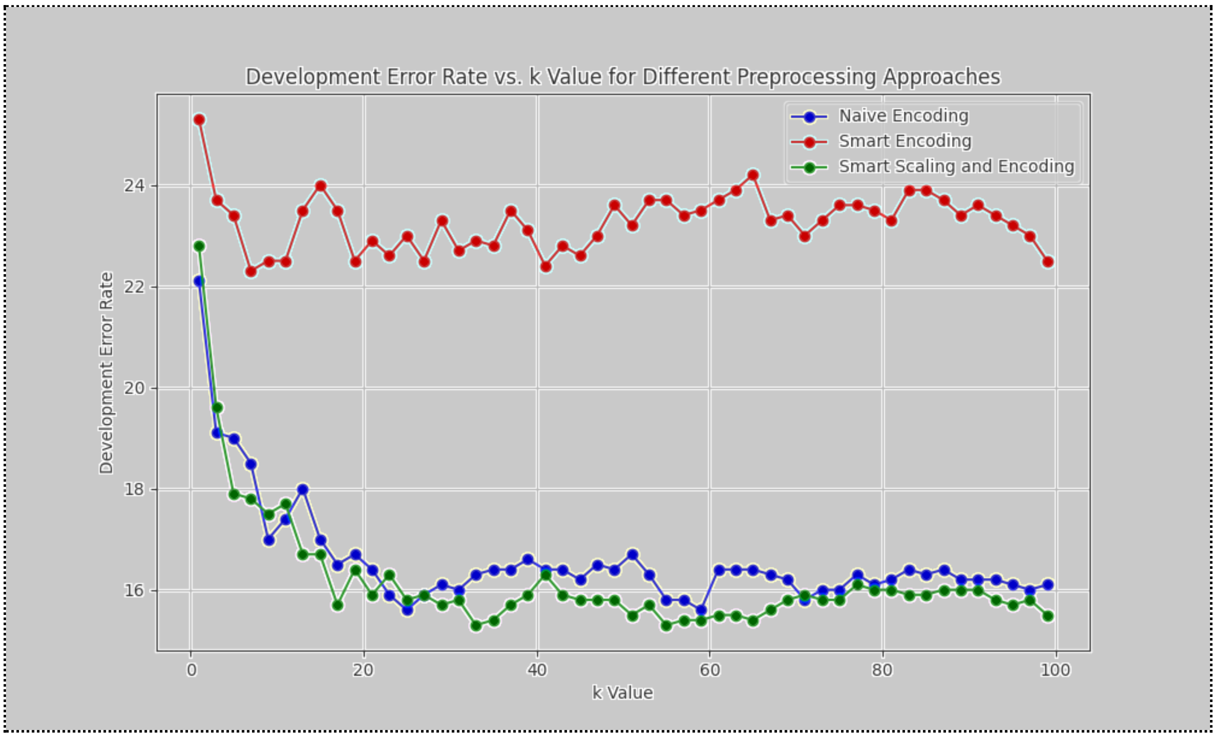
Observations

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

- All the features are equally important
- It cannot estimate the continuous values
- It has a fixed k-value which has to be updated when the data set evolves.

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? 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?
Answer: When we look at the best-performing models, we notice that they have a high bias over the training data. Given that the Age(30s) is a highly positive feature, the predicted values for the same feature using our model are likely to be even higher on the dev set. Our model is unable to capture the underlying data pattern due to underfitting. This exaggeration of the data's existing bias is a social issue.

Extra Credit Question



Debrief

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

    I spent around 1 hour every day.

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

    Moderate

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

    Mostly with other people

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

    70%

5. Any other comments?

    No.