CS519 - Applied Machine Learning (e-campus)
Instructor: Prof. Liang Huang
HW #1 -Monday April 19 @ 11:59pm on Canvas

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

   1) Take a look at the data. A training example looks like this:
   37, Private, Bachelors, Separated, Other-service, White, Male, 70, England, <=50K
   which includes the following 9 input fields plus one output field (y):
   age, sector, education, marital-status, occupation, race, gender, hours-per-week, country-of-origin, target
   Q: What are the positive % of training data? What about the dev set? Does it make sense given your knowledge of the average per capita income in the US?

   > Training data:
   > >50K: 1251; positive %: 25.02%
   > Dev set:
   > >50K: 236; positive %: 23.6%
   >
   > ```
   > cat income.train.txt.5k| cut -f 10 -d ','| sort -r | uniq -c | awk '{print $2, $1}'
   > (or)
   > awk '$10 == ">50K"' income.train.txt.5k| wc -l
   > ```
   >
   > These rates make sense given that the 1994 per-capita income in the US was only $27,350 as the data was from 1994 US Census (in 2017 it becomes $60,200).

   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 wor*k*? Hint:
   cat income.train.txt.5k | sort -nk1 | head -1

   > ages from 17 to 90
   > hours-per-week from 1 to 99

   3) There are two types of fields, numerical (age and hours-per-week), and categorical (everything else). The default preprocessing method is to binarize all categorical fields, e.g., race becomes many binary features such as race=White, race=Asian-Pac-Islander, etc. These resulting features are all binary, meaning their values can only be 0 or 1, and for each example, in each field, there is one and only one positive feature (this is the so-called ?one-hot? representation, widely used in ML and NLP).
   Q: Why do we need to binarize all categorical fields?

   > a) binarizing all categorical fields to a higher dimension with values 0 or 1, to calculate the distance
   > b) feature map to a higher dimension, easy to linearly separate
   > c) normalization is done in the binarization to some extent

   4) Q: If we do not count age and hours, what?s maximum possible Euclidean and Manhattan distances between two training examples? Explain.

It happens when there is no same categorical features between two training examples, and all categorical features in the two examples are not "unseen" in the feature map.

There are 7 categorical fields for each sample. For each categorical field, there are two "one-hot" at most, one from the training, one from the dev example.

Euclidean distance:

$\sqrt{2 * 7}$

Manhattan distances:

2*7

5) Why we do not want to binarize the two numerical fields, age and hours? What if we did? How should we define the distances on these two dimensions so that each field has equal weight? (In other words, the distance induced by each field should be bounded by 2).
Hint: numerical fields / 50, e.g., "age" / 50 so that the max distance on "age" is also 2.

The benefits if we do not binarize the two numerical fields are:

a) using difference between two ages/hours as the distance brings in more useful information. For example, if age_1=50, age_2=21, and age_3=20, then age_2 is closer to age_3 than age_1 to age_3.

b) to avoid feature mapping to a very high dimension. We have a 92-dimension feature map if we do not binarize the two numerical fields. (vs. 230-dim if we do)

If we did,
a) the dimensionality of the feature map becomes larger.
b) we lost the ability to accurately distinguish distances between two different ages/hours.

We should bound the two fields by 2. Since the max difference on each categorical field is 2 (please refer to question 1.4). We could use 50 as the divisor to normalize, alternatively, we could use max-min normalization instead.

6) Q: How many features do you have in total (i.e., the dimensionality)? Hint: should be around 90. How many features do you allocate for each of the 9 fields?
Hint: for i in 'seq 1 9'; do cat income.train.txt.5k | cut -f $i -d ',' | sort | uniq | wc -l; done

92
('sector', 7), ('education', 16), ('marital-status?, 7), ('occupation?, 14), ('race?,5), ('gender?, 2), ('country-of-origin?, 39)
('age',1), ('hours-per-week',1)

7) Q: How many features would you have in total if you binarize all fields?

230 ('age',67), ('hours-per-week',73)

2. Calculating Manhattan and Euclidean Distances

Hint: you can use the Matlab style "broadcasting" notations in numpy (such as matrix - vector) to calculate many distances in one shot. For example, if `A` is an $n \times m$ matrix (n rows, m columns, where n is the number of people and

m is the number of features), and `p` is an m-dimensional vector (1 row, m columns) representing the query person, then `A - p` returns the difference vectors from each person in `A` to the query person `p`, from which you can compute the distances:

```
>>> A = np.array([[1,2], [2,3], [4,5]])
>>> p = np.array([3,2])
>>> A - p
array([[-2,  0],
       [-1,  1],
       [ 1,  3]])
>>> np.linalg.norm(A-p, axis=1)
array([2.        , 1.41421356, 3.16227766])
```

This is Euclidean distance (what does `axis=1` mean?). You need to figure out Manhattan distance yourself.

> Set ord = 1 when calling np.linalg.norm() to calculate distance because Manhattan distance is L1 norm.
>
> ```
> >>> A = np.array([[1,2], [2,3], [4,5]])
> >>> p = np.array([3,2])
> >>> A - p
> array([[-2,  0],
>        [-1,  1],
>        [ 1,  3]])
> >>> np.linalg.norm(A-p, ord = 1, axis=1)
> array([2., 2., 4.])
> ```

To make sure your distance calculations are correct, we provide the following example calculations using the first person in the dev set:

```
$ head -1 income.dev.txt
45, Federal-gov, Bachelors, Married-civ-spouse, Adm-clerical, White, Male, 45, United-States, <=50K
```

The top-3 examples in the training set that are closest to the above person, according to the Manhattan distance, should be the following rows (note that the command `sed -n XXp` prints the XXth line of a file):

```
$ sed -n 4873p income.train.txt.5k
33, Federal-gov, Bachelors, Married-civ-spouse, Adm-clerical, White, Male, 42, United-States, >50K
$ sed -n 4788p income.train.txt.5k
47, Federal-gov, Bachelors, Married-civ-spouse, Adm-clerical, White, Male, 45, Germany, >50K
$ sed -n 2592p income.train.txt.5k
48, Federal-gov, Bachelors, Married-civ-spouse, Prof-specialty, White, Male, 44, United-States, >50K
```

Notice that the first of these three persons matches all categorical fields with the dev person, only differing slightly in the two numerical fields, and the second and third persons match all but one categorical fields. The Manhattan distances of these three people to the dev person are:

```
(45-33) / 50. + (45-42) / 50. = 0.3
(47-45) / 50. + (45-45) / 50. + 1 + 1 = 2.04
(48-45) / 50. + (45-44) / 50. + 1 + 1 = 2.08
```

Coincidentally, these three people are also the top-3 closest according to the Euclidean distances, with the distances being

```
sqrt( ((45-33) / 50.) ** 2 + ((45-42) / 50.) ** 2 ) = 0.24738633753705963
sqrt( ((47-45) / 50.) ** 2 + ((45-45) / 50.) ** 2  + 1 ** 2 + 1 ** 2 ) = 1.4147791347061915
sqrt( ((48-45) / 50.) ** 2 + ((45-44) / 50.) ** 2  + 1 ** 2 + 1 ** 2 ) = 1.4156270695349111
```

Also notice that in both cases, the 3-NN predictions are wrong, as the top-3 closest examples are all `>50K`.

Finally, remember that you don't really need to sort the distances in order to get the top-k closest examples.

Questions:

(a) Find the five (5) people closest to the last person (in Euclidean distance) in dev, and report their distances:

```
$ tail -1 income.dev.txt
58, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
```

The positions of the five closest people to the last person in the training set are 1011, 1714, 3770, 2004 and 2451 (starting from 1) with Euclidean distance 0.06, 0.16, 0.26, 0.283 and 0.34 respectively.

```
$ sed -n 1011p income.train.txt.5k
55, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
$ sed -n 1714p income.train.txt.5k
66, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
$ sed -n 3770p income.train.txt.5k
58, Private, HS-grad, Widowed, Adm-clerical, White, Female, 27, United-States, <=50K
$ sed -n 2004p income.train.txt.5k
68, Private, HS-grad, Widowed, Adm-clerical, White, Female, 30, United-States, <=50K
$ sed -n 2451p income.train.txt.5k
75, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
```

(b) Redo the above using Manhattan distance.

The positions of the five closest people to the last person in the training set are 1011, 1714, 3770, 2004 and 2451 (starting from 1) with Euclidean distance 0.06, 0.16, 0.26, 0.34 and 0.4 respectively. The five closest people are the same as the results of Euclidean distance.

(c) What are the 5-NN predictions for this person (Euclidean and Manhattan)? Are these predictions correct?

The 5-NN predicts this person negative (<=50K) with both Euclidean and Manhattan distance because all the income of the five closest people are <=50K with these two distance. And these predictions are correct since the ground truth is <=50K for this person.

(d) Redo all the above using 9-NN (i.e., find top-9 people closest to this person first).

Euclidean distance:
The positions of the nine closest people to the last person in the training set are 1011, 1714, 3770, 2004, 2451, 3699, 3681, 682 and 2732 (starting from 1) with Euclidean distance 0.06, 0.16, 0.26, 0.283, 0.34, 0.4, 0.439, 0.56 and 1.41 respectively.

```
$ sed -n 1011p income.train.txt.5k
55, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
$ sed -n 1714p income.train.txt.5k
66, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
$ sed -n 3770p income.train.txt.5k
58, Private, HS-grad, Widowed, Adm-clerical, White, Female, 27, United-States, <=50K
$ sed -n 2004p income.train.txt.5k
68, Private, HS-grad, Widowed, Adm-clerical, White, Female, 30, United-States, <=50K
$ sed -n 2451p income.train.txt.5k
75, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
$ sed -n 3699 income.train.txt.5k
59, Private, HS-grad, Widowed, Adm-clerical, White, Female, 60, United-States, <=50K
$ sed -n 3681 income.train.txt.5k
49, Private, HS-grad, Widowed, Adm-clerical, White, Female, 20, United-States, <=50K
$ sed -n 682 income.train.txt.5k
30, Private, HS-grad, Widowed, Adm-clerical, White, Female, 40, United-States, <=50K
$ sed -n 2732 income.train.txt.5k
58, Private, HS-grad, Divorced, Adm-clerical, White, Female, 40, United-States, <=50K
```

Manhattan distance:
The positions of the nine closest people to the last person in the training set are 1011, 1714, 3770, 2451, 2004, 3699, 682, 3681, 2732 (starting from 1) with Mahattan distance 0.06, 0.16, 0.26, 0.34, 0.4, 0.42, 0.56, 0.58 and 2.0 respectively. The closest people are the same as Euclidean distance. The KNN with k = 9 predicts this person negative (<=50K) with both Euclidean and Manhattan distance because all the income of the five closest people are <=50K with these two distance. And these predictions are correct since the ground truth is <=50K for this person.

3. k-Nearest Neighbor Classification

1) Implement the basic k-NN classifier (with the default Euclidean distance).
Q: Is there any work in training after finishing the feature map?
Q: What's the time complexity of k-NN to test one example (dimensionality d, size of training set $|D|$)?
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.

There is no work in training after finishing the feature map.

1) To calculate the distance between one training example and one dev sample: $O(d)$
2) To calculate $|D|$ distances for this dev example ($|D|$ training examples - one dev example): $O(|D| * d)$
3) To get the top k nearest distances: $O(|D| \log |D|)$ with sorting; O(|D|) with "quick-select" algorithm.
4) To predict (majority vote): $O(k)$, where $k \leq |D|$

With 'sort' algorithm (e.g., `np.sort`):
Time complexity: $O(|D| * d) + O(|D| \log |D|) + O(k) = O(|D|(d + \log |D|))$

With 'quick-select' algorithm (e.g., `np.partition`):
Time complexity: $O(|D| * d) + O(|D|) + O(k) = O(|D| * d)$

2) Q: Why the k in k-NN has to be an odd number?

> Tie-breaking vote.

3) Evaluate k-NN on the dev set and report the error rate and predicted positive rate for k = 1, 3, 5, 7, 9, 99, 999, 9999,
e.g., something like:

```
k=1    dev_err xx.x% (+:xx.x%)
k=3    ...
...
k=9999 ...
```

Q: what's your best error rate on dev, and where did you get it? (Hint: 1-NN dev error should be 23% and its positive % should be 27%).

```
k=1    dev_err 23.3% (+:27.1%)
k=3    dev_err 19.2% (+:25.8%)
k=5    dev_err 17.8% (+:25.0%)
k=7    dev_err 16.2% (+:24.2%)
k=9    dev_err 15.7% (+:22.3%)
k=99   dev_err 15.6% (+:19.2%)
k=999  dev_err 17.9% (+:11.1%)
k=9999 dev_err 23.6% (+:0.0%)
```

4) Now report both training and testing errors (your code needs to run a lot faster! See Question 4.3 for hints. See also week 2 videos for numpy and linear algebra tutorials, in case you?re not familiar with the ?Matlab?-style of thinking which is inherited by numpy):

```
k=1    train_err xx.x% (+:xx.x%)  dev_err xx.x% (+:xx.x%)
k=3    ...
...
k=9999 ...
```

Q: When k = 1, is training error 0%? Why or why not? Look at the training data to confirm your answer.

```
k=1    train_err 1.5% (+:25.1%) dev_err 23.3% (+:27.1%)
k=3    train_err 11.5% (+:24.0%) dev_err 19.2% (+:25.8%)
k=5    train_err 13.7% (+:24.3%) dev_err 17.8% (+:25.0%)
k=7    train_err 14.6% (+:24.0%) dev_err 16.2% (+:24.2%)
k=9    train_err 15.4% (+:24.0%) dev_err 15.7% (+:22.3%)
k=99   train_err 17.8% (+:19.5%) dev_err 15.6% (+:19.2%)
k=999  train_err 20.2% (+:10.4%) dev_err 17.9% (+:11.1%)
k=9999 train_err 25.0% (+:0.0%) dev_err 23.6% (+:0.0%)
```

When $k = 1$, the training error is not 0%, but rather 1.5%, since there are some people with the same features but different labels in the training data set. For example:

```
$ cat income.train.txt.5k | sort | cut -f 1-9 -d ',' | uniq -c | sort -nk1 | tail -1

  5 51, Private, HS-grad, Married-civ-spouse, Craft-repair, White, Male, 40, United-States

$ grep "51, Private, HS-grad, Married-civ-spouse, Craft-repair, White, Male, 40, United-States" \
  income.train.txt.5k

51, Private, HS-grad, Married-civ-spouse, Craft-repair, White, Male, 40, United-States, <=50K
51, Private, HS-grad, Married-civ-spouse, Craft-repair, White, Male, 40, United-States, <=50K
51, Private, HS-grad, Married-civ-spouse, Craft-repair, White, Male, 40, United-States, >50K
51, Private, HS-grad, Married-civ-spouse, Craft-repair, White, Male, 40, United-States, <=50K
51, Private, HS-grad, Married-civ-spouse, Craft-repair, White, Male, 40, United-States, <=50K
```

5) Q: What trends (train and dev error rates and positive ratios, and running speed) do you observe with increasing k? Do they relate to underfitting and overfitting?

   Q: What does $k = \infty$ do? Is it extreme overfitting or underfitting? What about $k = 1$?

Basically, as k increases, training error increases, but dev error first drops, and then increases, and eventually reaches the true positive ratio when $k \to \infty$.
Small k: overfitting. Large k: underfitting.
When $k = \infty$, k-NN simply returns the majority class of the whole training set, which is extreme underfitting.
When $k = 1$, it is extreme overfitting.
The running speed are positively correlated with the value of k. However, since $k \leq |D|$, it doesn't affect the running speed much.

6) Redo the evaluation using Manhattan distance. Better or worse? Any advantage of Manhattan distance?

```
k=1    train_err 1.5% (+:25.1%) dev_err 23.5% (+:26.9%)
k=3    train_err 11.5% (+:24.0%) dev_err 19.4% (+:25.4%)
k=5    train_err 13.9% (+:24.0%) dev_err 17.5% (+:24.7%)
k=7    train_err 14.7% (+:24.2%) dev_err 16.9% (+:23.9%)
k=9    train_err 15.1% (+:23.9%) dev_err 16.2% (+:22.4%)
k=99   train_err 18.1% (+:19.8%) dev_err 16.2% (+:18.8%)
k=999  train_err 20.1% (+:9.8%) dev_err 17.9% (+:10.5%)
k=9999 train_err 25.0% (+:0.0%) dev_err 23.6% (+:0.0%)
```

There is no big difference.

Reference Solution

Manhattan distance is supposed be faster than Euclidean distance, since the latter involves expensive float-point calculations such as squares and square roots. But surprisingly, at least in numpy, Euclidean is significantly faster than Manhattan, probably because ℓ_2-norm is used much more often than ℓ_1-norm so numpy has some special optimization in ℓ_2-norm. In Python, however, Manhattan is indeed a lot faster:

```
$ ipython3
In [1]: import numpy as np
In [2]: a=np.random.random(100000)

In [3]: %time np.linalg.norm(a,ord=1)
CPU times: user 1.11 ms, sys: 2.57 ms, total: 3.68 ms
Wall time: 9.18 ms
Out[3]: 49798.42595258584

In [4]: %time np.linalg.norm(a,ord=2)
CPU times: user 346 µs, sys: 1.36 ms, total: 1.71 ms
Wall time: 6.75 ms
Out[4]: 181.99014992485596

In [7]: from math import sqrt, fabs

In [11]: %time sum(fabs(x) for x in a)
CPU times: user 14.5 ms, sys: 712 µs, total: 15.2 ms
Wall time: 14.5 ms
Out[11]: 49798.42595258606

In [12]: %time sqrt(sum(x**2 for x in a))
CPU times: user 34.7 ms, sys: 971 µs, total: 35.7 ms
Wall time: 34.8 ms
Out[12]: 181.99014992485522
```

Overall comparison:

```
$ time python3 knn.py 1 3
dimensionality: 92
k=3    train_err 11.5% (+: 24.0%) dev_err 19.40% (+: 25.4%)

real 0m9.639s
user 0m9.525s
sys 0m0.228s

$ time python3 knn.py 2 3
dimensionality: 92
k=3    train_err 11.5% (+: 24.0%) dev_err 19.20% (+: 25.8%)

real 0m9.689s
user 0m9.573s
sys 0m0.218s
```

Overall, their running times are very similar. So the conclusion is that, with numpy, they are almost the same in both accuracy and speed.

7) Redo the evaluation using all-binarized features (with Euclidean). Better or worse? Does it make sense?

```
k=1    train_err 1.5% (+:25.1%) dev_err 23.2% (+:24.8%)
k=3    train_err 11.8% (+:23.3%) dev_err 18.9% (+:22.5%)
k=5    train_err 14.6% (+:22.2%) dev_err 18.1% (+:21.3%)
k=7    train_err 15.2% (+:21.6%) dev_err 17.7% (+:20.5%)
k=9    train_err 16.0% (+:21.1%) dev_err 17.5% (+:20.1%)
k=99   train_err 18.2% (+:17.1%) dev_err 15.7% (+:16.5%)
k=999  train_err 22.2% (+:4.8%) dev_err 20.0% (+:5.2%)
k=9999 train_err 25.0% (+:0.0%) dev_err 23.6% (+:0.0%)
```

Naive binarization loses the numerical influence. Now age=50 vs. age=20 and age=21 vs. age=20 contribute the same to the distances.
Naive binarization could make features more sparse, thus reduce the efficiency of these numerical features, as well as increase the amount of calculation.

4. Deployment

   1) Now try more k's and take your best model and run it on the semi-blind test data, and produce income.test.predicted, which has the same format as the training and dev files.
   Q: At which k and with which distance did you achieve the best dev results?
   Q: What's your best dev error rates and the corresponding positive ratios?
   Q: What's the positive ratio on test?
   Part of your grade will depend on the accuracy of income.test.predicted.

k=39 dev_err 14.5% (+: 20.3%) with Manhattan distance.
k=41 dev_err 14.6% (+: 20.0%) with Euclidean distance.
The following are results on the dev set with more k with Manhattan and Euclidean distance respectively:

```
k=11   dev_err 16.2% (+: 21.8%)  dev_err 16.0% (+: 21.4%)
k=13   dev_err 16.1% (+: 22.7%)  dev_err 16.7% (+: 21.9%)
k=15   dev_err 15.5% (+: 21.9%)  dev_err 15.9% (+: 21.5%)
k=17   dev_err 14.9% (+: 21.5%)  dev_err 15.6% (+: 21.6%)
k=19   dev_err 16.3% (+: 20.9%)  dev_err 16.2% (+: 21.0%)
k=21   dev_err 16.2% (+: 21.2%)  dev_err 15.8% (+: 21.2%)
k=23   dev_err 15.6% (+: 21.4%)  dev_err 15.3% (+: 21.3%)
k=25   dev_err 15.3% (+: 20.7%)  dev_err 15.4% (+: 21.4%)
k=27   dev_err 15.6% (+: 20.6%)  dev_err 15.3% (+: 20.9%)
k=29   dev_err 15.5% (+: 20.7%)  dev_err 15.4% (+: 20.8%)
k=31   dev_err 15.4% (+: 20.4%)  dev_err 15.7% (+: 20.9%)
k=33   dev_err 15.5% (+: 20.3%)  dev_err 15.6% (+: 20.2%)
k=35   dev_err 15.1% (+: 20.7%)  dev_err 15.2% (+: 21.0%)
k=37   dev_err 15.1% (+: 20.3%)  dev_err 15.0% (+: 20.8%)
k=39   dev_err 14.5% (+: 20.3%)  dev_err 14.7% (+: 20.7%)
k=41   dev_err 14.7% (+: 20.1%)  dev_err 14.6% (+: 20.0%)
k=43   dev_err 14.9% (+: 19.9%)  dev_err 14.6% (+: 20.0%)
k=45   dev_err 15.1% (+: 19.9%)  dev_err 14.6% (+: 20.2%)
k=47   dev_err 15.4% (+: 19.8%)  dev_err 15.0% (+: 19.8%)
k=49   dev_err 15.2% (+: 20.0%)  dev_err 14.9% (+: 19.9%)
......
```

Using $k = 39$ and Manhattan distance, the positive ratio on semi-blind test data is 20.7%.

5. Observations

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

i. No "learning" is performed; it doesn't extract any useful information (such as a trained modelin other classifiers) from the raw data.

ii. Testing is extremely slow. Testing complexity grows linearly with the training data size. Thus, we cannot use k-NN on big data.

iii. All dimensions are equally important.

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?

Yes, machine learning tends to exaggerate the existing bias in the training data. It is due to underfitting.

For example, income positive ratio for females in the training data set is 12.94% (206 females with income >50K out of 1592 females); in the dev set is 12.70% (40/315). However, on the dev set, only 20 females are predicted to have income >50K (positive ratio is only 6.35%).

You can observe a similar trend for those highly-educated (overly positive), those poorly-educated (overly negative), certain ethnic groups (underrepresented groups might be overly negative), etc.

This is similar to "prototype bias", and it is an increasingly important social issue as machine learning algorithms are being used increasingly more often in our modern society.

3) 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: (a) broadcasting (such as matrix - vector); (b) np.linalg.norm(..., axis=1); (c) np.argsort() or np.argpartition(); (d) 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.

a) broadcasting could save 80% of the total runtime. (A matrix including 5000 training samples - one test sample)
b) `np.linalg.norm(training_samples - one_test_sample, axis=1)` to calculate 5000 distances for one sample.
c) `np.argsort()` or `np.argpartition()`. Using the latter is faster due to quickselect.

4) 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.

```
$ ssh access.engr.oregonstate.edu

$ time python3 knn.py 2 99
dimensionality: 92
k=99   train_err 17.8% (+: 19.5%) dev_err 15.60% (+: 19.2%)

real 0m22.340s
user 0m12.405s  <----- mine is about 12.4 secs
sys 0m12.001s
```

5) What is a Voronoi diagram (shown in k-NN slides)? How does it relate to k-NN?

Voronoi diagram corresponds to 1-NN (see slides). Most students did not point out the 1-NN part.