CPSC 340 Assignment 1 (due 2018-01-17 at 9:00pm)

Instructions

Rubric: {mechanics:5}

IMPORTANT! Before proceeding, please carefully read the general homework instructions at `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/home/blob/master/homework_instructions.md`. Please pay special attention to the section on visualizations as there were no visualizations required in a0.

A note on the provided code: in the *code* directory we provide you with a file called *main.py*. This file, when run with different arguments, runs the code for different parts of the assignment. For example,

```
python main.py -q 1.1
```

runs the code for Question 1.1. At present, this should do nothing, because the code for Question 1.1 still needs to be written (by you). But we do provide some of the bits and pieces to save you time, so that you can focus on the machine learning aspects. The file *utils.py* contains some helper functions. You're not required to read/understand the code in there (although you're welcome to!) and will not need to modify it.

1 Data Exploration

Your repository contains the file *fluTrends.csv*, which contains estimates of the influenza-like illness percentage over 52 weeks on 2005-06 by Google Flu Trends. Your *main.py* loads this data for you and stores it in a pandas DataFrame `X`, where each row corresponds to a week and each column corresponds to a different region. If desired, you can convert from a DataFrame to a raw numpy array with `X.values()`.

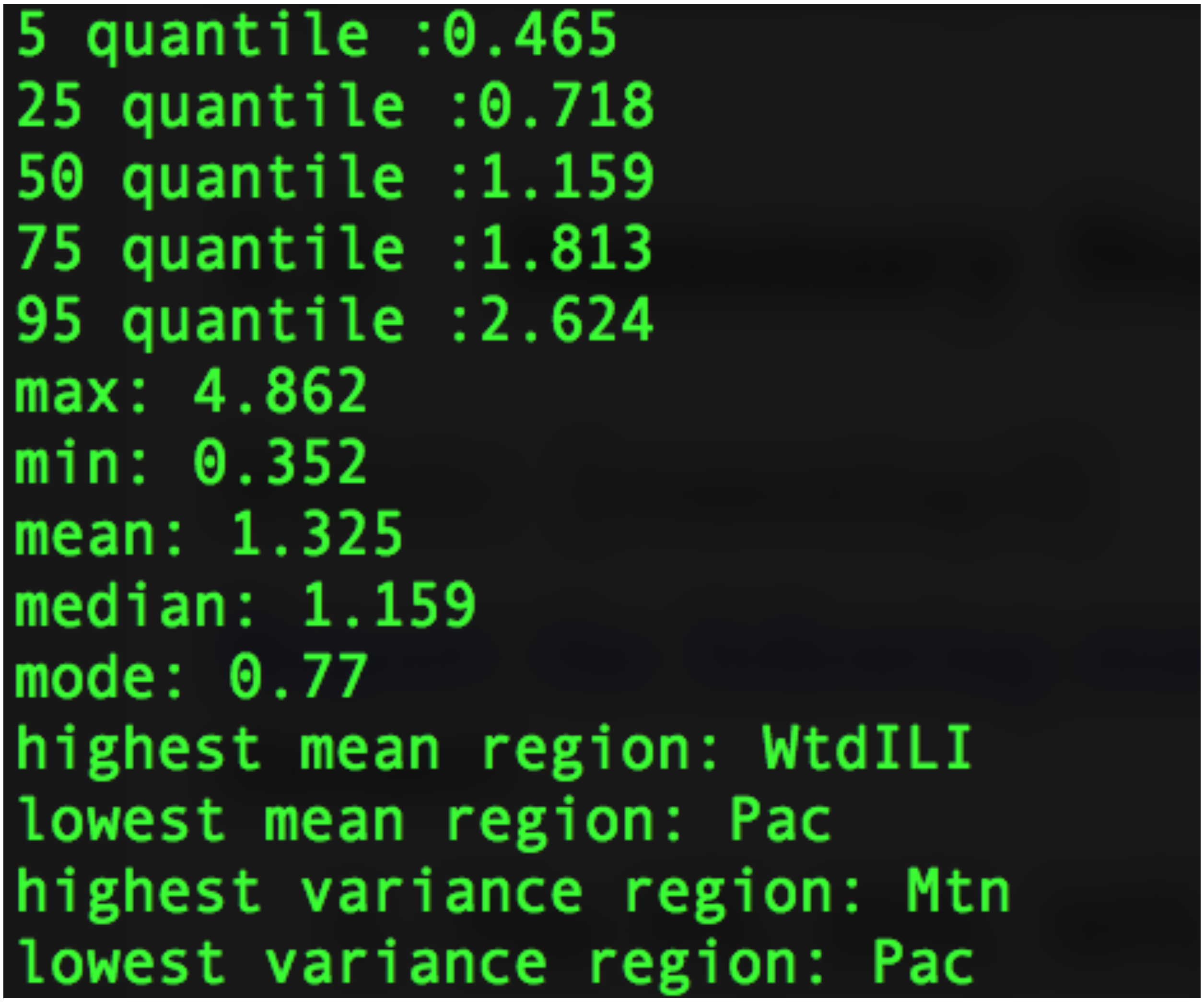
1.1 Summary Statistics

Rubric: {reasoning:2}

Report the following statistics: The minimum, maximum, mean, median, and mode of all values across the dataset.

1. The 5%, 25%, 50%, 75%, and 95% quantiles of all values across the dataset.
2. The names of the regions with the highest and lowest means, and the highest and lowest variances.

In light of the above, is the mode a reliable estimate of the most "common" value? Describe another way we could give a meaningful "mode" measurement for this (continuous) data. Note: the function `utils.mode()` will compute the mode value of an array for you.



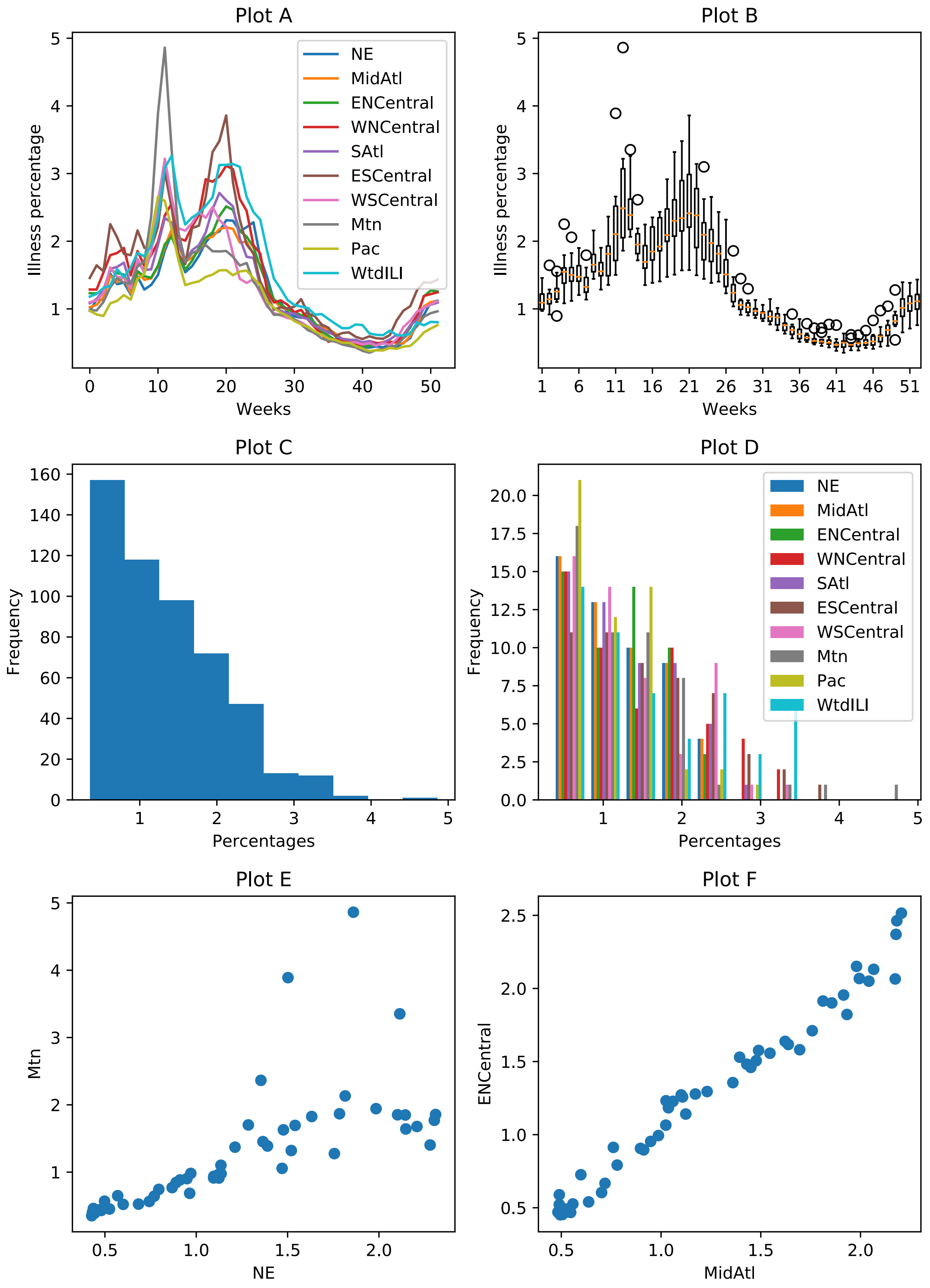
```
5 quantile :0.465
25 quantile :0.718
50 quantile :1.159
75 quantile :1.813
95 quantile :2.624
max: 4.862
min: 0.352
mean: 1.325
median: 1.159
mode: 0.77
highest mean region: WtdILI
lowest mean region: Pac
highest variance region: Mtn
lowest variance region: Pac
```

each region should have its own most common value,it does not make sense to have one most common value for all regions, the value may just appear in a single region so many times

1.2 Data Visualization

Rubric: {reasoning:3}

Consider the figure below.


Plot A
Illness percentage
Weeks
0
10
20
30
40
50
1
2
3
4
5
NE
MidAtl
ENCentral
WNCentral
SAtl
ESCentral
WSCentral
Mtn
Pac
WtdILI
Plot B
Illness percentage
Weeks
1
6
11
16
21
26
31
36
41
46
51
Plot C
Frequency
Percentages
0
20
40
60
80
100
120
140
160
Plot D
Frequency
Percentages
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
NE
MidAtl
ENCentral
WNCentral
SAtl
ESCentral
WSCentral
Mtn
Pac
WtdILI
Plot E
Mtn
NE
0.5
1.0
1.5
2.0
Plot F
ENCentral
MidAtl
0.5
1.0
1.5
2.0
2.5

The figure contains the following plots, in a shuffled order:

1. A single histogram showing the distribution of *each* column in X.
   D: have legends for each region

2. A histogram showing the distribution of each the values in the matrix X.
   C: x-axis is percentages

3. A boxplot grouping data by weeks, showing the distribution across regions for each week.
   B: box plot is apparently like this

4. A plot showing the illness percentages over time.
   A: x-axis is time, y-axis is percentage

5. A scatterplot between the two regions with highest correlation.
   F: values have an approximately linear relationship

6. A scatterplot between the two regions with lowest correlation.
   E: values are scattered in a wider range

Match the plots (labeled A-F) with the descriptions above (labeled 1-6), with an extremely brief (a few words is fine) explanation for each decision.

2 Decision Trees

If you run `python main.py -q 2`, it will load a dataset containing longtitude and latitude data for 400 cities in the US, along with a class label indicating whether they were a "red" state or a "blue" state in the 2012 election.[1] Specifically, the first column of the variable X contains the longitude and the second variable contains the latitude, while the variable y is set to 1 for blue states and 2 for red states. After it loads the data, it plots the data and then fits two simple classifiers: a classifier that always predicts the most common label (1 in this case) and a decision stump that discretizes the features (by rounding to the nearest integer) and then finds the best equality-based rule (i.e., check if a feature is equal to some value). It reports the training error with these two classifiers, then plots the decision areas made by the decision stump.

2.1 Splitting rule

Rubric: {reasoning:1}

Is there a particular type of features for which it makes sense to use an equality-based splitting rule rather than the threshold-based splits we discussed in class?

the values after rounding are pretty close so that it would be hard to find a threshold value for splitting

2.2 Decision Stump Implementation

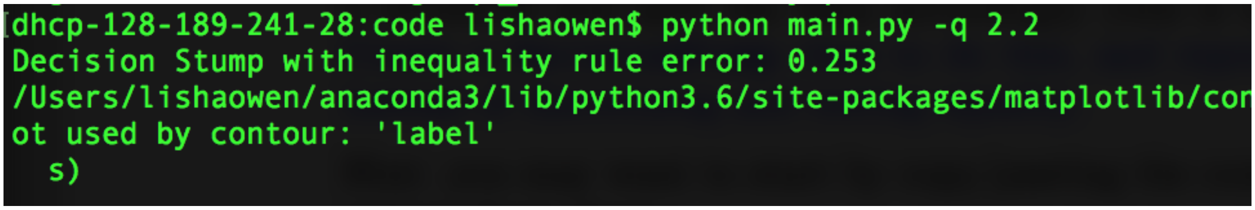
Rubric: {code:3}

The file *decision_stump.py* contains the class `DecisionStumpEquality` which finds the best decision stump using the equality rule and then makes predictions using that rule. Instead of discretizing the data and using a rule based on testing an equality for a single feature, we want to check whether a feature is above or below a threshold and split the data accordingly (this is the more sane approach, which we discussed in class).

[1] The cities data was sampled from `http://simplemaps.com/static/demos/resources/us-cities/cities.csv`. The election information was collected from Wikipedia.

Create a `DecisionStump` class to do this, and report the updated error you obtain by using inequalities instead of discretizing and testing equality.

Hint: you may want to start by copy/pasting the contents `DecisionStumpEquality` and then make modifications from there.



```
[dhcp-128-189-241-28:code lishaowen$ python main.py -q 2.2
Decision Stump with inequality rule error: 0.253
/Users/lishaowen/anaconda3/lib/python3.6/site-packages/matplotlib/con
ot used by contour: 'label'
  s)
```

2.3 Constructing Decision Trees

Rubric: {code:2}

Once your decision stump class is finished, running `python main.py -q 2.3` will fit a decision tree of depth 2 to the same dataset (which results in a lower training error). Look at how the decision tree is stored and how the (recursive) *predict* function works. Using the same splits as the fitted depth-2 decision tree, write an alternative version of the predict function for classifying one training example as a simple program using if/else statements (as in slide 6 of the Decision Trees lecture). Save your code in a new file called `simple_decision.py` (in the *code* directory) and make sure you link to this file from your README.

Note: this code should implement the specific, fixed decision tree which was learned after calling `fit` on this particular data set. It does not need to be a learnable model.

`https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/c7k0b_a1/blob/master/code/simple_decision.py`

2.4 Decision Tree Training Error

Rubric: {reasoning:2}

Running `python main.py -q 2.4` fits decision trees of different depths using two different implementations: first, our own implementation using your DecisionStump, and second, the decision tree implementation from the popular Python ML library *scikit-learn*. The decision tree from sklearn uses a more sophisticated splitting criterion called the information gain, instead of just the classification accuracy. Run the code and look at the figure. Describe what you observe. Can you explain the results?

the error of out algorithm plateaus after a relatively shallower depth and the error is high, while it levels out deeper and reaches lower error in the scikit-learn

Note: we set the `random_state` because sklearn's DecisionTreeClassifier is non-deterministic. I'm guessing this is because it breaks ties randomly.

Note: the code also prints out the amount of time spent. You'll notice that sklearn's implementation is substantially faster. This is because our implementation is based on the $O(n^2d)$ decision stump learning algorithm and sklearn's implementation presumably uses the faster $O(nd\log n)$ decision stump learning algorithm that we discussed in lecture.

our naive implementation cannot exploit the data that much like the scikit-learn implementation,after certain depth when the classification tends to be more similar after splitting, it is not that meaningful to continue splitting, while the information gain algorithm tends to get some useful information every step, so increase in depth becomes more useful

2.5 Cost of Fitting Decision Trees

Rubric: {reasoning:3}

In class, we discussed how in general the decision stump minimizing the classification error can be found in $O(nd \log n)$ time. Using the greedy recursive splitting procedure, what is the total cost of fitting a decision tree of depth m in terms of n, d, and m?

Hint: even thought there could be $(2^m - 1)$ decision stumps, keep in mind not every stump will need to go through every example. Note also that we stop growing the decision tree if a node has no examples, so we may not even need to do anything for many of the $(2^m - 1)$ decision stumps.

3 Training and Testing

If you run `python main.py -q 3`, it will load *citiesSmall.pkl*. Note that this file contains not only training data, but also test data, `X_test` and `y_test`. After training a depth-2 decision tree it will evaluate the performance of the classifier on the test data.[2] With a depth-2 decision tree, the training and test error are fairly close, so the model hasn't overfit much.

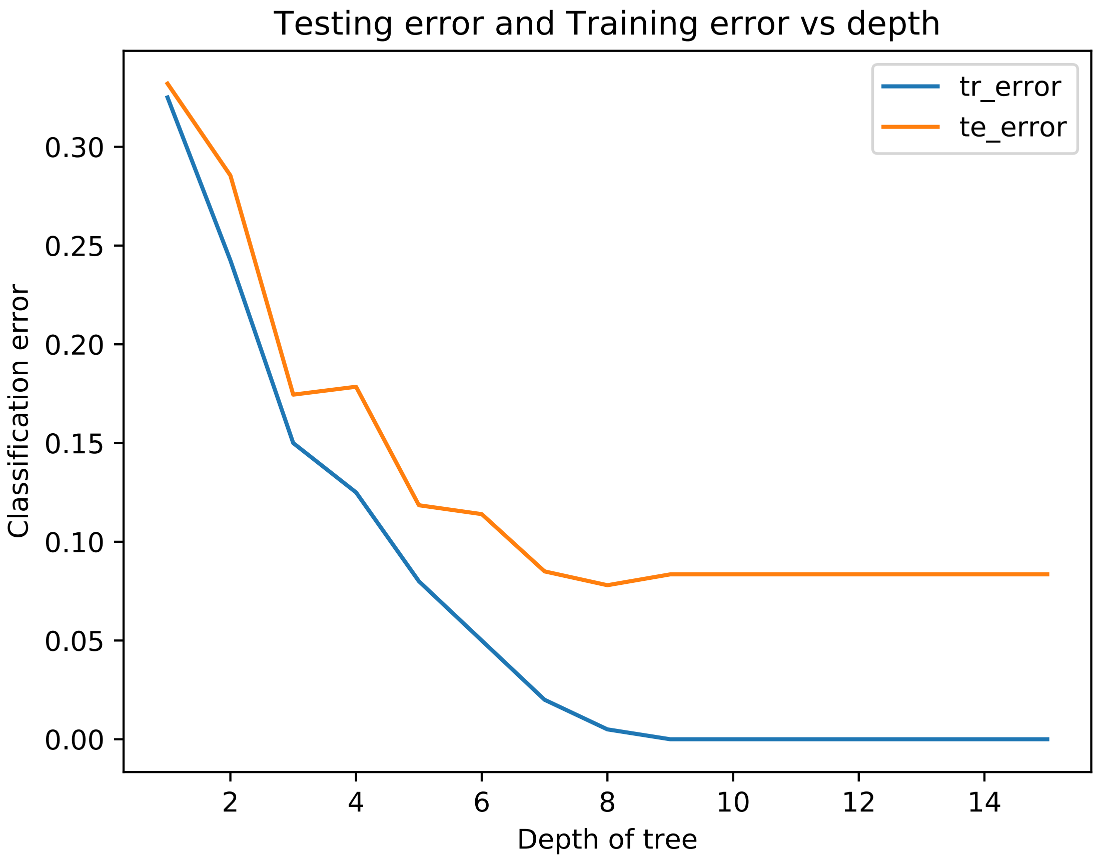
3.1 Training and Testing Error Curves

Rubric: {reasoning:2}

Make a plot that contains the training error and testing error as you vary the depth from 1 through 15. How do each of these errors change with the decision tree depth?

Note: it's OK to reuse the provided code from part 2.4 as a starting point.

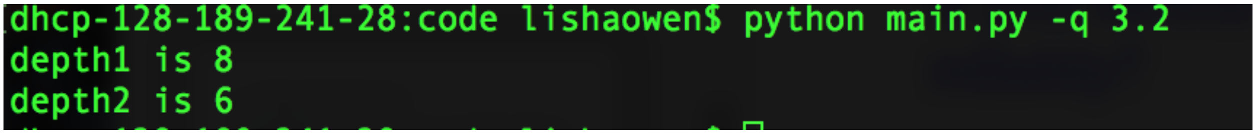
[2] The code uses the "information gain" splitting criterion; see the Decision Trees bonus slides for more information.



3.2 Validation Set

Rubric: {reasoning:3}

Suppose that we didn't have an explicit test set available. In this case, we might instead use a *validation* set. Split the training set into two equal-sized parts: use the first $n/2$ examples as a training set and the second $n/2$ examples as a validation set (we're assuming that the examples are already in a random order). What depth of decision tree would we pick to minimize the validation set error? Does the answer change if you switch the training and validation set? How could use more of our data to estimate the depth more reliably?



```
dhcp-128-189-241-28:code lishaowen$ python main.py -q 3.2
depth1 is 8
depth2 is 6
```

I would pick depth 8, and the answer changes as I switch two sets.
We can use cross validation for more accuracy in deciding the depth

4 K-Nearest Neighbours

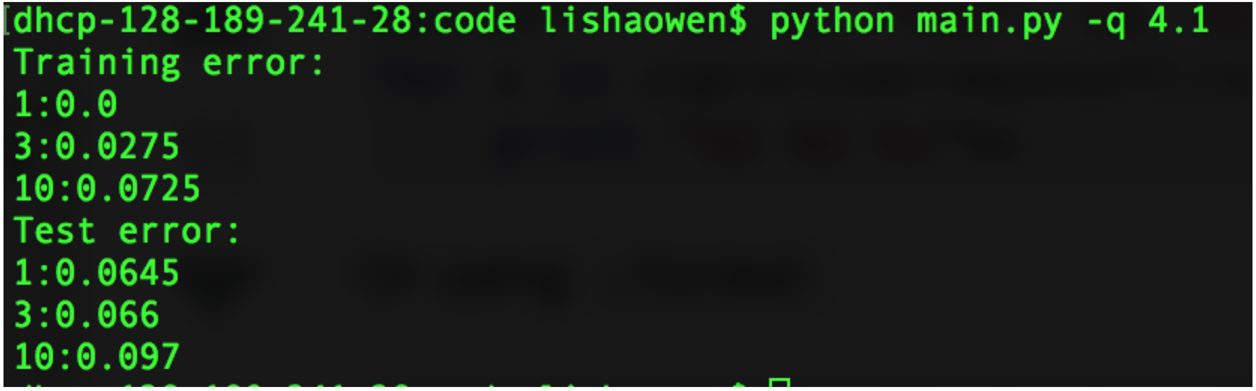
In the *citiesSmall* dataset, nearby points tend to receive the same class label because they are part of the same U.S. state. This indicates that a k-nearest neighbours classifier might be a better choice than a decision tree. The file *knn.py* has implemented the training function for a k-nearest neighbour classifier (which is to just memorize the data).

4.1 KNN Prediction

Rubric: {code:3, reasoning:4}

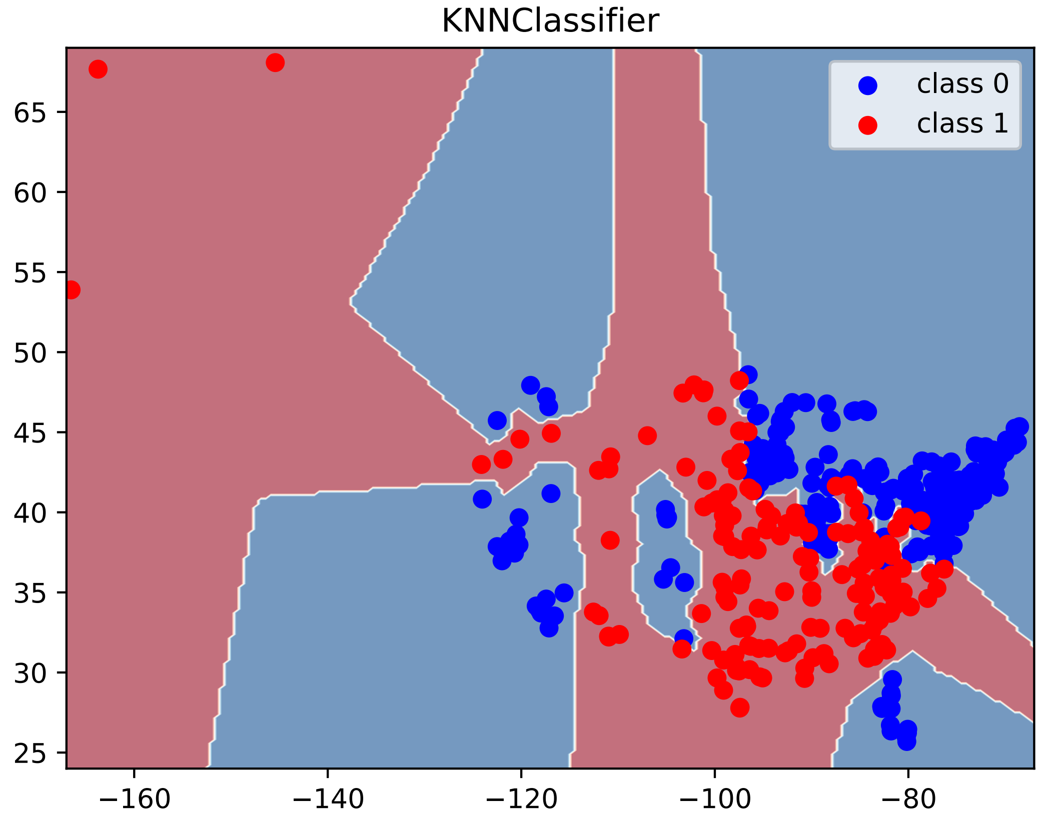
Fill in the *predict* function in *knn.py* so that the model file implements the k-nearest neighbour prediction rule. You should Euclidean distance, and may numpy's *sort* and/or *argsort* functions useful. You can also use *utils.euclidean_dist_squared*, which computes the squared Euclidean distances between all pairs of points in two matrices.

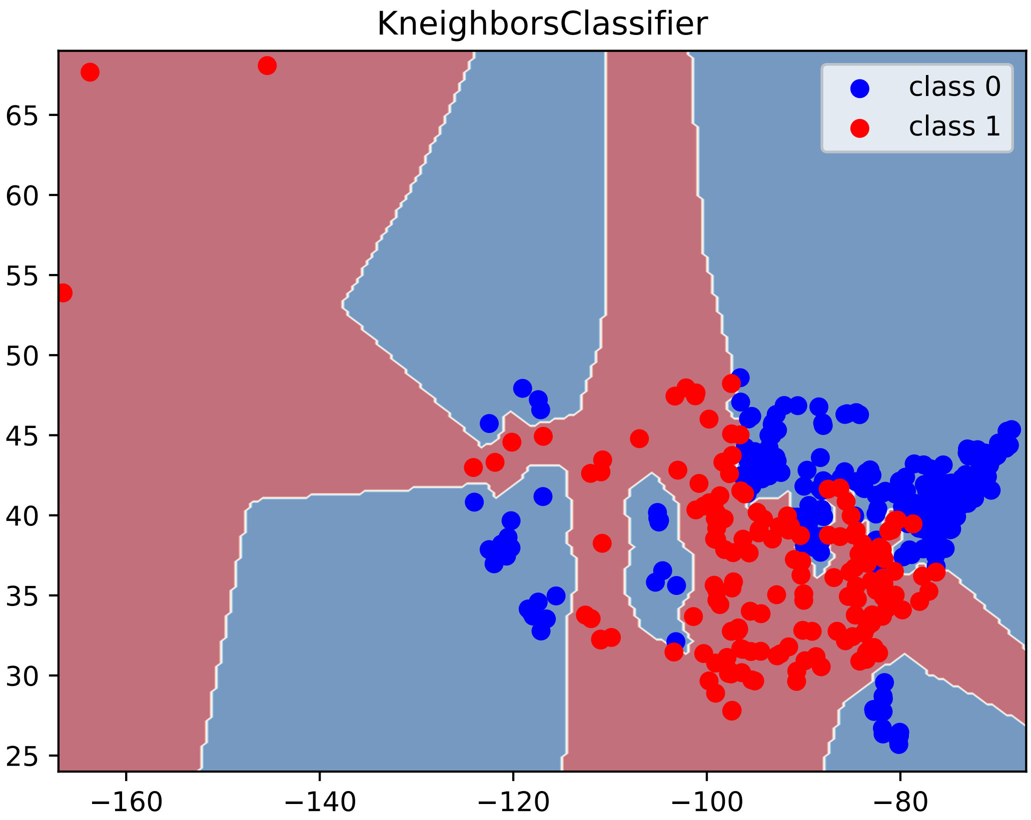
1. Write the *predict* function.
   `https://github.ugrad.cs.ubc.ca/CPSC340-2017W-T2/c7k0b_a1/blob/master/code/knn.py`
2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 3$, and $k = 10$. How do these numbers compare to what you got with the decision tree?

   ```
   [dhcp-128-189-241-28:code lishaowen$ python main.py -q 4.1
   Training error:
   1:0.0
   3:0.0275
   10:0.0725
   Test error:
   1:0.0645
   3:0.066
   10:0.097
   ```

   they are lower than those of decision tree
3. Hand in the plot generated by *utils.plotClassifier* on the *citiesSmall* dataset for $k = 1$, using both your implementation of KNN and the KNeighborsClassifier from scikit-learn.


KNNClassifier
class 0
class 1
65
60
55
50
45
40
35
30
25
−160
−140
−120
−100
−80



4. Why is the training error 0 for $k = 1$?
   when you predict on the training set, the closet one is the one you use to train itself, it is equal to look up the correct answer
5. If you didn't have an explicit test set, how would you choose k?
   use cross validation

4.2 Condensed Nearest Neighbours

Rubric: {code:3, reasoning:5}

The dataset *citiesBig1* contains a version of this dataset with more than 30 times as many cities. KNN can obtain a lower test error if it's trained on this dataset, but the prediction time will be very slow. A common strategy for applying KNN to huge datasets is called *condensed nearest neighbours*, and the main idea is to only store a *subset* of the training examples (and to only compare to these examples when making predictions). If the subset is small, then prediction will be faster.

The most common variation of this algorithm works as follows:

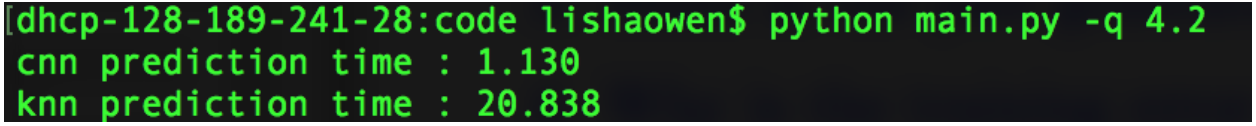
```
initialize subset with first training example;
for each subsequent training example do
    if the example is incorrectly classified by the KNN classifier using the current subset then
        add the current example to the subset;
    else
        do not add the current example to the subset (do nothing);
    end
end
```

Algorithm 1: Condensed Nearest Neighbours

You are provided with an implementation of this *condensed nearest neighbours* algorithm in *knn.py*.

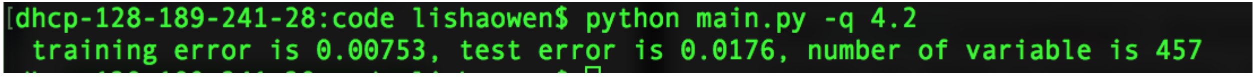
Your tasks:

1. The point of this algorithm is to be faster than KNN. Try running the condensed NN on the *citiesBig1* dataset and report how long it takes to make a prediction. What about if you try to use KNN for this dataset – how long did it take before you panicked and went for CTRL-C... or did it actually finish?



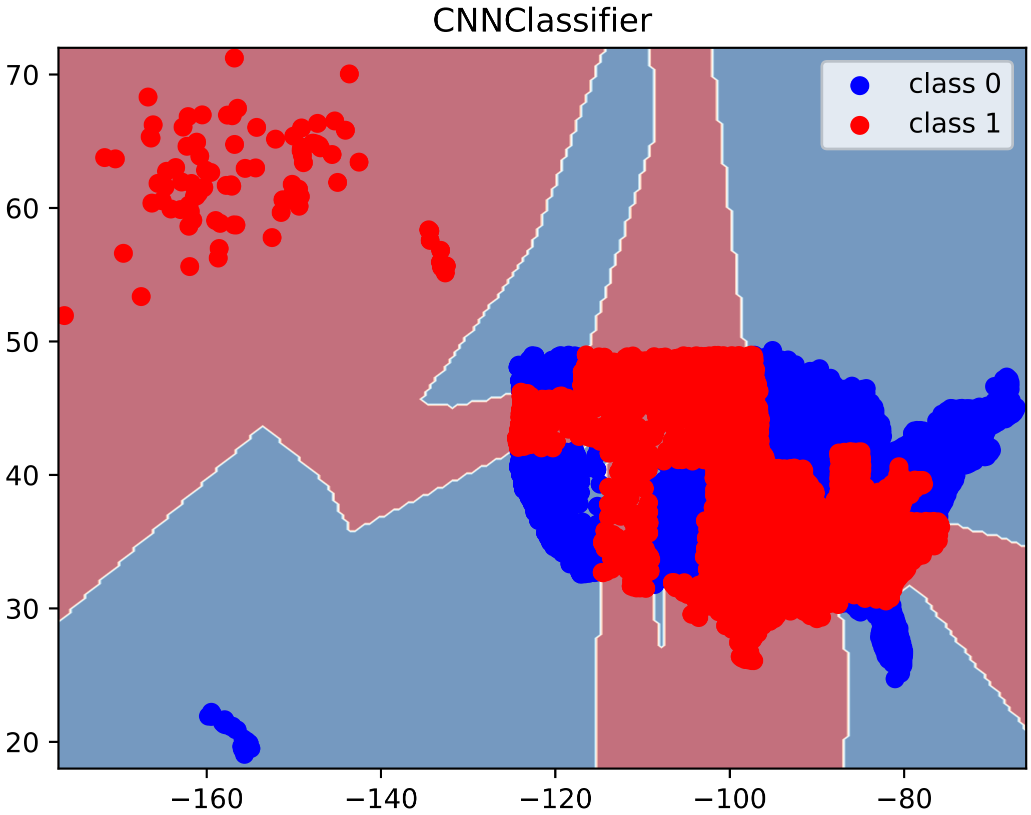
```
[dhcp-128-189-241-28:code lishaowen$ python main.py -q 4.2
 cnn prediction time : 1.130
 knn prediction time : 20.838
```

2. Report the training and testing errors for condensed NN, as well as the number of variables in the subset, on the *citiesBig1* dataset with $k = 1$.



```
[dhcp-128-189-241-28:code lishaowen$ python main.py -q 4.2
  training error is 0.00753, test error is 0.0176, number of variable is 457
```

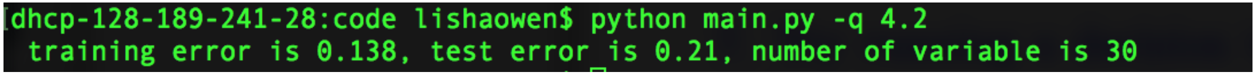
3. Hand in the plot generated by *utils.plotClassifier* on the *citiesBig1* dataset for $k = 1$.



4. Why is the training error with $k = 1$ now greater than 0?
   This time it is not like the case in KNN where you just look up the right answer itself, in CNN since you don't store all training data, there would be some approximation

5. If you have s examples in the subset, what is the cost of running the predict function on t test examples in terms of n, d, t, and s?
   O(sdt)

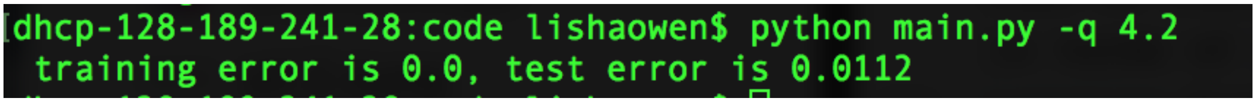
6. Try out your function on the dataset *citiesBig2*. Why are the test error *and* training error so high (even for $k = 1$) for this method on this dataset?



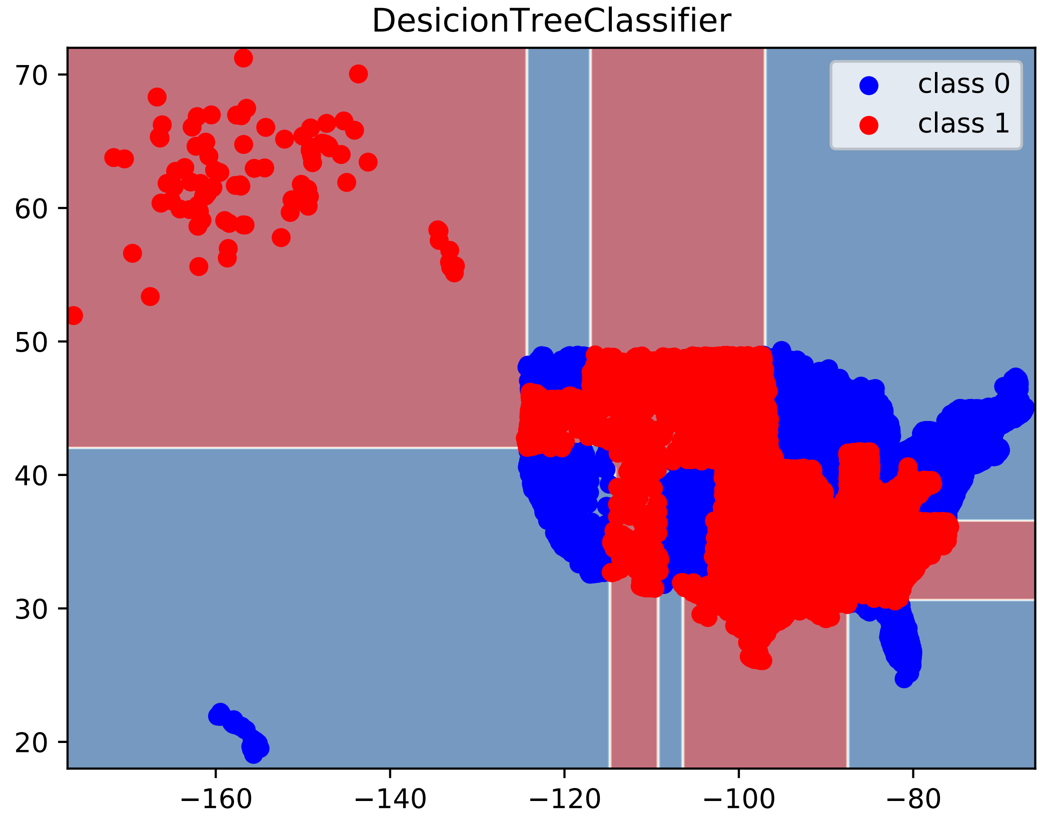
```
[dhcp-128-189-241-28:code lishaowen$ python main.py -q 4.2
 training error is 0.138, test error is 0.21, number of variable is 30
```

   the size of the subset is so small, it might because the training data are so close and not general enough

7. Try running a decision tree on *citiesBig1* using scikit-learn's `DecisionTreeClassifier` with default hyperparameters. Does it work? Is it fast? Any other thoughts? Overall, do you prefer decicison trees of (condensed) KNN for this data set? Include the usual plot of the decision surface.



```
[dhcp-128-189-241-28:code lishaowen$ python main.py -q 4.2
 training error is 0.0, test error is 0.0112
```



it works and the test error is relatively low. And it is pretty fast, at least not slower then CNN. For this type of data , i prefer to use KNN or CNN, since the euclidean distance is quite similar to the real distance. Thus, the outcome can be more convincing.

5 Very-Short Answer Questions

Rubric: {reasoning:8}

Write a short one or two sentence answer to each of the questions below. Make sure your answer is clear and concise.

1. What is one reason we would want to look at scatterplots of the data before doing supervised learning?
   To get a sense of the pattern of the data, or check if there is any "dirty data"
2. What is a reason that the examples in a training and test set might not be IID?
   sequence of data might not be random when we split the training and test data or data are not sampled independently
3. What is the difference between a validation set and a test set?
   test set are the data that we don't know the labels while validation set are those we know the labels and we use validation error to approximate test error Validation set is part of the training set

4. What is the main advantage of non-parametric models?
   the model becomes more complicated as we get more data, so more data help more and enable us to make more accurate prediction

5. A standard pre-processing step is "standardization" of the features: for each column of X we subtract its mean and divide by its variance. Would this pre-processing change the accuracy of a decision tree classifier? Would it change the accuracy of a KNN classifier?
   It won't change the accuracy of decision tree, but it do effect KNN, without scaling, one feature might be dominant in calculating the euclidean distance and influence the prediction

6. Does increasing k in KNN affect the training or prediction asymptotic runtimes?
   It does not influence the training runtime since actually there is no training in KNN. It also doesn't influence the prediction runtime since no matter what K you use, you will calculate the distance with all training data, K only decides how many of them you pick out to do prediction

7. How does increase the parameter k in k-nearest neighbours affect the two parts (training error and approximation error) of the fundamental trade-off (hint: think of the extreme values).
   increasing in K will make training error larger since you might involve some data which actually are not quite close, but it will decrease the approximation error since more K makes the data we use to predict more general thus we can make more accurate prediction

8. For any parametric model, how does increasing number of training examples n affect the two parts of the fundamental trade-off.

~~more training examples make the model more complicated and decrease the training error. in terms of the test error, at first, more training examples do help and make the test error lower but eventually bringing more training data does not help decrease the test error~~

Increase number of training examples will increase the training error, but it can make the training error a good approximation of the test error