CS 519: Applied Machine Learning **(Due: April 20, 2020)**

Submission Homework 1

Instructor: Prof. Liang Huang *Name:* Rashmi Jadhav*, ONID:* 934-069-574

1. Data (Pre-)Processing (Feature Map)

1. The positive % of the training data is **25.02%** with 1251 salaries >50K and 3749 salaries ≤50K out of a total of 5000 training examples:
   `$ cat income.train.txt.5k | grep '>50K' | wc -l`
   The positive % of the dev set is **23.6%** with 236 salaries >50K and 764 salaries ≤50K out of a total of 1000 dev examples:
   `$ cat income.dev.txt | grep '>50K' | wc -l`
   Given the average per capita income of ∼27K in the US as per 1994 and even for today, our training data as well as dev set pretty much mimic this behavior as they are only ∼24-25% positive. This means that ∼75-76% people earn <50K which is in congruence with the average per capita income (also less than 50K).

2. The youngest and oldest ages in the training set are **17** and **90** respectively.
   `$ cat income.train.txt.5k | cut -f 8 -d ',' | sort -nk1 | head -1`
   The least amount of hours per week is **1** and the most amount of hours per week are **99** in the training set.
   `$ cat income.train.txt.5k | cut -f 8 -d ',' | sort -nk1 | tail -1`

3. Consider categorical field `sector` having 7 distinct values {`Federal-gov, Local-gov, Private, Self-emp-inc, Self-emp-not-inc, State-gov, Without-pay`}. If we create mappings from feature values to vectors as:
   ```
   Federal-gov = 1
   Local-gov = 2
   Private = 3
   Self-emp-inc = 4
   Self-emp-not-inc = 5
   State-gov = 6
   Without-pay = 7
   ```
   This mapping turns an unordered set of `sectors` into an ordered set {1, 2, 3, 4, 5, 6, 7}. By doing this, we are saying that `Federal-gov` and `Local-gov` (distance of 1) are more similar than `Federal-gov` and `Without-pay` (distance of 6). We don't mean to say this nor do we want to introduce such a bias. Hence we should binarize all categorical fields to one-hot representation to consider all possible values that the field can have without introducing any distance bias.

4. If we binarize all the 7 categorical fields {*sector, education, marital-status, occupation, race, gender, country-of-origin*}, the features will be binary which will result in strictly one positive feature per field. This means that one training example will be converted to a vector with seven 1's and rest all 0's. The maximum possible Euclidean and Manhattan distances will occur when the two training examples are like the following:
   ```
   [1 0 1 0 1 0 1 0 1 0 1 0 1 0]
   [0 1 0 1 0 1 0 1 0 1 0 1 0 1]
   ```
   Manhattan distance: 14
   Euclidean distance: $\sqrt{14}$

5. If we binarize *age* and *hours-per-week* like the categorical fields, the features will become binary like `age=17, age=18, ..., age=90` with values 0 or 1. When we calculate the Manhattan distance between two examples with ages 17 and 90, it will be a minimum of 0 and a maximum of 2 with binarization. However, mathematically, distance between 17 and 90 is 73. Thus we have converted the ordered set of ages to an unordered set with any pair of ages being equidistant which is not what we want to achieve.
   Also, if we do not binarize numeric fields like *age* and *hours-per-week*, these fields will overpower the other binarized fields and carry more weight which will influence the distance metric. The categorical binarized

fields are bounded by a maximum distance of 2. We should thus try to normalize numeric fields such that they are bounded by 2. Thus, we should divide all the field values in a numeric field by half the maximum value of the corresponding field leading to the field values ranging from 0 - 2. Therefore, we may divide `age` by **45** and `hours-per-week` by **49.5**, however, we approximate these numbers to **50** considering the real world scenario. All the fields have now been treated equally through normalization and binarization.

6. After applying normalization on numeric fields and binarization on categorical fields, we get a total of **92** features. Following are the number of features allocated per field:

```
Numeric field:  # age = 1
Categorical field:  # sector = 7
Categorical field:  # education = 16
Categorical field:  # marital-status = 7
Categorical field:  # occupation = 14
Categorical field:  # race = 5
Categorical field:  # gender = 2
Numeric field:  # hours-per-week = 1
Categorical field:  # country-of-origin = 39
```

7. After applying binarization on all the fields, we get a total of **230** features. Following are the number of features allocated per field: (*shouldn't binarize all fields)

```
# age = 67
# sector = 7
# education = 16
# marital-status = 7
# occupation = 14
# race = 5
# gender = 2
# hours-per-week = 73
# country-of-origin = 39
```

2: Calculating Manhattan and Euclidean Distances

```
def find_topk(k, manh_or_eucl, vector_train, vector_dev):
    diff = vector_train - vector_dev[:, np.newaxis]
    if manh_or_eucl == "eucl":
        dist = np.linalg.norm(diff, axis = 2)
    elif manh_or_eucl == "manh":
        dist = np.sum(np.abs(diff), axis = 2)
    train_rows = vector_train.shape[0]
    kVals = range(k) if k < train_rows else range(train_rows)
    top_k_indices = np.argpartition(dist, kVals, axis = 1)[:, :k]
    top_k_dist = dist[np.arange(dist.shape[0])[:, None], top_k_indices]
    return top_k_indices, top_k_dist
```

1. The five people closest to the last person in dev set with their corresponding **Euclidean distances** are as follows:

```
top_k_indices, top_k_dist = find_topk(5, "eucl", vector_train, np.array([vector_dev[-1]]))
print(top_k_indices)
print(top_k_dist)
for row in top_k_indices:
    for index in row:
        !sed -n {index + 1}p 'hw1-data/income.train.txt.5k'
```

```
[[1010 1713 3769 2003 2450]]
[[0.06       0.16       0.26       0.28284271 0.34      ]]
55, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
66, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
58, Private, HS-grad, Widowed, Adm-clerical, White, Female, 27, United-States, <=50K
68, Private, HS-grad, Widowed, Adm-clerical, White, Female, 30, United-States, <=50K
75, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
```

2. The five people closest to the last person in dev set with their corresponding **Manhattan distances** are as follows:

```
top_k_indices, top_k_dist = find_topk(5, "manh", vector_train, np.array([vector_dev[-1]]))
print(top_k_indices)
print(top_k_dist)
for row in top_k_indices:
    for index in row:
        !sed -n {index + 1}p 'hw1-data/income.train.txt.5k'
```

```
[[1010 1713 3769 2450 2003]]
[[0.06 0.16 0.26 0.34 0.4 ]]
55, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
66, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
58, Private, HS-grad, Widowed, Adm-clerical, White, Female, 27, United-States, <=50K
75, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
68, Private, HS-grad, Widowed, Adm-clerical, White, Female, 30, United-States, <=50K
```

3. The target values for top-5 nearest neighbors of the last vector in dev set are equivalent to the actual target value of the last vector in dev set. Thus our prediction of it being $\leq$50K is correct.

```
top_k_indices, top_k_dist = find_topk(5, "manh", vector_train, np.array([vector_dev[-1]]))

!echo "Target value for last vector in dev:"
!sed -n 1000p 'hw1-data/income.dev.txt' | cut -f 10 -d ","

!echo "Target values for 5 nearest neighbors:"
for row in top_k_indices:
    for index in row:
        !sed -n {index + 1}p 'hw1-data/income.train.txt.5k' | cut -f 10 -d ","
```

```
Target value for last vector in dev:
 <=50K
Target values for 5 nearest neighbors:
 <=50K
 <=50K
 <=50K
 <=50K
 <=50K
```

4. 9NN for last dev row:
   Euclidean distances

```
top_k_indices, top_k_dist = find_topk(9, "eucl", vector_train, np.array([vector_dev[-1]]))
print(top_k_indices)
print(top_k_dist)
for row in top_k_indices:
    for index in row:
        !sed -n {index + 1}p 'hw1-data/income.train.txt.5k'
```

```
[[1010 1713 3769 2003 2450 3698 3680  681 2731]]
[[0.06       0.16       0.26       0.28284271 0.34       0.40049969
  0.43863424 0.56       1.41421356]]
55, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
66, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
58, Private, HS-grad, Widowed, Adm-clerical, White, Female, 27, United-States, <=50K
68, Private, HS-grad, Widowed, Adm-clerical, White, Female, 30, United-States, <=50K
75, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
59, Private, HS-grad, Widowed, Adm-clerical, White, Female, 60, United-States, <=50K
49, Private, HS-grad, Widowed, Adm-clerical, White, Female, 20, United-States, <=50K
30, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
58, Private, HS-grad, Divorced, Adm-clerical, White, Female, 40, United-States, <=50K
```

   Manhattan distances

```
top_k_indices, top_k_dist = find_topk(9, "manh", vector_train, np.array([vector_dev[-1]]))
print(top_k_indices)
print(top_k_dist)
for row in top_k_indices:
    for index in row:
        !sed -n {index + 1}p 'hw1-data/income.train.txt.5k'
```

```
[[1010 1713 3769 2003 2450 3698 3680  681 2731]]
[[0.06       0.16       0.26       0.28284271 0.34       0.40049969
  0.43863424 0.56       1.41421356]]
55, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
66, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
58, Private, HS-grad, Widowed, Adm-clerical, White, Female, 27, United-States, <=50K
68, Private, HS-grad, Widowed, Adm-clerical, White, Female, 30, United-States, <=50K
75, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
59, Private, HS-grad, Widowed, Adm-clerical, White, Female, 60, United-States, <=50K
49, Private, HS-grad, Widowed, Adm-clerical, White, Female, 20, United-States, <=50K
30, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
58, Private, HS-grad, Divorced, Adm-clerical, White, Female, 40, United-States, <=50K
```

   The target values for top-9 nearest neighbors of the last vector in dev set are equivalent to the actual target value of the last vector in dev set. Thus our prediction of it being $\leq$50K is correct.

```
top_k_indices, top_k_dist = find_topk(9, "eucl", vector_train, np.array([vector_dev[-1]]))
!echo "Target value for last vector in dev:"
!sed -n 1000p 'hw1-data/income.dev.txt' | cut -f 10 -d ","

!echo "Target values for 9 nearest neighbors:"
for row in top_k_indices:
    for index in row:
        !sed -n {index + 1}p 'hw1-data/income.train.txt.5k' | cut -f 10 -d ","
```

```
Target value for last vector in dev:
 <=50K
Target values for 9 nearest neighbors:
 <=50K
 <=50K
 <=50K
 <=50K
 <=50K
 <=50K
 <=50K
 <=50K
 <=50K
```

3: *k*-Nearest Neighbor Classification

1. KNN classifier implementation:

```
def knn(k, manh_or_eucl, vector_train, vector_dev, train_targets):
    top_k_indices, top_k_dist = find_topk(k, manh_or_eucl, vector_train, vector_dev)
    predicted_list = []
    for row in top_k_indices:
        nearby_targets = []
        for index in row:
            nearby_targets.append(train_targets[index])
        predicted_list.append(max(set(nearby_targets), key=nearby_targets.count))
    return predicted_list
```

i. After finishing the feature map and after converting training dataset into binarized-normalized vector there's no work in training, provided we have extracted targets to use for predictions during KNN implementation.

ii. The time complexity of k-NN to test one example of dimensionality d and D sized training set is O(d * D + k * D). However, with optimization in sort using quicksort + slicing, the complexity turns out to be **O(d*D)**

iii. The distances need not be sorted first in order to pick top k closest neighbors. We can use quick sort technique to keep only top k values sorted and not care about the rest but rather slice them off.

2. *k* in k-NN is usually chosen to be odd when the number of classes is even. If we were to choose an even k, the decision would be difficult when exact half neighbors are from one class and the rest half of the other class.

3. k-NN execution on dev set:

```
    ks = [1, 3, 5, 7, 9, 99, 999, 9999]
    for k in ks:
        predicted_list = knn(k, "eucl", vector_train, vector_dev, train_targets)
        dev_positivity = 0
        dev_errors = 0
        for idx, val in enumerate(predicted_list):
            if val == ">50K":
                dev_positivity += 1
            if val != dev_targets[idx]:
                dev_errors += 1
        dev_positivity = dev_positivity / dev_rows * 100
        dev_errors = dev_errors / dev_rows * 100

        print("k = %d\t dev error %.2f%% (+:%.2f%%)" % (k, dev_errors, dev_positivity))
```

```
k = 1    dev error 23.30% (+:26.90%)
k = 3    dev error 19.60% (+:26.20%)
k = 5    dev error 17.80% (+:24.60%)
k = 7    dev error 16.40% (+:24.00%)
k = 9    dev error 15.90% (+:21.90%)
k = 99   dev error 15.70% (+:19.10%)
k = 999  dev error 18.10% (+:11.30%)
k = 9999        dev error 23.60% (+:0.00%)
```

The best error rate on dev set occurs at ***k*=99** (15.70%)

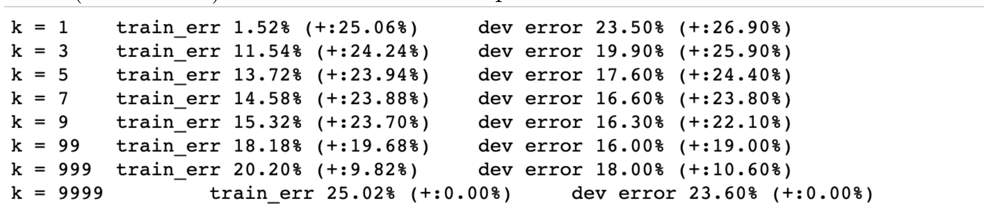
4. k-NN (Euclidean) on dev and train examples:

```
k = 1    train_err 1.52% (+:25.06%)     dev error 23.30% (+:26.90%)
k = 3    train_err 11.48% (+:24.18%)    dev error 19.60% (+:26.20%)
k = 5    train_err 13.74% (+:24.28%)    dev error 17.80% (+:24.60%)
k = 7    train_err 14.52% (+:23.82%)    dev error 16.40% (+:24.00%)
k = 9    train_err 15.50% (+:24.00%)    dev error 15.90% (+:21.90%)
k = 99   train_err 17.88% (+:19.50%)    dev error 15.70% (+:19.10%)
k = 999  train_err 20.30% (+:10.44%)    dev error 18.10% (+:11.30%)
k = 9999        train_err 25.02% (+:0.00%)     dev error 23.60% (+:0.00%)
```

At k = 1, training error is 1.52%. The reason behind it not being 0% is the training data. There are certain duplicate training examples for which the target values for salary are different. These examples contribute to the 1.52% error. In general, if the training error is 0%, it means that your model has overfit the data which is not good.

5. At k = 1, train error is the least and the model exhibits overfitting with training error of 1.52%. This means that the classifier considers only one nearest neighbor leading to highest dev error at k = 1. Dev error starts to decrease up to 14.4% at k = 41 and after that, it starts to increase, this is the point where the model has overfit the training data. While k = 1 considers only one neighbor to predict the target, k = ∞ considers all the other data point leading to not giving importance to closeness but to the whole dataset.

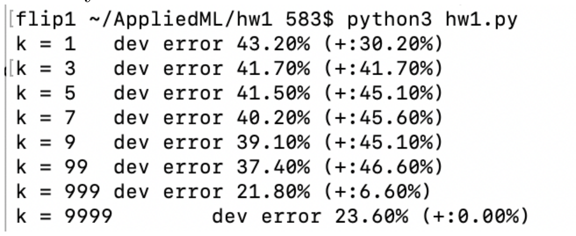
6. k-NN (Manhattan) on dev and train examples:



```
k = 1    train_err 1.52% (+:25.06%)     dev error 23.50% (+:26.90%)
k = 3    train_err 11.54% (+:24.24%)    dev error 19.90% (+:25.90%)
k = 5    train_err 13.72% (+:23.94%)    dev error 17.60% (+:24.40%)
k = 7    train_err 14.58% (+:23.88%)    dev error 16.60% (+:23.80%)
k = 9    train_err 15.32% (+:23.70%)    dev error 16.30% (+:22.10%)
k = 99   train_err 18.18% (+:19.68%)    dev error 16.00% (+:19.00%)
k = 999  train_err 20.20% (+:9.82%)     dev error 18.00% (+:10.60%)
k = 9999        train_err 25.02% (+:0.00%)     dev error 23.60% (+:0.00%)
```

The dev error rate for Manhattan method is slightly higher than Euclidean method and at the same time, dev positivity for Manhattan is slightly less than Euclidean. This means that Manhattan does a bit worse than Euclidean. However, L1 norm is preferred over L2 norm for the case of high dimensional data since L2 norm tends to make the distances too uniform.

7. With Euclidean k-NN having all-binarized features, the vectors became as huge as the shapes (5000, 233) & (1000, 233). Tthe algorithm didn't run properly and went into memory error. However, I could note the dev error and dev positivity which was observed to be much more than normalized and binarized k-NN. This tells us that binarizing numeric fields age and hours-per-week led to introducing errors. By binarizing these fields, we have eliminated the numeric distance which exists between ages and this is not what we intend to do. Thus, we shouldn't ever binarize numeric fields. Also, with huge vectors, the Euclidean k-NN went into memory error which shows a drawback of Euclidean k-NN in cases of datasets with a large number of features.



```
[flip1 ~/AppliedML/hw1 583$ python3 hw1.py
k = 1   dev error 43.20% (+:30.20%)
[k = 3   dev error 41.70% (+:41.70%)
k = 5   dev error 41.50% (+:45.10%)
k = 7   dev error 40.20% (+:45.60%)
k = 9   dev error 39.10% (+:45.10%)
k = 99  dev error 37.40% (+:46.60%)
k = 999 dev error 21.80% (+:6.60%)
k = 9999        dev error 23.60% (+:0.00%)
```

4: Deployment

1. i. With **Euclidean** distance K-NN, at **k = 41**, the best dev results were achieved.

   ii. k = 41 dev error **14.40% (+: 20.00%)**

   iii. The positive ratio on test is **20.4%**

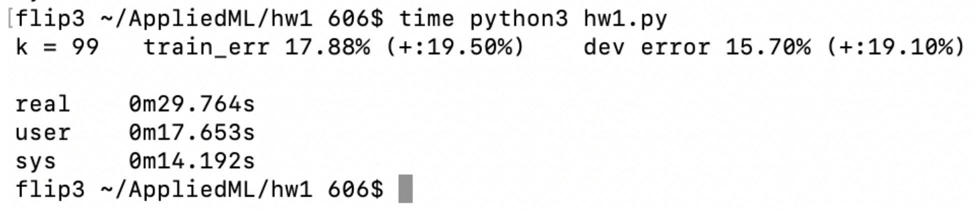
5: Observations

1. k-NN simply uses the training data for classification and it doesn't really learn much of anything from the data. This results in the algorithm not being able to generalize well(generalization problem).
   If there are large number of training examples, computing the distances and finding the top-k closest might become time consuming.
   All the features are given equal importance which may not be the case always.
   Different values of k can give different classification and thus as dataset evolves, k should be timely updated

2. Bias is essentially the difference between the average prediction of our model and the truth value which we are trying to predict. If we consider the best performing models, they tend to exhibit high bias over the training data. If we consider education=Doctorate to be a highly positive feature, chances are the predicted values for the same feature would be even more on dev set using our model. Such models with high bias pay very little attention to the training data and oversimplify the model. This is the problem of underfitting wherein the model is unable to capture the underlying data pattern. This exaggeration of the existing bias in the data is indeed a societal problem.
   As per one of the faculty talks I attended(Causal Inference for Responsible Data Science), the model predicted overall less number of females in AWS job application portal whereas if you query per job role, turns out that females were more in numbers than males. Such errors in ML lead to wrong interpretations in the society and thus it is very important to take care of your models in those ways.

3. Since computation in Python becomes time consuming, we should use Matlab style computation so that the algorithm takes lesser time to execute. Broadcasting for doing multi-dimensional vector subtraction was used. `np.linalg.norm()` was used to calculate L2 norm and `np.sum(np.abs())` for L1 norm. `np.argpartition()` was used to sort only top k rows. Slicing was used for not having to deal with large vectors

4. My k-NN model takes **17.653s** of user time to print the training and dev errors for k = 99 on ENGR servers.



```
[flip3 ~/AppliedML/hw1 606$ time python3 hw1.py
 k = 99   train_err 17.88% (+:19.50%)    dev error 15.70% (+:19.10%)

 real    0m29.764s
 user    0m17.653s
 sys     0m14.192s
 flip3 ~/AppliedML/hw1 606$
```

5. The Voronoi diagram divides datasets into different cells, each cell contains multiple examples with same classification value. For our example, we may have two cells with classification values >50K and ≤50K with all such examples lying in their corresponding areas. k-NN essentially helps in classifying where a particular example may belong.

Debriefing

1. I spent around 25 hours on this assignment.
2. The assignment is moderate but lengthy.
3. I worked mostly alone whilst also reading people's Q&A on Slack.
4. I think I understand about 90% of the material covered in the assignment. Understanding multi-dimensional vector subtraction (broadcasting) was a bit tricky.
5. This was quite a thorough assignment that it didn't just make me implement k-NN but also helped me analyze many things associated with it. It was a really bad idea to start late. I should have been interactive on Slack to stay on similar page as that of other students. Nonetheless, I will take care of the material henceforth in a better way.