CPSC 340 Assignment 1

Name: Zhi Ting (Jessica) Yang

Student #: 33586158

CS ID: c5m0b

1 Linear Algebra Review

1.1 Basic Operations

Rubric: {reasoning:7}

Use the definitions below,

$$\alpha = 2, \quad x = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \quad y = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}, \quad z = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}, \quad A = \begin{bmatrix} 3 & 2 & 2 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix},$$

and use x_i to denote element i of vector x. Evaluate the following expressions (you do not need to show your work).

1. $\sum_{i=1}^{n} x_i y_i$ (inner product).

   $= 14$

2. $\sum_{i=1}^{n} x_i z_i$ (inner product between orthogonal vectors).

   $= 0$

3. $\alpha(x + z)$ (vector addition and scalar multiplication)

$$= \begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix}$$

4. $x^T z + \|x\|$ (inner product in matrix notation and Euclidean norm of x).

   $= \sqrt{5}$

5. Ax (matrix-vector multiplication).

$$= \begin{bmatrix} 6 \\ 5 \\ 7 \end{bmatrix}$$

6. $x^T A x$ (quadratic form).

   $= 19$

7. $A^T A$ (matrix tranpose and matrix multiplication).

$$= \begin{bmatrix} 11 & 10 & 10 \\ 10 & 14 & 10 \\ 10 & 10 & 14 \end{bmatrix}$$

1.2 Matrix Algebra Rules

Rubric: {reasoning:10}

Assume that $\{x, y, z\}$ are $n \times 1$ column vectors, $\{A, B, C\}$ are $n \times n$ real-valued matrices, 0 is the zero matrix of appropriate size, and I is the identity matrix of appropriate size. State whether each of the below is true in general (you do not need to show your work).

1. $x^T y = \sum_{i=1}^n x_i y_i$.

   True

2. $x^T x = \|x\|^2$.

   True

3. $x^T x = x x^T$.

   False

4. $(x - y)^T (x - y) = \|x\|^2 - 2x^T y + \|y\|^2$.

   True

5. $AB = BA$.

   False

6. $A^T (B + IC) = A^T B + A^T C$.

   True

7. $(A + BC)^T = A^T + B^T C^T$.

   False

8. $x^T Ay = y^T A^T x$.

   True

9. $A^T A = AA^T$ if A is a symmetric matrix.

   True

10. $A^T A = 0$ if the columns of A are orthonormal.

    False

2 Probability Review

2.1 Rules of probability

Rubric: {reasoning:6}

Answer the following questions. You do not need to show your work.

1. You are offered the opportunity to play the following game: your opponent rolls 2 regular 6-sided dice. If the difference between the two rolls is at least 3, you win \$15. Otherwise, you get nothing. What is a fair price for a ticket to play this game once? In other words, what is the expected value of playing the game?

   \$5

2. Consider two events A and B such that $\Pr(A, B) = 0$ (they are mutually exclusive). If $\Pr(A) = 0.4$ and $\Pr(A \cup B) = 0.95$, what is $\Pr(B)$? Note: $p(A, B)$ means "probability of A and B" while $p(A \cup B)$ means "probability of A or B". It may be helpful to draw a Venn diagram.

   0.55

3. Instead of assuming that A and B are mutually exclusive ($\Pr(A, B) = 0$), what is the answer to the previous question if we assume that A and B are independent?

   $\frac{0.55}{0.6}$

2.2 Bayes Rule and Conditional Probability

Rubric: {reasoning:10}

Answer the following questions. You do not need to show your work.

Suppose a drug test produces a positive result with probability 0.97 for drug users, $P(T = 1 \mid D = 1) = 0.97$. It also produces a negative result with probability 0.99 for non-drug users, $P(T = 0 \mid D = 0) = 0.99$. The probability that a random person uses the drug is 0.0001, so $P(D = 1) = 0.0001$.

1. What is the probability that a random person would test positive, $P(T = 1)$?

   0.010096

2. In the above, do most of these positive tests come from true positives or from false positives?

   Most of the positive tests come from false positives.

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

   0.009607765

4. Suppose you have given this test to a random person and it came back positive, are they likely to be a drug user?

   No, as most of the positive tests come from false positives. The chance of someone actually being a drug user when testing positive is only about 1%.

5. What is one factor you could change to make this a more useful test?

   Decreasing the probability of a positive result for non-drug users (decreasing $P(T = 1 \mid D = 0)$) would help increase the accuracy of the test.

3 Calculus Review

3.1 One-variable derivatives

Rubric: {reasoning:8}

Answer the following questions. You do not need to show your work.

1. Find the derivative of the function $f(x) = 3x^2 - 2x + 5$.

$f'(x) = 6x - 2$

2. Find the derivative of the function $f(x) = x(1 - x)$.

   $f'(x) = 1 - 2x$

3. Let $p(x) = \frac{1}{1+\exp(-x)}$ for $x \in \mathbb{R}$. Compute the derivative of the function $f(x) = x - \log(p(x))$ and simplify it by using the function $p(x)$.

   $f'(x) = p(x)$

Remember that in this course we will $\log(x)$ to mean the "natural" logarithm of x, so that $\log(\exp(1)) = 1$. Also, obseve that $p(x) = 1 - p(-x)$ for the final part.

3.2 Multi-variable derivatives

Rubric: {reasoning:5}

Compute the gradient vector $\nabla f(x)$ of each of the following functions. You do not need to show your work.

1. $f(x) = x_1^2 + \exp(x_1 + 2x_2)$ where $x \in \mathbb{R}^2$.

   $\nabla f(x) =< 2x_1 + \exp(x_1 + 2x_2), 2exp(x_1 + 2x_2) >$

2. $f(x) = \log\left(\sum_{i=1}^3 \exp(x_i)\right)$ where $x \in \mathbb{R}^3$ (simplify the gradient by defining $Z = \sum_{i=1}^3 \exp(x_i)$).

   $\nabla f(x) = \frac{1}{Z} < \exp(x_1), \exp(x_2), \exp(x_3)) >$

3. $f(x) = a^T x + b$ where $x \in \mathbb{R}^3$ and $a \in \mathbb{R}^3$ and $b \in \mathbb{R}$.

   $\nabla f(x) =< a_1, a_2, a_3 >= a$

4. $f(x) = \frac{1}{2}x^\top Ax$ where $A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$ and $x \in \mathbb{R}^2$.

   $\nabla f(x) =< 2x_1 - x_2, -x_1 + 2x_2 >= Ax$

5. $f(x) = \frac{1}{2}\|x\|^2$ where $x \in \mathbb{R}^d$.

   $\nabla f(x) =< x_1, x_2, ..., x_n >= x$

3.3 Optimization

Find the following quantities. You do not need to show your work. You can/should use your results from parts 3.1 and 3.2 as part of the process.

1. min $3x^2 - 2x + 5$, or, in words, the minimum value of the function $f(x) = 3x^2 - 2x + 5$ for $x \in \mathbb{R}$.

   $\frac{14}{3}$

2. max $x(1 - x)$ for $x \in [0, 1]$.

   $\frac{1}{4}$

3. min $x(1 - x)$ for $x \in [0, 1]$.

   0

4. arg max $x(1 - x)$ for $x \in [0, 1]$.

   $x = \frac{1}{2}$

5. min $x_1^2 + \exp(x_2)$ where $x \in [0, 1]^2$, or in other words $x_1 \in [0, 1]$ and $x_2 \in [0, 1]$.

1

6. $\arg\min x_1^2 + \exp(x_2)$ where $x \in [0,1]^2$.

   $x_1 = 0, x_2 = 0$

3.4 Derivatives of code

Rubric: {code:4}

```
def foo_grad(x):
    return 4*x**3

def bar_grad(x):
    grad = np.ones(len(x))
    for i in range(len(x)):
        for j in range(len(x)):
            if i != j:
                grad[i] = grad[i] * x[j]
    return grad
```

See `grads.py` for full code and `main.py` for tests

4 Algorithms and Data Structures Review

4.1 Trees

Rubric: {reasoning:2}

Answer the following questions. You do not need to show your work.

1. What is the minimum depth of a binary tree with 64 leaf nodes?

   6

2. What is the minimum depth of binary tree with 64 nodes (includes leaves and all other nodes)?

   6

Note: we'll use the standard convention that the leaves are not included in the depth, so a tree with depth 1 has 3 nodes with 2 leaves.

4.2 Common Runtimes

Rubric: {reasoning:4}

Answer the following questions using big-O notation You do not need to show your work.

1. What is the cost of finding the largest number in an unsorted list of n numbers?

   $O(n)$

2. What is the cost of finding the smallest element greater than 0 in a *sorted* list with n numbers.

   $O(log(n))$

3. What is the cost of finding the value associated with a key in a hash table with n numbers? (Assume the values and keys are both scalars.)

$O(1)$

4. What is the cost of computing the inner product $a^T x$, where a is $d \times 1$ and x is $d \times 1$?

   $O(d)$

5. What is the cost of computing the quadratic form $x^T A x$ when A is $d \times d$ and x is $d \times 1$.

   $O(d^2)$

4.3 Running times of code

Rubric: {reasoning:4}

Your repository contains a file named `bigO.py`, which defines several functions that take an integer argument N. For each function, state the running time as a function of N, using big-O notation. Please include your answers in your report. Do not write your answers inside `bigO.py`.

`func1` is $O(N)$

`func2` is $O(N)$

`func3` is $O(1)$

`func4` is $O(N^2)$

5 Data Exploration

5.1 Summary Statistics

Rubric: {reasoning:2}

Report the following statistics:

See code used to calculate the below at `main.py`

1. The minimum, maximum, mean, median, and mode of all values across the dataset.

   Minimum: 0.35200000000000004

   Maximum: 4.862

   Mean: 1.3246250000000002

   Median: 1.1589999999999998

   Mode: 0.77

2. The 5%, 25%, 50%, 75%, and 95% quantiles of all values across the dataset.

   5% quantile: 0.46495000000000003

   25% quantile: 0.718

   50% quantile: 1.1589999999999998

   75% quantile: 1.81325

   95% quantile: 2.6240499999999999

3. The names of the regions with the highest and lowest means, and the highest and lowest variances.

   The region with the highest mean: WtdILI

The region with the lowest mean: Pac

The region with the highest var: Mtn

The region with the lowest var: Pac

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.

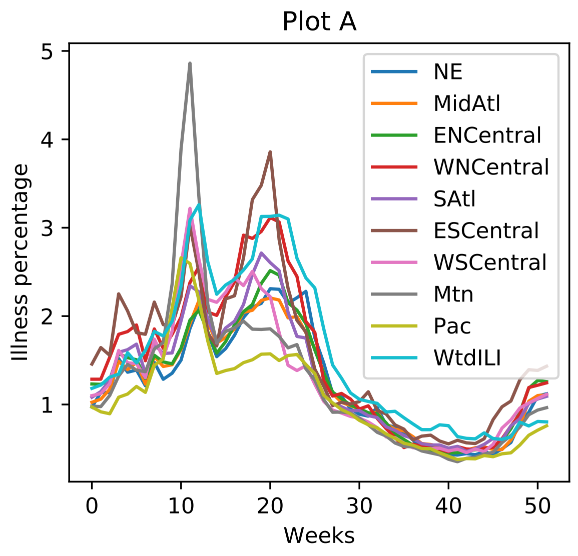
The mode is not always a reliable estimate of the most ”common” value for continuous data, as it does not take account how ”close together” values are to each other, only whether they are the exact same value. The mode was 0.77 as that particular exact value just happened to appear in the data the most times.

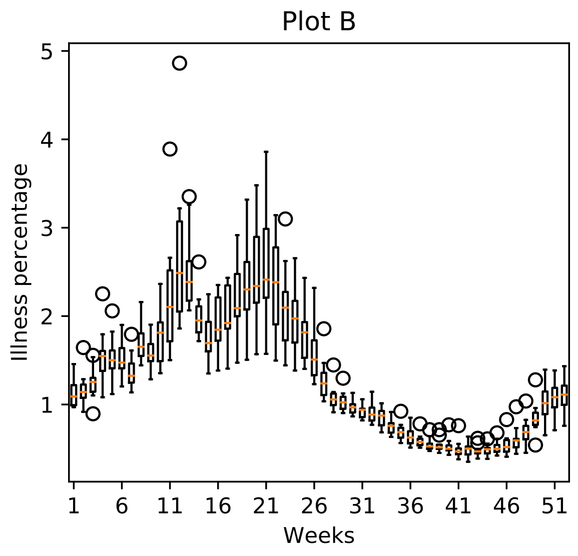
A way we could compute a more meaningful ”mode” measurement for this data is to bin the data by rounding to 1 decimal place, and then taking the mode of the new discretized data.

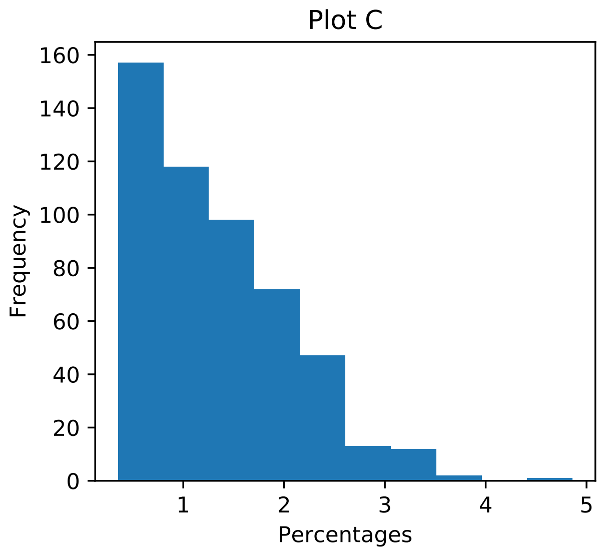
5.2 Data Visualization

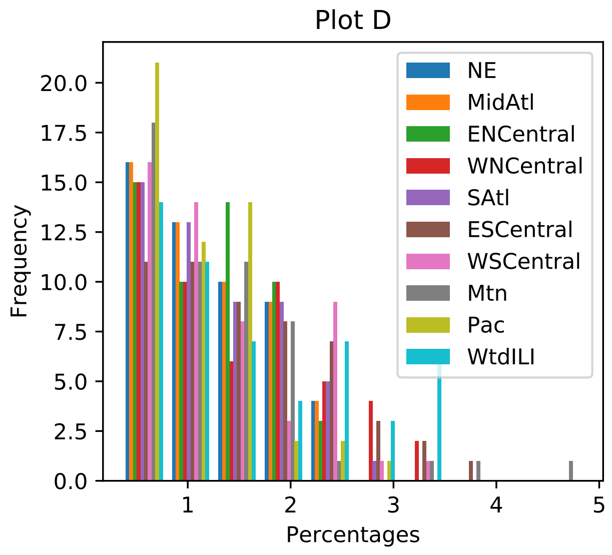
Rubric: {reasoning:3}

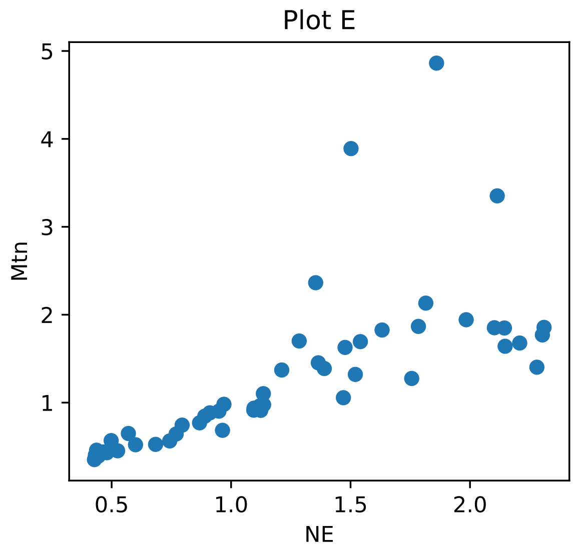
Consider the figure below.

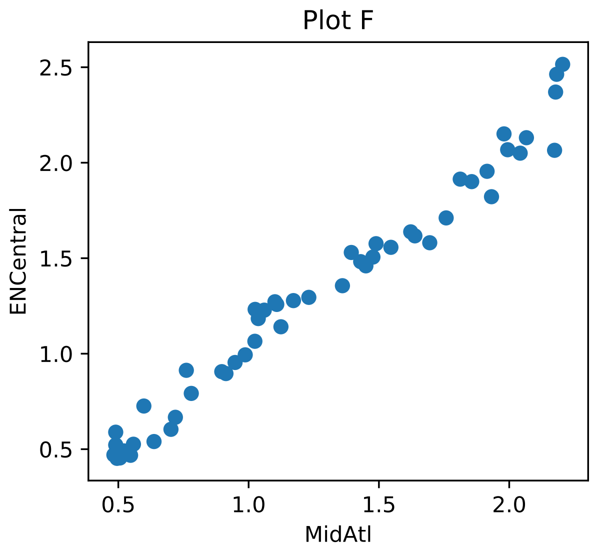

Plot A
Illness percentage
Weeks
NE
MidAtl
ENCentral
WNCentral
SAtl
ESCentral
WSCentral
Mtn
Pac
WtdILI


Plot B
Illness percentage
Weeks


Plot C
Frequency
Percentages


Plot D
Frequency
Percentages
NE
MidAtl
ENCentral
WNCentral
SAtl
ESCentral
WSCentral
Mtn
Pac
WtdILI


Plot E
Mtn
NE


Plot F
ENCentral
MidAtl

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

1. A single histogram showing the distribution of *each* column in X.
2. A histogram showing the distribution of each the values in the matrix X.
3. A boxplot grouping data by weeks, showing the distribution across regions for each week.
4. A plot showing the illness percentages over time.
5. A scatterplot between the two regions with highest correlation.
6. A scatterplot between the two regions with lowest correlation.

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.

Plot A: Description 4. It is the only regular plot, so it has to map to Description 4. It shows the illness percentages by week, so over time.

Plot B: Description 3. It is the only boxplot, so it has to map to Description 3. By grouping data by weeks against illness percentage, the distribution over regions for each week is shown via each box.

Plot C: Description 2. It is a histogram, so it has to map to either Description 1 or 2. It is Description 2 as it does not show the distribution of each region, so it is not Description 1.

Plot D: Description 1. See above, this is a histogram that shows the distribution of each region, and each region is a column in X.

Plot E: Description 6. It is a scatterplot, so it has to map to either Description 5 or 6. The correlation is lower than Plot F, so it maps to Description 6.

Plot F: Description 5. See above, here, the correlation is high, so it maps to Description 5.

6 Decision Trees

6.1 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?

Yes, categorical features, as threshold-based splits wouldn't be possible.

6.2 Decision Stump Implementation

Rubric: {code:3}

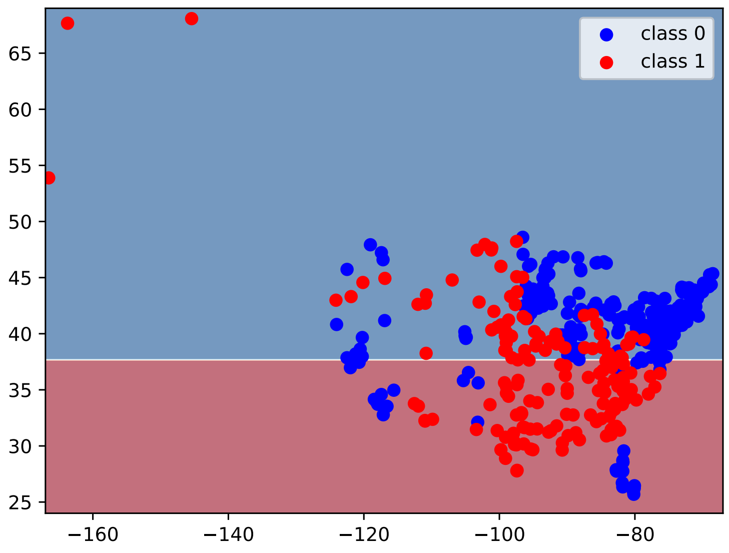
The file `decision_stump.py` contains the class `DecisionStumpEquality` which finds the best decision stump using the equality rule and then makes predictions using that rule. Instead of discretizing the data and using a rule based on testing an equality for a single feature, we want to check whether a feature is above or below a threshold and split the data accordingly (this is a more sane approach, which we discussed in class). Create a `DecisionStumpErrorRate` class to do this, and report the updated error you obtain by using inequalities instead of discretizing and testing equality. Also submit the generated figure of the classification boundary.

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

```
class DecisionStumpErrorRate:

    def __init__(self):
        pass

    # Copied and then modified from DecisionStumpEquality.fit
    def fit(self, X, y):
        # X is N by D, N rows and D columns
        N, D = X.shape

        # Get an array with the number of 0's, number of 1's, etc.
        # count = occurrence of each unique label
        count = np.bincount(y)

        # Get the index of the largest value in count.
        # Thus, y_mode is the mode (most popular value) of y
        y_mode = np.argmax(count)

        # Initialize prediction to y_mode
        self.splitSat = y_mode
        self.splitNot = None
        # Feature we split on
        self.splitVariable = None
        # Threshold we split with
        self.splitValue = None

        # If all the labels are the same, no need to split further
        # Always predict y_mode
        if np.unique(y).size <= 1:
            return

        # Initialize minError as number of times we predict wrong if we always predict y_mode
        # This is computing the error if using "baseline" rule, number of times y_i does not equ
        minError = np.sum(y != y_mode)

        # Loop over features looking for the best split
        # No need to round as we are doing a threshold-based split
        # For feature d of D features
        for d in range(D):
            # For example n of N examples
            for n in range(N):
                # Choose value to equate to
                # This is a threshold
                value = X[n, d]

                # Find most likely class for each split
                # Change from equality-based split to threshold-based split
                # Set y_sat to most common label of examples satisfying rule
                y_sat = utils.mode(y[X[:,d] > value])
                # Set y_not to most common label of examples not satisfying rule
                y_not = utils.mode(y[X[:,d] <= value])

                # Make predictions
                y_pred = y_sat * np.ones(N)
                y_pred[X[:, d] <= value] = y_not
                # Compute error
                errors = np.sum(y_pred != y)

                # Compare to minimum error so far
                if errors < minError:
                    # This is the lowest error, store this value
                    minError = errors
                    self.splitVariable = d
                    self.splitValue = value
                    self.splitSat = y_sat
                    self.splitNot = y_not
```

The updated error obtained by using inequalities is 0.253.

Generated figure:



6.3 Decision Stump Info Gain Implementation

Rubric: {code:3}

In `decision_stump.py`, create a `DecisionStumpInfoGain` class that fits using the information gain criterion discussed in lecture. Report the updated error you obtain, and submit the classification boundary figure.

Notice how the error rate changed. Are you surprised? If so, hang on until the end of this question!

Note: even though this data set only has 2 classes (red and blue), your implementation should work for any number of classes, just like `DecisionStumpEquality` and `DecisionStumpErrorRate`.

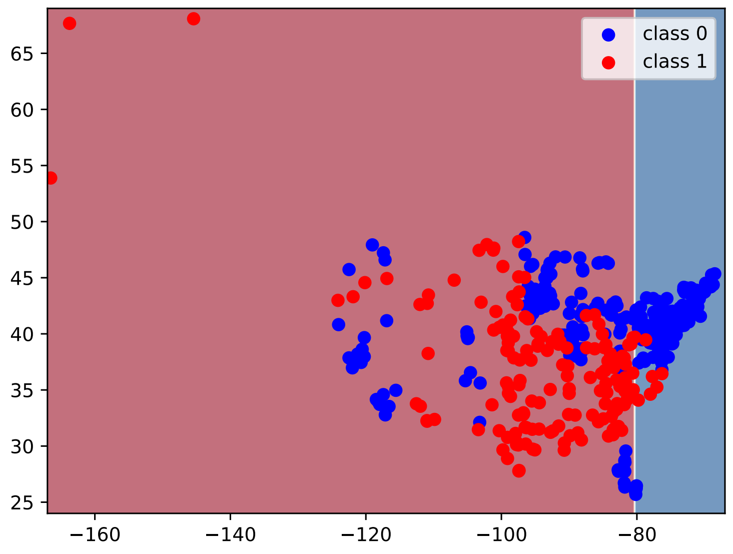
Hint: take a look at the documentation for `np.bincount`, at `https://docs.scipy.org/doc/numpy/reference/generated/numpy.bincount.html`. The `minlength` argument comes in handy here to deal with a tricky corner case: when you consider a split, you might not have any examples of a certain class, like class 1, going to one side of the split. Thus, when you call `np.bincount`, you'll get a shorter array by default, which is not what you want. Setting `minlength` to the number of classes solves this problem.

```
# This is not required, but one way to simplify the code is
# to have this class inherit from DecisionStumpErrorRate.
# Which methods (init, fit, predict) do you need to overwrite?

# Only need to overwrite fit as init and predict are the same as DecisionStumpErrorRate
class DecisionStumpInfoGain(DecisionStumpErrorRate):

    # Copied and then modified from DecisionStumpErrorRate.fit
    def fit(self, X, y):
        # X is N by D, N rows and D columns
        N, D = X.shape

        # Get an array with the number of 0's, number of 1's, etc.
        # count = occurrence of each unique label
        count = np.bincount(y)

        # Get the index of the largest value in count.
        # Thus, y_mode is the mode (most popular value) of y
        y_mode = np.argmax(count)

        # Initialize prediction to y_mode
        self.splitSat = y_mode
        self.splitNot = None
        # Feature we split on
        self.splitVariable = None
        # Threshold we split with
        self.splitValue = None

        # Initialize info gain to 0, this is the baseline rule ("do nothing")
        maxInfoGain = 0
        # Get an array with the probabilities of the occurrence of each unique label
        p = count/np.sum(count)
        # Find entropy of labels before split
        preSplitEntropy = entropy(p)

        # If all the labels are the same, no need to split further
        # Always predict y_mode
        if np.unique(y).size <= 1:
            return

        # Loop over features looking for the best split to maximize info gain
        # For feature d of D features
        for d in range(D):
            # For example n of N examples
            for n in range(N):
                # Choose value to equate to
                # This is a threshold
                value = X[n, d]

                examples_satisfying_rule = X[:,d] > value
                # Calculate number of examples satisfying rule / number of examples to get proba
                y_prob = np.sum(examples_satisfying_rule) / N
                # Calculate probability of "no" side
                n_prob = 1 - y_prob
```

```
# Count number of labels satisfying rule
y_labels = y[examples_satisfying_rule]
y_count = np.bincount(y_labels, minlength = len(count))
# Calculate number of labels not satisfying rule
n_count = count - y_count

# Calculate entropy of labels for examples satisfying rule
y_p = y_count / np.sum(y_count)
y_entropy = entropy(y_p)

# Calculate entropy of labels for examples not satisfying rule
n_p = n_count / np.sum(n_count)
n_entropy = entropy(n_p)

# Calculate info gain from split
splitInfoGain = preSplitEntropy - (y_prob * y_entropy) - (n_prob * n_entropy)

# Store split if info gain is greater than best found so far
if splitInfoGain > maxInfoGain:
    self.splitSat = np.argmax(y_count)
    self.splitNot = np.argmax(n_count)
    self.splitVariable = d
    self.splitValue = value
    # Update new best info gain found so far
    maxInfoGain = splitInfoGain
```

The updated error obtained by using info gain is 0.325.

Generated figure:



6.4 Constructing Decision Trees

Rubric: {code:2}

Once your DecisionStumpInfoGain class is finished, running `python main.py -q 6.4` 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 splits from the fitted depth-2 decision tree, write a hard-coded version of the `predict` function that classifies one example using simple if/else statements (see the Decision Trees lecture). Save your code in a new file called `simple_decision.py` (in the `code` directory) and make sure you include the text of this file in your LaTeX report.

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. You should just hard-code the split values directly into the code. Only the `predict` function is needed.

Hint: if you plot the decision boundary you can do a visual sanity check to see if your code is consistent with the plot.

```
# Predict for example n

# n[0] is the longitude
# n[1] is the latitude

# 0 is blue state
# 1 is red state
def predict(n):
        if n[0] > -80.305106:
                if n[1] > 36.453576:
                        return 0
                else:
                        return 0
        else:
                if n[1] > 37.669007:
                        return 0
                else:
                        return 1
```

6.5 Decision Tree Training Error

Rubric: {reasoning:2}

Running `python main.py -q 6.5` fits decision trees of different depths using the following different implementations:

1. A decision tree using `DecisionStump`
2. A decision tree using `DecisionStumpInfoGain`
3. The `DecisionTreeClassifier` from the popular Python ML library *scikit-learn*

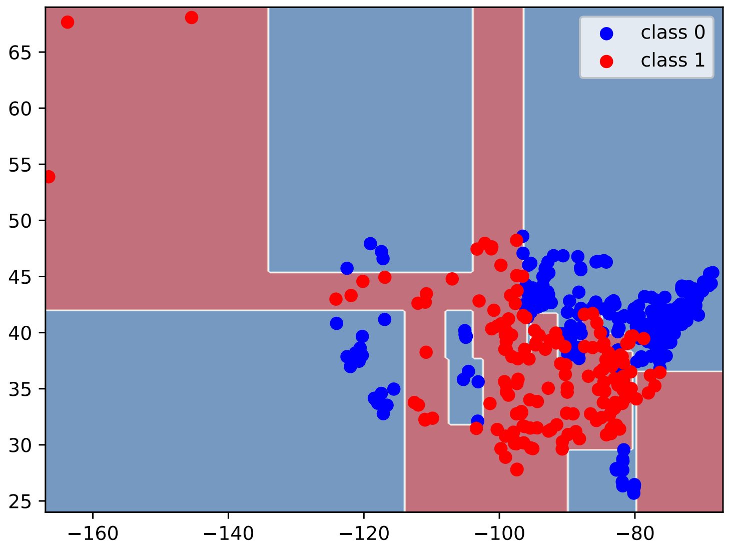
Run the code and look at the figure. Describe what you observe. Can you explain the results? Why is approach (1) so disappointing? Also, submit a classification boundary plot of the model with the lowest training error.

Note: we set the `random_state` because sklearn's `DecisionTreeClassifier` is non-deterministic. This is probably 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.

`DecisionTreeErrorRate`'s lowest classification error is about 0.10 which occurs at the 4th split, and further splits do not decrease the classification error. However, both `DecisionTreeInfoGain` and `DecisionTreeClassifier` from *scikit-learn* are able to obtain a classification error of 0 given a deep enough tree. `DecisionTreeErrorRate` cannot obtain a lower classification score after the 4th split as for every leaf, there is no rule that exists that would be better than just predicting the mode.

Classification boundary plot of the model with the lowest training error, using `DecisionTreeClassifier` with depth 10:



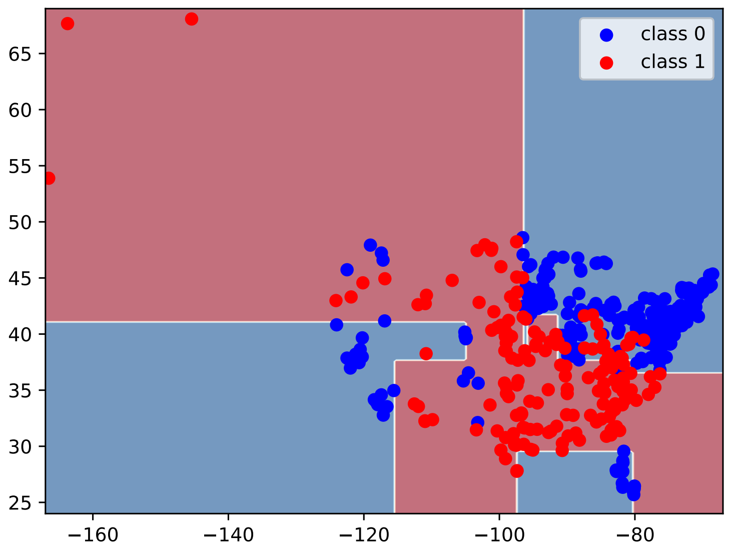
6.6 Comparing implementations

Rubric: {reasoning:2}

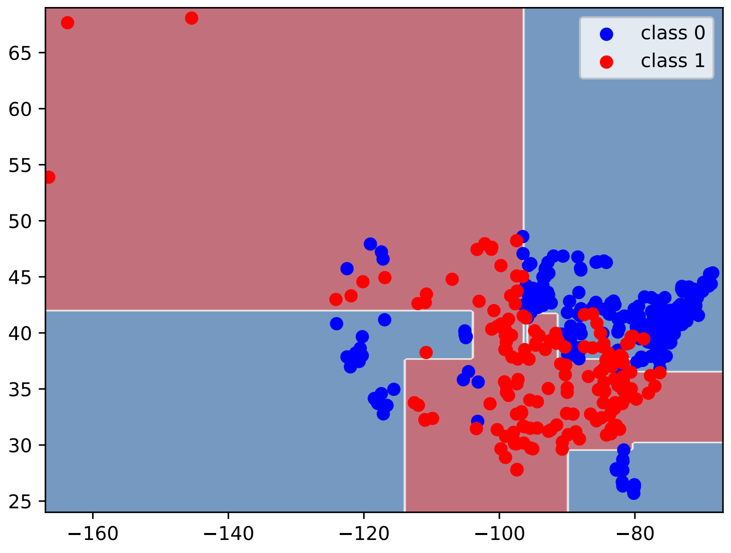
In the previous section you compared different implementations of a machine learning algorithm. Let's say that two approaches produce the exact same curve of classification error rate vs. tree depth. Does this conclusively demonstrate that the two implementations are the same? If so, why? If not, what other experiment might you perform to build confidence that the implementations are probably equivalent?

No, for example, the implementation of `DecisionTreeInfoGain` is not the same as sklearn's `DecisionTreeClassifier`.

The classification boundary plots of the `DecisionTreeInfoGain` vs. `DecisionTreeClassifier` at a depth of 5 is not the same:



vs.



Though they both have the same error rate at depth 5, they classify examples differently.

An experiment I might perform to build confidence that the implementations are probably equivalent is to see if they classify examples the same. If they both make the same predictions, the implementations are probably the same.

6.7 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.

The tree is of depth m. There are n examples at the root, and at each iteration, we split the n examples across each stump we generate, so there are n examples at each depth. The cost to build each level of the decision tree is $O(nd \log n)$, the same cost for fitting the root decision stump, as both operations have to look at the same number of examples. The cost for fitting the whole tree of depth m would be $O(mnd \log n)$.

7 Appendix

7.1 Full Code for 3.4

grads.py

```python
import numpy as np

def example(x):
    return np.sum(x**2)

def example_grad(x):
    return 2*x

def foo(x):
    result = 1
    λ = 4 # this is here to make sure you're using Python 3
    for x_i in x:
        result += x_i**λ
    return result

def foo_grad(x):
    return 4*x**3

def bar(x):
    return np.prod(x)

def bar_grad(x):
    grad = np.ones(len(x))
    for i in range(len(x)):
        for j in range(len(x)):
            if i != j:
                grad[i] = grad[i] * x[j]
    return grad
```

7.2 Full main.py Code

main.py

```python
# standard Python imports
import os
import argparse
import time
import pickle

# 3rd party libraries
import numpy as np                              # this comes with Anaconda
import pandas as pd                             # this comes with Anaconda
import matplotlib.pyplot as plt                 # this comes with Anaconda
from scipy.optimize import approx_fprime        # this comes with Anaconda
from sklearn.tree import DecisionTreeClassifier # if using Anaconda, install with `conda install
""" NOTE:
Python is nice, but it's not perfect. One horrible thing about Python is that a
package might use different names for installation and importing. For example,
seeing code with `import sklearn` you might sensibly try to install the package
with `conda install sklearn` or `pip install sklearn`. But, in fact, the actual
way to install it is `conda install scikit-learn` or `pip install scikit-learn`.
Wouldn't it be lovely if the same name was used in both places, instead of
`sklearn` and then `scikit-learn`? Please be aware of this annoying feature.
"""

# CPSC 340 code
import utils
import grads
from decision_stump import DecisionStumpEquality, DecisionStumpErrorRate, DecisionStumpInfoGain
from decision_tree import DecisionTree


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-q','--question', required=True)

    io_args = parser.parse_args()
    question = io_args.question

    if question == "3.4":
        # Here is some code to test your answers to Q3.4
        # Below we test out example_grad using scipy.optimize.approx_fprime, which approximates
        # if you want, you can use this to test out your foo_grad and bar_grad

        def check_grad(fun, grad):
            x0 = np.random.rand(5) # take a random x-vector just for testing
            diff = approx_fprime(x0, fun, 1e-4)  # don't worry about the 1e-4 for now
            print("\n** %s **" % fun.__name__)
            print("My gradient     : %s" % grad(x0))
            print("Scipy's gradient: %s" % diff)

        check_grad(grads.example, grads.example_grad)
        check_grad(grads.foo, grads.foo_grad)
        check_grad(grads.bar, grads.bar_grad)


    elif question == "5.1":
        # Load the fluTrends dataset
        df = pd.read_csv(os.path.join('..','data','fluTrends.csv'))
        X = df.values
        names = df.columns.values
        # 5.1.1
        print("\n\n5.1.1:")
        print("Minimum: %s" % np.amin(X))
        print("Maximum: %s" % np.amax(X))
        print("Mean: %s" % np.mean(X))
        print("Median: %s" % np.median(X))
        print("Mode: %s" % utils.mode(X))

        # 5.1.2
        print("\n\n5.1.2:")
        print("5%% quantile: %s" % np.quantile(X, 0.05))
        print("25%% quantile: %s" % np.quantile(X, 0.25))
        print("50%% quantile: %s" % np.quantile(X, 0.5))
        print("75%% quantile: %s" % np.quantile(X, 0.75))
        print("95%% quantile: %s" % np.quantile(X, 0.95))

        # 5.1.3
        print("\n\n5.1.3:")

        meansByRegion = np.mean(X, axis = 0)
        varsByRegion = np.var(X, axis = 0)
        print("The region with the highest mean: %s" % names[np.argmax(meansByRegion)])
        print("The region with the lowest mean: %s" % names[np.argmin(meansByRegion)])
        print("The region with the highest var: %s" % names[np.argmax(varsByRegion)])
        print("The region with the lowest var: %s" % names[np.argmin(varsByRegion)])

    elif question == "6":
        # 1Load citiesSmall dataset
        with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
            dataset = pickle.load(f)

        X = dataset["X"]
        y = dataset["y"]

        # 2Evaluate majority predictor model
        y_pred = np.zeros(y.size) + utils.mode(y)

        error = np.mean(y_pred != y)
        print("Mode predictor error: %.3f" % error)

        # 3Evaluate decision stump
        model = DecisionStumpEquality()
        model.fit(X, y)
        y_pred = model.predict(X)

        error = np.mean(y_pred != y)
        print("Decision Stump with equality rule error: %.3f"
              % error)

        # Plot result
        utils.plotClassifier(model, X, y)
        fname = os.path.join("..", "figs", "q6_decisionBoundary.pdf")
        plt.savefig(fname)
        print("\nFigure saved as '%s'" % fname)
```

```
elif question == "6.2":
    # Load citiesSmall dataset
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X = dataset["X"]
    y = dataset["y"]

    # Evaluate decision stump
    model = DecisionStumpErrorRate()
    model.fit(X, y)
    y_pred = model.predict(X)

    error = np.mean(y_pred != y)
    print("Decision Stump with inequality rule error: %.3f" % error)

    # Plot result
    utils.plotClassifier(model, X, y)

    fname = os.path.join("..", "figs", "q6_2_decisionBoundary.pdf")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)

elif question == "6.3":
    # 1. Load citiesSmall dataset
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X = dataset["X"]
    y = dataset["y"]

    # 3. Evaluate decision stump
    model = DecisionStumpInfoGain()
    model.fit(X, y)
    y_pred = model.predict(X)

    error = np.mean(y_pred != y)
    print("Decision Stump with info gain rule error: %.3f" % error)

    # PLOT RESULT
    utils.plotClassifier(model, X, y)

    fname = os.path.join("..", "figs", "q6_3_decisionBoundary.pdf")
    plt.savefig(fname)
    print("\nFigure saved as '%s'" % fname)

elif question == "6.4":
    with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
        dataset = pickle.load(f)

    X = dataset["X"]
    y = dataset["y"]

    model = DecisionTree(max_depth=2,stump_class=DecisionStumpInfoGain)
    model.fit(X, y)
```

```
        y_pred = model.predict(X)
        error = np.mean(y_pred != y)

        print("Error: %.3f" % error)

        utils.plotClassifier(model, X, y)

        fname = os.path.join("..", "figs", "q6_4_decisionBoundary.pdf")
        plt.savefig(fname)
        print("\nFigure saved as '%s'" % fname)


    elif question == "6.5":
        with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
            dataset = pickle.load(f)

        X = dataset["X"]
        y = dataset["y"]
        print("n = %d" % X.shape[0])

        depths = np.arange(1,15) # depths to try




        t = time.time()
        my_tree_errors = np.zeros(depths.size)
        for i, max_depth in enumerate(depths):
            model = DecisionTree(max_depth=max_depth)
            model.fit(X, y)
            y_pred = model.predict(X)
            my_tree_errors[i] = np.mean(y_pred != y)
        print("Our decision tree with DecisionStumpErrorRate took %f seconds" % (time.time()-t))

        plt.plot(depths, my_tree_errors, label="errorrate")


        t = time.time()
        my_tree_errors_infogain = np.zeros(depths.size)
        for i, max_depth in enumerate(depths):
            model = DecisionTree(max_depth=max_depth,stump_class=DecisionStumpInfoGain)
            model.fit(X, y)
            y_pred = model.predict(X)
            my_tree_errors_infogain[i] = np.mean(y_pred != y)
        print("Our decision tree with DecisionStumpInfoGain took %f seconds" % (time.time()-t))

        plt.plot(depths, my_tree_errors_infogain, label="infogain")

        t = time.time()
        sklearn_tree_errors = np.zeros(depths.size)
        for i, max_depth in enumerate(depths):
            model = DecisionTreeClassifier(max_depth=max_depth, criterion='entropy', random_stat
            model.fit(X, y)
            y_pred = model.predict(X)
            sklearn_tree_errors[i] = np.mean(y_pred != y)
        print("scikit-learn's decision tree took %f seconds" % (time.time()-t))

        plt.plot(depths, sklearn_tree_errors, label="sklearn", linestyle=":", linewidth=3)

        plt.xlabel("Depth of tree")
        plt.ylabel("Classification error")
        plt.legend()
        fname = os.path.join("..", "figs", "q6_5_tree_errors.pdf")
        plt.savefig(fname)

        # Classification boundary plot of the model with the lowest training error
        model = DecisionTreeClassifier(max_depth=10, criterion='entropy', random_state=1)
        model.fit(X, y)

        y_pred = model.predict(X)
        error = np.mean(y_pred != y)

        print("Lowest training error: %.3f" % error)

        utils.plotClassifier(model, X, y)

        fname = os.path.join("..", "figs", "q6_5_lowestTrainingError.pdf")
        plt.savefig(fname)
        print("\nFigure saved as '%s'" % fname)

    elif question == "6.6":
        with open(os.path.join('..','data','citiesSmall.pkl'), 'rb') as f:
            dataset = pickle.load(f)

        X = dataset["X"]
        y = dataset["y"]

        # Classification boundary plot of DecisionTreeInfoGain at depth 5
        model = DecisionTree(max_depth=5,stump_class=DecisionStumpInfoGain)
        model.fit(X, y)

        y_pred = model.predict(X)
        error = np.mean(y_pred != y)

        utils.plotClassifier(model, X, y)

        fname = os.path.join("..", "figs", "q6_6_infoGain.pdf")
        plt.savefig(fname)
        print("\nFigure saved as '%s'" % fname)

        # Classification boundary plot of DecisionTreeClassifier at depth 5
        model = DecisionTreeClassifier(max_depth=5, criterion='entropy', random_state=1)
        model.fit(X, y)

        y_pred = model.predict(X)
        error = np.mean(y_pred != y)

        utils.plotClassifier(model, X, y)

        fname = os.path.join("..", "figs", "q6_6_classifier.pdf")
        plt.savefig(fname)
        print("\nFigure saved as '%s'" % fname)
    else:
        print("No code to run for question", question)
```