Applied Machine Learning HW1: Feature Map and k-NN (15%)

Due Monday April 20 @ 11:59pm on Canvas

(you should finish part 1 by the end of week 1, and part 2 by the end of week 2!)

Instructions:

1. This HW, like all other programming HWs, should be done in Python and numpy only. Either Python 2 or 3 is fine. See the course homepage for a numpy tutorial. If you don't have a Python+numpy installation yourself, you can use the College of Engineering servers by `ssh access.engr.oregonstate.edu`. If you don't have an ENGR account, ~~email us~~ see `https://it.engineering.oregonstate.edu/get-engr-account`.
2. Besides machine learning, this HW also teaches you data (pre-)processing skills and Unix (Linux or Mac OS X) command lines tools. These skills are even more important than machine learning itself for a software engineer or data scientist. As stated in the syllabus, Windows is **not** supported. Use **cmder, cygwin**, or ssh (see above) if you don't have Linux or Mac OS X on your own computer.
3. Ask questions on ~~Canvas~~ **Slack**. You're encouraged to answer other students' questions as well.
4. Do not use machine learning packages such as `sklearn`, though you can use them to verify your results.
5. Do not use data analysis packages such as `pandas` or `seaborn`. Your code should only depend on standard built-in packages plus `numpy`.
6. Download HW1 data from the course homepage. It contains the training data, the dev set (held-out), and a semi-blind test set. The test set does not contain target labels, and you will need to predict them using your best model. Part of your grade is based on your prediction accuracy on test.
7. You should submit a single `.zip` file containing `hw1-report.pdf`, `income.test.predicted`, and all your code. LaTeX'ing is recommended but not required. **Do not forget the debrief section (in each HW).**

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?

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? Hint:

   `cat income.train.txt.5k | sort -nk1 | head -1`

3. There are two types of fields, *numerical* (*age* and *hours-per-week*), and *categorical* (everything else).[1] 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

[1] In principle, we could also convert *education* to a numerical feature, but we choose **not** to do it to keep it simple.

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?

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

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** (N.B.: not 1! why?)).

   Hint: first, observe that the max distance between two people on a categorial field is 2. If we simply "normalize" a numerical field by, say, age / 100, it might look OK but now what's the max distance between two people on age? Are you treating all fields equally?

6. Q: How many features do you have in total (i.e., the dimensionality)? Hint: should be around ~~100~~ **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
   ```

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

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.

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:

1. 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
   ```

2. Redo the above using Manhattan distance.

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

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

YOU SHOULD FINISH EVERYTHING UP TO HERE BY THE END OF WEEK 2.

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.

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

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%).

4. Now report both training and ~~testing~~ **dev** errors (your code needs to run a lot faster! See Question ~~4.3~~ **5.3** 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.

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

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

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

4 Deployment

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`.

5 Observations

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

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?

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.

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. (Mine was about 14 seconds).

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

Debriefing (required in your report)

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

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

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

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

5. Any other comments?