CPSC 340 Assignment 1 (due 2017-01-22 at 11:59pm)

Data Exploration, Decision Trees, Training and Testing, Naive Bayes

IMPORTANT!!!!! Before proceeding, please carefully read the general homework instructions at `https://github.ubc.ca/cpsc340/home/blob/master/homework_instructions.md`.

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 homework 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. For example, you'll see that the provided code already loads the datasets for you. The file *utils.py* contains some helper functions. You don't need to understand or modify the code in there. To complete your assignment, you will need to modify *main.py*, *decision_stump.py*, and *naive_bayes.py*.

We have tried to provide high-qualty code. However, if we come across any bugs in the provided code, we will send you a pull request to fix it. You will then receive a notification from GitHub.

We use blue to highlight the deliverables that you must answer/do/submit with the assignment.

1 Data Exploration

Your repository contains the file *fluTrends.pkl*, 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 the 2-D numpy array `X`, where each row corresponds to a week and each column corresponds to a different region. The names of the columns are loaded into the variable `names`.[1]

1.1 Summary Statistics

Report the following statistics:

1. The minimum, maximum, mean, median, and mode of all values across the dataset.
   Min = 0.352 Max = 4.862 Mean = 1.325 Median = 1.159 Mode = 0.770
2. The 10%, 25%, 50%, 75%, and 90% quantiles across the dataset.
   10th quantile = 0.502 25th quantile = 0.718 50th quantile = 1.159 75th quantile = 1.813 90th quantile = 2.315

[1] If you are already a Python user, you may be wondering why we aren't using Pandas. This is certainly a great library and you are welcome to convert the data to a Pandas DataFrame if you wish. But my current plan is to avoid Pandas in this course, in order to reduce the number of libraries that you are required to learn.

3. The regions with the highest and lowest means, and the highest and lowest variances.
   Region with: highest mean at WtdILI, lowest mean at Pac
   Region with: highest variance at Mtn, lowest variance at Pac

4. The pairs of regions with the highest and lowest correlations.
   Regions with lowest correlation are NE and Mtn,
   Regions with highest correlation are MidAtl and ENCentral.

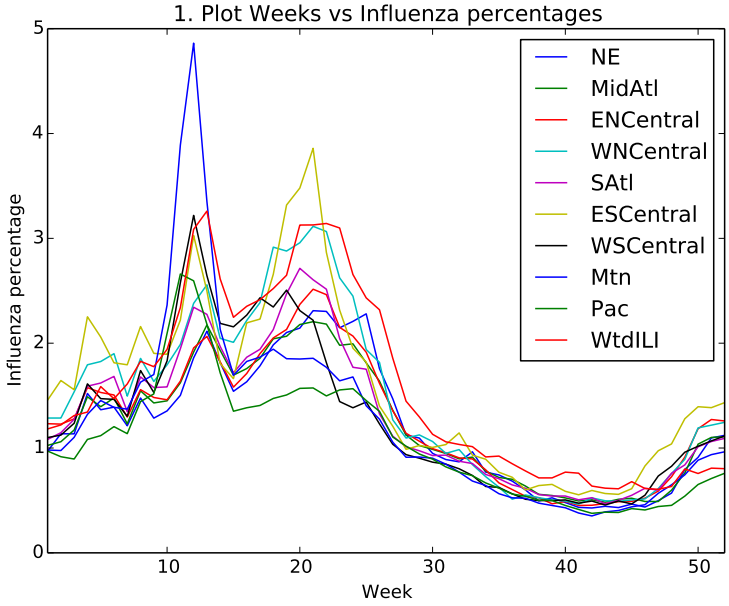
In light of parts 1 and 2, is the mode a reliable estimate of the most "common" value? Describe another way we could give a meaningful "mode" measurement for this (continuous) data.

No this mode is not an accurate measure as decimal values tend to differ, a more accurate mode would be rounding off all the float values in data down to 1 or 2 decimal places and then calculating the mode

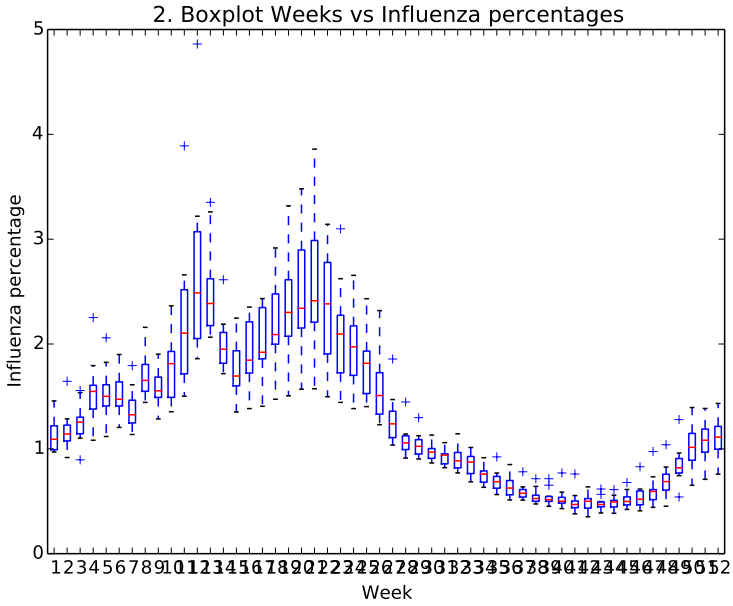
1.2 Data Visualization

Show the following figures:

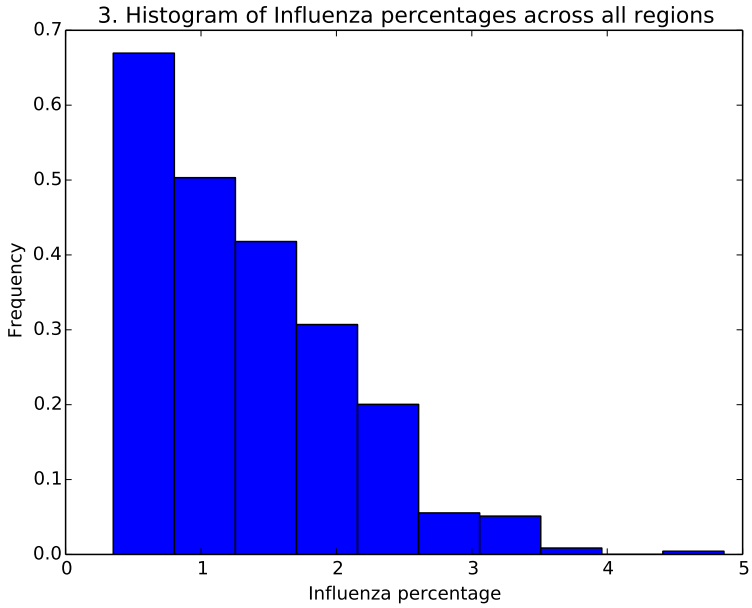
1. A plot containing the weeks on the x-axis and the percentages for each region on the y-axis.



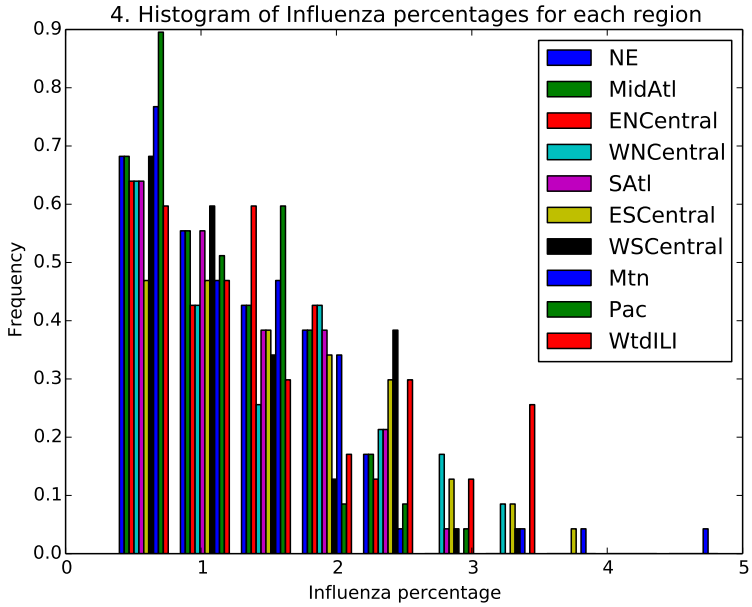
2. A boxplot grouping data by weeks, showing the distribution across regions for each week.



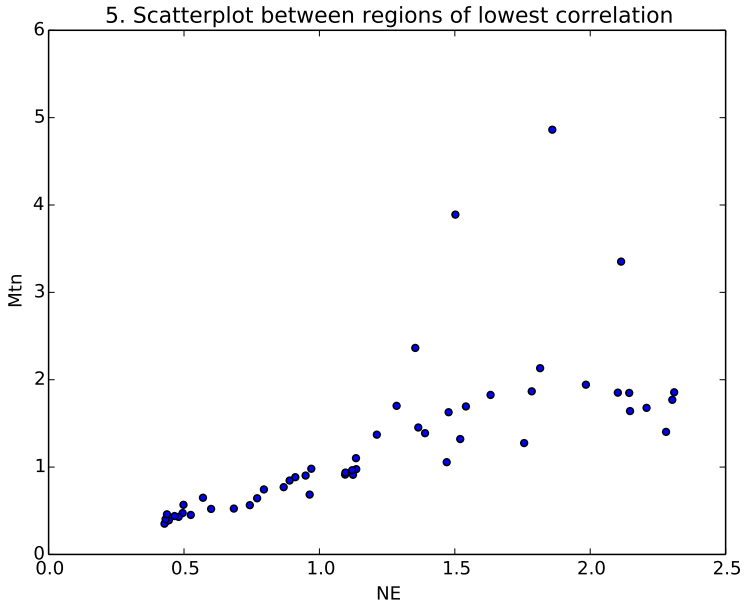
3. A histogram showing the distribution of each the values in the matrix X.



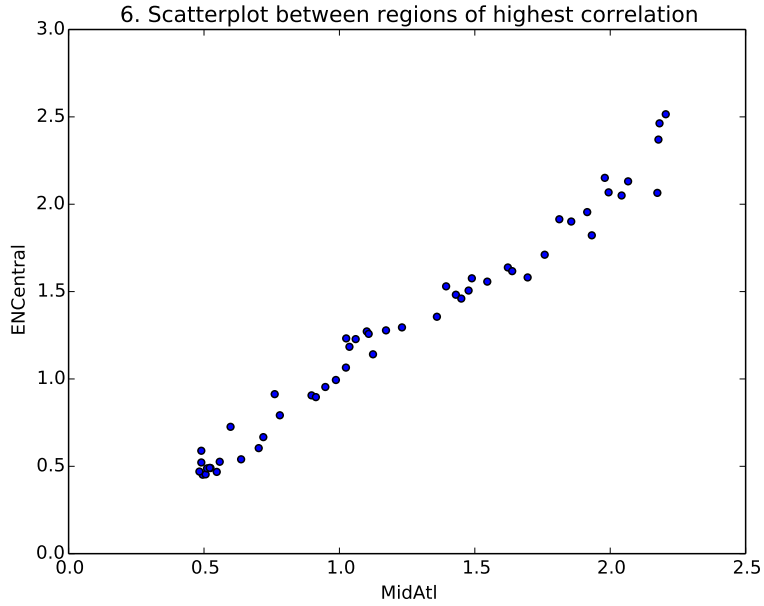
4. A single histogram showing the distribution of *each* column in X.



5. A scatterplot between the two regions with lowest correlation.



6. A scatterplot between the two regions with highest correlation.



Note: this is the first time we are asking you to include a plot with your submission. Please refer to the "Figures" section of the homework instructions document.

I have embedded the CPSC 340 logo below, so you have an example of how to embed a figure in LaTeX.

This figure:

was embedded with this code:

```
\begin{center}
\fig{0.1}{../figs/cpsc340-logo}
\end{center}
```

The 0.1 represents the desired width of the image. The centering commands are for centering the image in the page; you can try removing them to see what happens. The `fig` command is a custom command defined at the top of this file, which calls the `includegraphics` command. This comes from the `graphicx` package which is loaded at the top of this file with the `usepackage` command. Finally, as far as I can tell, including the file extension is optional when including an image in LaTeX. In the above, if you replace `cpsc340-logo` with `cpsc340-logo.png` it should work exactly the same way.

2 Decision Trees

If you run `python main.py -q 2.1`, 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.[2] 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 Decision Stump Implementation

The file *decision_stump.py* contains the functions `fit_equality` and `predict_equality` 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 (as discussed in class). Fill in the `fit` and `predict` functions 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 `fit_equality` and `predict_equality` and making modifications from there.

Decision Stump with inequality rule error = 0.265

2.2 Constructing Decision Trees

Once your decision stump is finished, the code in `decision_tree` will be able to 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 9 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.

Modify the *depth* variable in this demo to fit deeper decision trees. Notice that the error stops decreasing after a certain depth. In contrast, if you call the scikit-learn version (see code) you will notice that the error eventually decreases to 0. Why does the training error stop decreasing when you use classification accuracy?

The training error stops decreasing at a certain point in our code, since our decision stump is not able to fit a model to small datasets, and returns a null model instead which leads to the base case of recursion, whereas the scikit-learn version has more accurate classification in their decision stumps, and are thus able to classify small datasets eventually leading to 0 training error.

2.3 Cost of Fitting Decision Trees

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.

cost = $O(mnd \log n)$

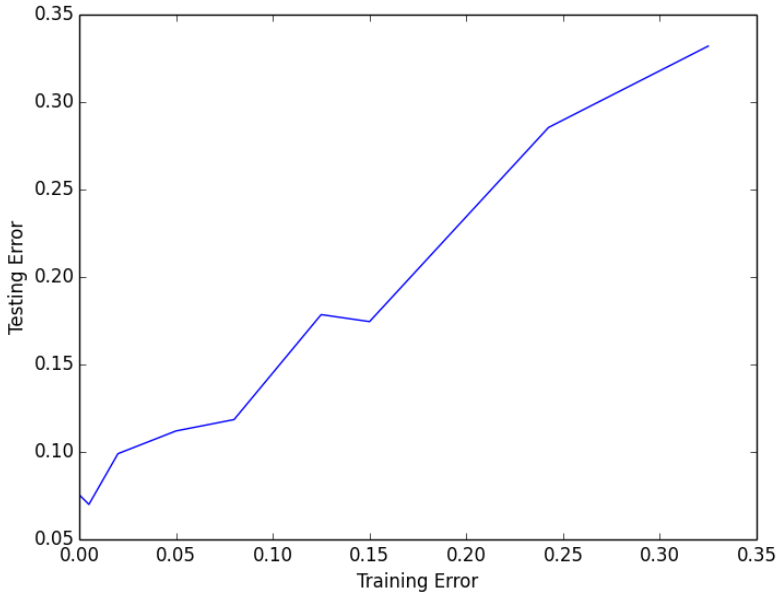
[2] The cities data was sampled from `http://simplemaps.com/static/demos/resources/us-cities/cities.csv`. The election information was collected from Wikipedia.

3 Training and Testing

If you run `python main.py -q 3.1`, 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.[3] 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

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?



The errors decrease up to a certain decision tree depth (7 in this case), at which point on increasing the maximum decision tree depth further, training error remains continues to decrease (if it hasnt reached 0 yet), and testing error increases.

3.2 Validation Set

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?

The validation error is minimized when the decision tree has a max depth of 9. If we switch the training and validation sets, the depth at which validation error is minimized changes to 6. We can divide the data into more parts and use more validation sets to accurately predict the desired depth of the decision tree.

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

4 Naive Bayes

In this section we'll first review a standard way that Bayes rule is used and then explore implementing a naive Bayes classifier, a very fast classification method that is often surprisingly accurate for text data with simple representations like bag of words.

4.1 Bayes rule for drug testing

Suppose a drug test produces a positive result with probability 0.99 for drug users, $P(T = 1|D = 1) = 0.99$. It also produces a negative result with probability 0.99 for non-drug users, $P(T = 0|D = 0) = 0.99$. The probability that a random person uses the drug is 0.001, so $P(D = 1) = 0.001$.

What is the probability that a random person who tests positive is a user, $P(D = 1|T = 1)$?

$P(T = 1) = P(T = 1|D = 1)P(D = 1) + P(T = 1|D = 0)P(D = 0)$

$P(T = 1) = 0.99 * 0.001 + 0.001 * 0.99 = 0.00198$

$P(D = 1|T = 1) = P(T = 1|D = 1)P(D = 1)/P(T = 1)$

$P(D = 1|T = 1) = 0.99 * 0.001/0.00198 = 0.5$

4.2 Naive Bayes by Hand

Consider the dataset below, which has 10 training examples and 2 features:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Suppose you believe that a naive Bayes model would be appropriate for this dataset, and you want to classify the following test example:

$$\hat{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}.$$

(a) Compute the estimates of the class prior probabilities:

- $p(y = 1) = 0.6$
- $p(y = 0) = 0.4$

(b) Compute the estimates of the 4 conditional probabilities required by naive Bayes for this example:

- $p(x_1 = 1|y = 1) = 0.5$
- $p(x_2 = 1|y = 1) = 0.667$
- $p(x_1 = 1|y = 0) = 1$
- $p(x_2 = 1|y = 0) = 0.25$

(c) Under the naive Bayes model and your estimates of the above probabilities, what is the most likely label for the test example? (Show your work.)

We need to calculate $p(y = 1|x_1 = 1, x_2 = 1)$ and $p(y = 0|x_1 = 1, x_2 = 1)$

Under naive Bayes we assume conditional independence of x_1 and x_2 given y, then:

$p(y = 1|x_1 = 1, x_2 = 1) = p(x_1 = 1, x_2 = 1|y = 1)p(y = 1)/p(x_1 = 1, x_2 = 1)$

$p(y = 1|x_1 = 1, x_2 = 1) = p(x_1 = 1|y = 1)p(x_2 = 1|y = 1)p(y = 1) = 0.5 * 0.667 * 0.6 = 0.2$

$p(y = 0|x_1 = 1, x_2 = 1) = p(x_1 = 1, x_2 = 1|y = 0)p(y = 0)/p(x_1 = 1, x_2 = 1)$

$p(y = 0|x_1 = 1, x_2 = 1) = p(x_1 = 1|y = 0)p(x_2 = 1|y = 0)p(y = 0) = 1 * 0.25 * 0.4 = 0.1$

Since $p(y = 1|x_1 = 1, x_2 = 1) > p(y = 0|x_1 = 1, x_2 = 1)$, naive Bayes classifies the test example as $y = 1$

4.3 Naive Bayes Implementation

If you run `python main.py -q 4.3` it will first load the following dataset:

1. *groupnames*: The names of four newsgroups.
2. *wordlist*: A list of words that occur in posts to these newsgroups.
3. X: A binary matrix. Each row corresponds to a post, and each column corresponds to a word from the word list. A value of 1 means that the word occured in the post.
4. y: A vector with values 1 through 4, with the value corresponding to the newsgroup that the post came from.
5. $Xvalidate$ and $yvalidate$: the word lists and newsgroup labels for additional newsgroup posts.

It will train a decision tree of depth 20 and report its validation error, followed by training a naive Bayes model.

While the *predict* function of the naive Bayes classifier is already implemented, the calculation of the variable *p_xy* is incorrect (right now, it just sets all values to 1/2). Modify this function so that *p_xy* correctly computes the conditional probability of these values based on the frequencies in the data set. Hand in your code and report the validation error that you obtain.

Naive Bayes Validation error = 0.188

4.4 Runtime of Naive Bayes for Discrete Data

Assume you have the following setup:

- The training set has n objects each with d features.
- The test set has t objects with d features.
- Each feature can have up to c discrete values (you can assume $c \leq n$).
- There are k class labels (you can assume $k \leq n$)

You can implement the training phase of a naive Bayes classifier in this setup in $O(nd)$, since you only need to do a constant amount of work for each $X(i, j)$ value. (You do not have to actually implement it in this way for the previous question, but you should think about how this could be done). What is the cost of classifying t test examples with the model? cost = $O(tdk)$