Machine Learning HW1: Feature Map and k-NN

Apoorv Malik

October 23rd 2023

1 Hands-on Exploration of the Income Dataset

1. About 25% of the training data has a positive label (>50K). The current average US per-capita income is around 38K, so this percentage makes sense.

2. In this dataset, the youngest age is 17, and the oldest age is 90. The least amount of hours-per-week that people work is 1, and the most amount of hours-per-week that people work is 99.

3. Most Machine Learning Algorithms work by performing mathematical operations on the input data. Categorical data, by nature, doesn't have any mathematical meaning, therefore it is challenging for the ML algorithms to interact with it directly. Binarizing categorical data provides a numerical representation for these algorithms to work on. Also, the k-NN algorithm relies on the distance metric for classification. Having binarized numerical values for categorical features ensures each category has an equal effect on the distance computation.

   If we were to binarize a numerical field, we would need a binary column for each unique value in that field. For fields with many unique values, this can lead to a massive increase in dimensionality, which can be computationally expensive and may lead to overfitting. Binarizing would also reduce the granularity of the data. For example, binarizing age would mean representing everyone of age 25 and everyone of age 35 with the same value (either 0 or 1). This would eventually lead to reduced model performance.

4. For many machine learning algorithms including k-NN, normalizing numerical fields can lead to better model performance. A feature that spans a large numerical range can unduly influence the distance computation. Therefore, it is absolutely necessary for all features to be in the same range (in our case between 0-2) for getting the most effective k-NN model.

5. After binarizing the categorical fields, we have 92 total fields. There are 90 binarized categorical fields and 2 numerical fields (age and hours-per-week).

2 Data Preprocessing and Feature Extraction I: Naive Binarization

2. If we apply pd.get_dummies() to both the train and test sets, we might end up with different numbers of features for each set. This discrepancy can arise because certain categorical values may be present in the training set but absent in the test set, or vice versa. As a result, the train and test sets could

have different dimensionalities. Therefore, a machine learning model trained on this data would not work, as it expects consistent input dimensionality for training and prediction data.

3. I have 230 features after implementing the naive binarization on the training dataset. No, it doesn't match the results from Part 1 $Q5$ (I got 90 for that question, because I only counted unique categorical values and ignored numerical values).

4. Here are the results and some statistics for Naive Binarization

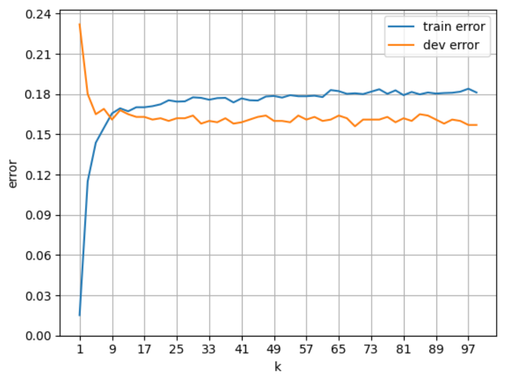
   - The best dev error is **15.6**% at $k = 69$
   - 1-NN dev error is **23.2**%

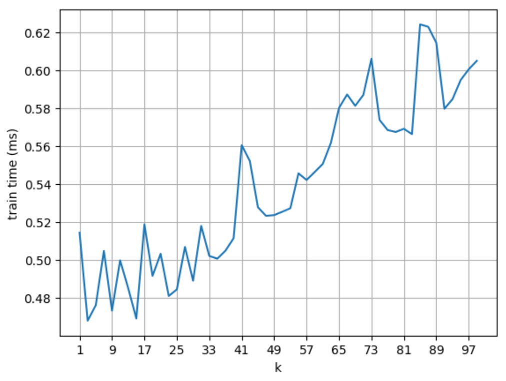
```
[k]      [Train Error]     [Dev Error]       [Train Positive Rate]     [Dev Positive Rate]       [Time (ms)]
1        0.015             0.232             0.251                     0.248                     803
3        0.115             0.180             0.232                     0.222                     493
5        0.144             0.165             0.229                     0.197                     502
7        0.155             0.169             0.219                     0.209                     507
9        0.166             0.161             0.217                     0.199                     523
11       0.169             0.168             0.208                     0.188                     565
13       0.167             0.165             0.203                     0.181                     593
15       0.170             0.163             0.199                     0.185                     574
17       0.170             0.163             0.194                     0.187                     516
19       0.171             0.161             0.192                     0.185                     529
21       0.172             0.162             0.191                     0.188                     615
23       0.175             0.160             0.191                     0.182                     666
25       0.174             0.162             0.188                     0.176                     535
27       0.175             0.162             0.189                     0.176                     587
29       0.178             0.164             0.184                     0.178                     574
31       0.177             0.158             0.183                     0.178                     590
33       0.176             0.160             0.190                     0.186                     540
35       0.177             0.159             0.185                     0.179                     549
37       0.177             0.162             0.183                     0.178                     526
39       0.174             0.158             0.184                     0.184                     520
41       0.177             0.159             0.185                     0.185                     561
43       0.175             0.161             0.182                     0.183                     596
45       0.175             0.163             0.183                     0.181                     581
47       0.178             0.164             0.182                     0.180                     539
49       0.179             0.160             0.179                     0.176                     532
51       0.177             0.160             0.178                     0.182                     542
53       0.179             0.159             0.178                     0.177                     660
55       0.178             0.164             0.181                     0.178                     571
57       0.178             0.161             0.179                     0.177                     547
59       0.179             0.163             0.178                     0.181                     537
61       0.178             0.160             0.176                     0.178                     547
63       0.183             0.161             0.177                     0.179                     566
65       0.182             0.164             0.180                     0.176                     618
67       0.180             0.162             0.177                     0.178                     593
69       0.181             0.156             0.178                     0.176                     643
71       0.180             0.161             0.177                     0.171                     606
73       0.182             0.161             0.173                     0.177                     604
75       0.184             0.161             0.175                     0.177                     596
77       0.180             0.163             0.177                     0.177                     606
79       0.183             0.159             0.175                     0.173                     659
81       0.179             0.162             0.175                     0.174                     623
83       0.182             0.160             0.175                     0.172                     712
85       0.180             0.165             0.175                     0.173                     617
87       0.181             0.164             0.176                     0.174                     586
89       0.180             0.161             0.174                     0.173                     584
91       0.181             0.158             0.173                     0.174                     599
93       0.181             0.161             0.172                     0.173                     593
95       0.182             0.160             0.174                     0.174                     593
97       0.184             0.157             0.173                     0.175                     594
99       0.181             0.157             0.175                     0.173                     590
```

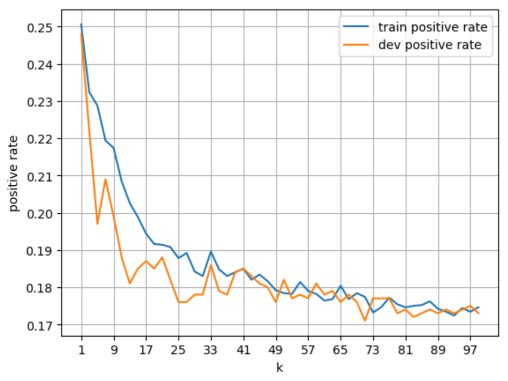
5. When $k = 1$, the training error is **1.5**%. Although this is the case of extreme overfitting, this is not exactly **0**%. This is because there are a few nearly identical data points in the training data with different labels.

6. These are the trends observed for

   - The dev error is maximum at $k = 1$ (about 23.2%), it then quickly decreases till $k = 9$, and after that it stays relatively flat till k=99 (as seen in the graph). At $k = 9$, the model has lower dev error when compared with train error (16.1% vs 16.6%).
   - A similar trend is seen for the train error, which increases till k=9, and then relatively stays flat till $k = 99$. For $9 \leq k \leq 99$, the dev error is less than the train error.
   - The running time graph (below) shows the cumulative prediction running time of train and test sets.
   - Both train and dev positive ratios decrease with increasing k.







7.
   - For k=1, the training error is minimum (1.5%), but the dev error is high (23.2%). This looks like extreme overfitting.
   - For $k = \infty$ (k=5000), the training error is maximum (25%), and the dev error is also maximum (23.6%). This looks like extreme underfitting.

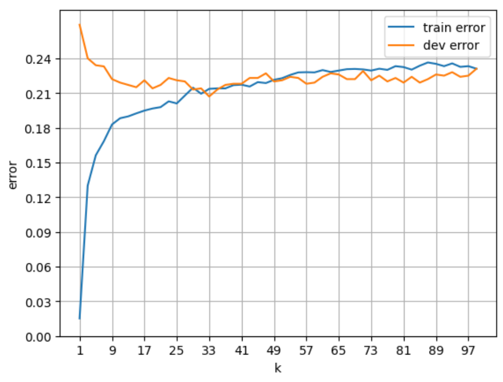
3 Data Preprocessing and Feature Extraction II: Smart Binarization

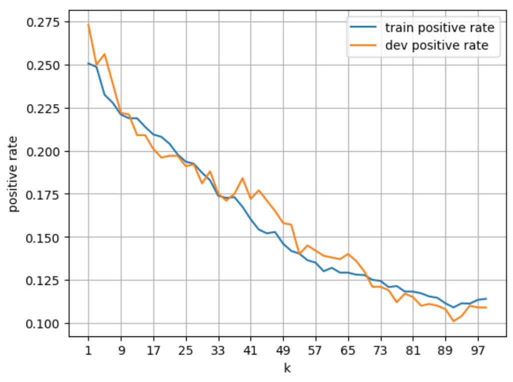
1. No, there is no performance improvements when compared with the results of Naive Binarization experiment. Infact, this model is performing worse than before. This is because, if some features (like the binarized categorical ones) have values in the range of 0 and 1, while others (like some numerical features) have a much broader range, the algorithm might be biased towards the features with larger magnitudes. This can lead to suboptimal performance.

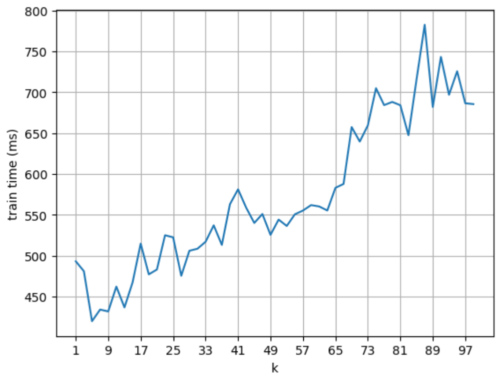
   These are the results for Smart Binarization without Scaling (Tested on M1 Pro processor):

   - The best dev error is **20.7**% at $k = 33$.
   - 1-NN dev error is **26.9**%.

```
[k]     [Train Error]   [Dev Error]     [Train Positive Rate]   [Dev Positive Rate]     [Time (ms)]
1       0.015           0.269           0.251                   0.273                   493
3       0.130           0.240           0.249                   0.250                   481
5       0.156           0.234           0.232                   0.256                   420
7       0.168           0.233           0.228                   0.239                   434
9       0.183           0.222           0.221                   0.222                   432
11      0.188           0.219           0.219                   0.221                   462
13      0.190           0.217           0.219                   0.209                   437
15      0.192           0.215           0.214                   0.209                   467
17      0.195           0.221           0.209                   0.201                   515
19      0.197           0.214           0.208                   0.196                   477
21      0.198           0.217           0.204                   0.197                   483
23      0.203           0.223           0.198                   0.197                   525
25      0.201           0.221           0.194                   0.191                   522
27      0.208           0.220           0.192                   0.192                   475
29      0.215           0.213           0.187                   0.181                   506
31      0.209           0.214           0.183                   0.188                   508
33      0.214           0.207           0.174                   0.175                   517
35      0.214           0.213           0.173                   0.171                   537
37      0.214           0.217           0.173                   0.175                   513
39      0.217           0.218           0.167                   0.184                   563
41      0.217           0.218           0.160                   0.172                   581
43      0.216           0.223           0.154                   0.177                   559
45      0.219           0.223           0.152                   0.171                   540
47      0.219           0.227           0.153                   0.165                   551
49      0.221           0.220           0.146                   0.158                   525
51      0.223           0.221           0.142                   0.157                   544
53      0.226           0.224           0.140                   0.140                   536
55      0.228           0.223           0.136                   0.145                   550
57      0.228           0.218           0.135                   0.142                   555
59      0.228           0.219           0.130                   0.139                   562
61      0.230           0.224           0.132                   0.138                   560
63      0.228           0.227           0.129                   0.137                   555
65      0.229           0.226           0.129                   0.140                   583
67      0.231           0.222           0.128                   0.136                   588
69      0.231           0.222           0.128                   0.130                   657
71      0.230           0.229           0.125                   0.121                   639
73      0.229           0.221           0.124                   0.121                   659
75      0.231           0.225           0.121                   0.119                   705
77      0.230           0.220           0.121                   0.112                   684
79      0.233           0.223           0.118                   0.117                   688
81      0.232           0.219           0.118                   0.115                   684
83      0.230           0.224           0.117                   0.110
85      0.234           0.219           0.115                   0.111                   716
87      0.236           0.222           0.115                   0.110                   782
89      0.235           0.226           0.111                   0.108                   682
91      0.233           0.225           0.109                   0.101                   743
93      0.236           0.228           0.111                   0.104                   697
95      0.233           0.224           0.111                   0.110                   725
97      0.233           0.225           0.113                   0.109                   686
99      0.231           0.231           0.114                   0.109                   685
5000    0.250           0.236           0.000                   0.000                   4588
```





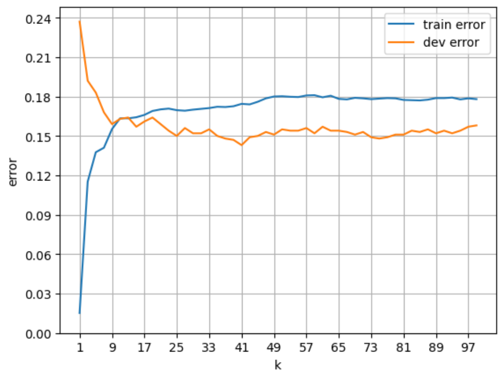


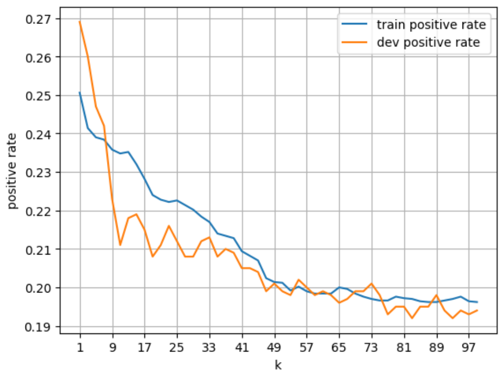
2. Yes, there is performance improvements when compared with the results of Naive Binarization experiment. This model is the best one till now. This is because, now every feature is in the range of 0 to 2. Consequently, each feature will weigh in equally for distance computation which improves the model's performance.

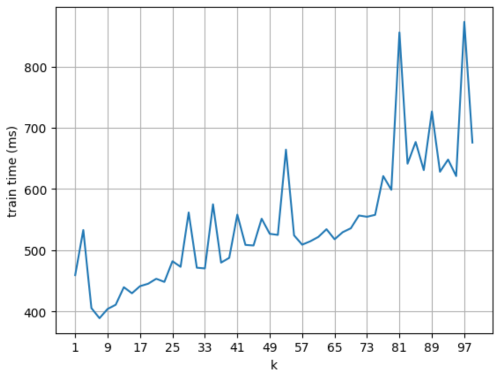
   These are the results for Smart Binarization + Numerical Features Scaling (Tested on M1 Pro processor):

   - The best dev error is **14.3**% at $k = 41$
   - 1-NN dev error is **23.7**%.

```
[k]     [Train Error]   [Dev Error]     [Train Positive Rate]   [Dev Positive Rate]     [Time (ms)]
1       0.015           0.237           0.251                   0.269                   459
3       0.115           0.192           0.241                   0.260                   533
5       0.138           0.183           0.239                   0.247                   405
7       0.141           0.168           0.238                   0.242                   389
9       0.155           0.159           0.236                   0.223                   404
11      0.163           0.163           0.235                   0.211                   411
13      0.163           0.164           0.235                   0.218                   440
15      0.164           0.157           0.232                   0.219                   430
17      0.166           0.161           0.228                   0.215                   441
19      0.169           0.164           0.224                   0.208                   445
21      0.170           0.159           0.223                   0.211                   453
23      0.171           0.154           0.222                   0.216                   448
25      0.170           0.150           0.223                   0.212                   482
27      0.169           0.156           0.221                   0.208                   473
29      0.170           0.152           0.220                   0.208                   562
31      0.171           0.152           0.218                   0.212                   471
33      0.171           0.155           0.217                   0.213                   470
35      0.172           0.150           0.214                   0.208                   575
37      0.172           0.148           0.213                   0.210                   480
39      0.173           0.147           0.213                   0.209                   487
41      0.174           0.143           0.209                   0.205                   558
43      0.174           0.149           0.208                   0.205                   509
45      0.176           0.150           0.207                   0.204                   508
47      0.179           0.153           0.202                   0.199                   551
49      0.180           0.151           0.201                   0.201                   527
51      0.180           0.155           0.201                   0.199                   525
53      0.180           0.154           0.199                   0.198                   664
55      0.180           0.154           0.200                   0.202                   524
57      0.181           0.156           0.199                   0.200                   509
59      0.181           0.152           0.198                   0.198                   515
61      0.179           0.157           0.198                   0.199                   522
63      0.181           0.154           0.198                   0.198                   534
65      0.178           0.154           0.200                   0.196                   518
67      0.178           0.153           0.200                   0.197                   529
69      0.179           0.151           0.198                   0.199                   536
71      0.179           0.153           0.198                   0.199                   557
73      0.178           0.149           0.197                   0.201                   555
75      0.178           0.148           0.197                   0.198                   558
77      0.179           0.149           0.197                   0.193                   621
79      0.179           0.151           0.198                   0.195                   599
81      0.177           0.151           0.197                   0.195                   856
83      0.177           0.154           0.197                   0.192                   641
85      0.177           0.153           0.196                   0.195                   677
87      0.178           0.155           0.196                   0.195                   631
89      0.179           0.152           0.196                   0.198                   727
91      0.179           0.154           0.197                   0.194                   628
93      0.179           0.152           0.197                   0.192                   648
95      0.178           0.154           0.198                   0.194                   621
97      0.179           0.157           0.196                   0.193                   873
99      0.178           0.158           0.196                   0.194                   676
5000    0.250           0.236           0.000                   0.000                   3780
```







4 Implement your own k-Nearest Neighbor Classifiers

1. These are the distances between the query person (first in the dev data) and its top 3 nearest neighbours (in the train data):
   - **Sklearn** (index, distance): (4872, 0.334), (4787, 1.415), (2591, 1.417)
   - **My implementation** (index, distance): (4872, 0.33440), (4787, 1.41528), (2591, 1.41674)
2. **Implement your own k-NN classifier** (with the default Euclidean distance)

   (a) Yes, we need to store the training data (feature vectors and labels) in our k-NN class object. For Sklearn's k-NN this is done by calling the fit() method.

   (b) The time complexity of k-NN to test one example involves these steps:

      i. Computing distances for all training examples: $\mathbf{O(d \times |D|)}$

      ii. Selecting k smallest distances using **np.argpartition**: $\mathbf{O(|D|)}$

      iii. Majority vote based on k nearest neighbors: $\mathbf{O(k)}$

   Time Complexity $= \mathbf{O(d \times |D| \ + \ |D| \ + \ k)}$

   (c) Sorting $|\mathbf{D}|$ distances will take $\mathbf{O(|D| \times \log|D|)}$ time. But, there is a faster way, which is to use **np.argpartition**. This function uses the Introselect algorithm to find the **k** smallest distances. Introselect is similar to Quickselect with $\mathbf{O(|D|)}$ time complexity in the average case.

   (d) I used the following tricks:

      i. **Broadcasting (matrix - vector):** To quickly subtract the test vector from my training data matrix. For example, `self.points - new_point`

      ii. **np.linalg.norm(..., axis=1)**: For calculating Euclidean distances. For example, `distances = np.linalg.norm(self.points - new_point, axis=1)`

      iii. **np.argpartition()**: For selecting k smallest distances in $\mathbf{O(|D|)}$ time. For example, `k_min_indices = np.argpartition(distances, self.k - 1)[:self.k]`

   (e) It took about **8.5 seconds** to print the training and dev errors for $k = 99$ on ENGR servers.
3. The average dev error ($1 \leq k \leq 99$) is almost the same for Euclidean and Manhattan distance implementations (**15.7**% vs **15.8**%). I got best dev error of **14.1**% (at $k = 41$) using the Manhattan distance. I got the best dev error of **14.4**% (at $k = 41$) using the Euclidean distance. Therefore, using Manhattan distance is a bit better.

5 Deployment

(a) I tested my own k-NN classifier (using Manhattan distance) with $1 \leq k \leq 1001$ on the dev dataset. The best dev error is $\mathbf{14.1}\%$ (at $k = 41$).

(b) The best dev error is $\mathbf{14.1}\%$ (at $k = 41$) on the dev set. The positive ratio on the dev set at $k = 41$ is $\mathbf{20.5}\%$.

(c) The positive ratio on the test set is $\mathbf{20.9}\%$ using my own k-NN classifier (using Manhattan distance) at $k = 41$.

6 Observations

1. These are some of the drawbacks:

   - k-NN treats all features equally. In the real world data, some features may be more influential than the others. To find more influential features, one can analyze the correlation between each feature and the target variable. Features that have a strong correlation (either positive or negative) might be more important.
   - k-NN suffers from high computational costs. That is, for each prediction, distances must be computed between the test point and every point in the training dataset, making predictions slow for large datasets. Also, k-NN requires storing the entire training dataset, which can be memory-intensive for large datasets.

2. Yes, the best performing k-NN model tends to amplify existing biases in the training data. k-NN algorithm rely directly on the training data points to make predictions. Our training data clearly has an inherent bias that lean towards having more fraction of males making greater than $50k$ USD per year when compared to the female fraction ($\mathbf{28.7}\%$ vs $\mathbf{12.7}\%$). If the training data has inherent biases, these biases can become more pronounced in the model's predictions.

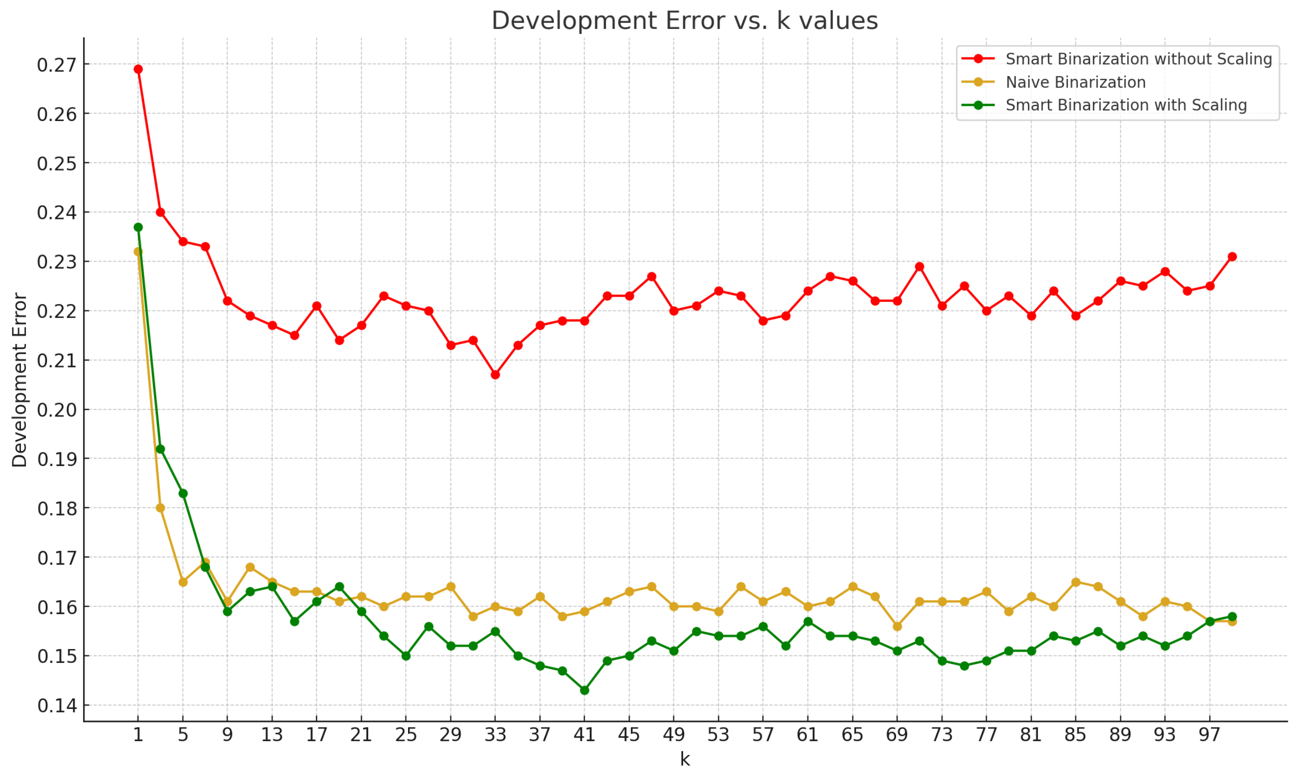
   This is more due to overfitting than underfitting. Overfitting occurs when a model captures not just the underlying patterns in the data, but also the noise and biases. A smaller k tends to overfit because it adapts to every bias in the training data, leading to a zigzagged decision boundary. With larger k we get more generalization, leading to a smoother decision boundary. However, even with large values of k, if there's a significant bias in the data, the decision boundary may still lean towards that bias.

   This amplification of biases can have conspicuous social implications, especially when machine learning models are used in critical areas like:

   - **Hiring:** If past hiring data is biased towards a particular gender or race, ML models could potentially learn that bias, leading to unfair hiring process.
   - **Financial Services:** Loan or credit approval models could potentially unreasonably favor or disfavor individuals based on gender, race, or socio-economic status.

7 Extra Credit Question

Here's the visualization:



8 Debriefing

1. I spent about 10 hours on this homework.
2. I would rate it easy.
3. I worked on it alone.
4. I understand the material 100%.
5. The module videos are very good. I couldn't find any better tutorial of Numpy and Linux commands than these ones.