CPSC 340 Assignment 1

Yihao Zhang

January 2019

1 Linear Algebra Review

1.1 Basic Operations

Use the definitions below,

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

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

1. $\sum_{i=1}^{n} x_i y_i = 14$

2. $\sum_{i=1}^{n} x_i z_i = 0$

3. $\alpha(x + z) = \begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix}$

4. $x^T z + \|x\| = \sqrt{5}$

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

6. $x^T A x = 19$

7. $A^T A = \begin{bmatrix} 11 & 10 & 10 \\ 10 & 14 & 10 \\ 10 & 10 & 14 \end{bmatrix}$

1.2 Matrix Algebra Rules

Assume that x, y, z are n*1 column vectors, A, B, C are n*n real-valued matrices, 0 is the zero matrix of appropriate size, and I is the identity matrix of appropriate size. Stat whether each of the below is true in general.

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. $(x-y)^T(x-y) = \|x\|^2 - 2x^T y + \|y\|^2$ False
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. $x^T A y = y^T A^T x$ True
9. $A^T A = A A^T$ 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

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?
   $price = 6/36 * 15 = \$2.5$

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)?
   $Pr(B) = 0.95 - 0.4 = 0.55$

3. Instead of assuming that A and B are mutually exclusive, what is the answer to the previous question if we assume that A and B are independent?
   $Pr(B) = Pr(A \cup B) + Pr(A \cap B) - Pr(A) = 0.55 + Pr(A \cap B) = 0.55 + Pr(A) * Pr(B)$
   $Pr(B) = \frac{11}{12}$

2.2 Bayes Rule and Conditional Probability

Suppose a drug test produces a positive result with probability 0.97 for drug users, $P(T = 1|D = 1) = 0.97$. 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 0.0001, so $P(D = 1) = 0.0001$.

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

$$\begin{aligned} P(T = 1) &= P(T = 1|D = 1) * P(D = 1) + P(T = 1|D = 0) * P(D = 0) \\ &= 0.010096 \end{aligned}$$

2. In the above, do most of these positive tests come from true positive or false positive?
   Most from false positive P(T=1—D=0)*P(D=0)

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

$$\begin{aligned} P(D = 1|T = 1) &= \frac{P(D = 1 \cap T = 1)}{P(T = 1)} \\ &= \frac{P(T = 1|D = 1) * P(D = 1)}{P(T = 1)} \\ &= \frac{0.97 * 0.0001}{0.010096} \\ &= 0.00961 \end{aligned}$$

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 they are likely to be a non-drug user

5. What is one factor could change to make this a more useful test?
   Most of the results are from non-drug users because the probability that a random person uses the drug is 0.001. Increasing such probability will make the test more useful.

3 Calculus Review

3.1 One-variable derivatives

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

$$\begin{aligned} f'(x) &= \frac{-1}{(e^x+1)^2} \\ &= -(1-p(x))^2 \end{aligned}$$

3.2 Multi-variable derivatives

1. $f(x) = x_1^2 + exp(x_1 + 2x_2) where x \in R^2$
   $\nabla f(x) = \begin{bmatrix} 2x_1 + e^{x_1} \\ 2e^{x_1+2x_2} \end{bmatrix}$

2. $f(x) = log(\sum_{i=1}^{3} exp(x_i)) where x \in R^3$
   $\nabla f(x) = \frac{1}{Z} * \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ e^{x_3} \end{bmatrix}$

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

4. $f(x) = \frac{1}{2} x^T A x where A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} and x \in R^2$
   $\nabla f(x) = \begin{bmatrix} 2x_1 - x_2 \\ 2x_2 - x_1 \end{bmatrix}$

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

3.3 Optimization

1. $min 3x^2 - 2x + 5$ or in words, the minimum value of the function $f(x) = 3x^2 - 2x + 5 for x \in R$
   $min = \frac{14}{3}$

2. $max x(1-x) for x \in [0,1]$
   $max = \frac{1}{4}$

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

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

5. $minx_1^2 + exp(x_2)wherex \in [0,1]^2, orinotherwordsx_1 \in [0,1]andx_2 \in [0,1]$
   $min = 1$

6. $argminx_1^2 + exp(x_2)wherex \in [0,1]^2$
   $argmin = 0$

3.4 Derivatives of code

The homework zip file contains a file named grads.py which defines several Python functions that take in an input variable x, which we assume to be a 1-d array (in math terms, a vector). It also includes (blank) functions that return the corresponding gradients. For each function, write code that computes the gradient of the function in Python. You should do this directly in grads.py; no need to make a fresh copy of the file. However, per the homework instructions, you should include it in the latex report as described so that the TA can assess it easily. When finished, you can run python main.py -q 3.4 to test out your code.

```
import numpy as np

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


def example_grad(x):
    return 2*x

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

def foo_grad(x):
    lamda = 4
    res = lamda * x ** (lamda - 1)
    return res

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

def bar_grad(x):
    return np.prod(x)/x
```

4 Algorithms and Data Structures Review

4.1 Trees

1. What is the minimum depth of a binary tree with 64 leaf nodes?
   depth of 6

2. What is the minimum depth of binary tree with 64 nodes (includes leaves and all other nodes)?
   depth of 6

4.2 Common Runtimes

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

2. What is the cost finding smallest element greater than 0 in a sorted list with n numbers?
   $O(logn)$

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×1 and x is d×1?
   $O(d)$

5. What is the cost of computing the quadratic form $x^T Ax$ when A is d×d and x is d×1.
   $O(d^2)$

4.3 Running time of code

1.
```
def func1(N):
    for i in range(N):
        print("Hello!")
```
   O(N)

2.
```
def func2(N):
    x = np.zeros(N)
    x += 1000
    return x
```
   O(N)

3.
```
def func3(N):
    x = np.zeros(1000)
    x = x * N
    return x
```

O(1)

4.
```
def func4(N):
    x = 0
    for i in range(N):
        for j in range(i,N):
            x += (i*j)
    return x
```

$O(N^2)$

5 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().

5.1 Summary Statistics

1. The minimum, maximum, mean, median, and mode of all values across the dataset.
   The minimum is: 0.352
   The maximum is: 4.862
   The mean is: 1.324625
   The median is: 1.159
   The mode is 0.77

2. The 5%, 25%, 50%, 75%, and 95% quantiles of all values across the dataset.
   The 5% quantile is 0.46495
   The 25% quantile is 0.718
   The 50% quantile is 1.159
   The 75% quantile is 1.81325
   The 95% quantile is 2.62405

3. The names of the regions with the highest and lowest means, and the highest and lowest variances.
   The region WtdILI has the highest mean of 1.5669615384615385
   The region Pac has the lowest mean of 1.0632115384615384

```
The region Mtn has the highest variance of 0.7834400188609467
The region Pac has the lowest mean of 0.3158458206360947
```

Code for Question 5.1

```python
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

    # YOUR CODE HERE
    # 1. The minimum, maximum, mean, median,
    # and mode of all values across the dataset.
    print("The minimum is: %s" % np.ndarray.min(X))
    print("The maximum is: %s" % np.ndarray.max(X))
    print("The mean is: %s" % np.ndarray.mean(X))
    print("The median is: %s" % np.median(X))
    print("The mode is %s" % utils.mode(X))

    # 2. The 5%, 25%, 50%, 75%, and 95% quantiles
    # of all values across the dataset.
    print("The 5%% quantile is %s" % np.percentile(X, 5))
    print("The 25%% quantile is %s" % np.percentile(X, 25))
    print("The 50%% quantile is %s" % np.percentile(X, 50))
    print("The 75%% quantile is %s" % np.percentile(X, 75))
    print("The 95%% quantile is %s" % np.percentile(X, 95))

    # 3. The names of the regions with the highest and lowest means,
    # and the highest and lowest variances.
    regionMean = np.mean(X, axis=0)
    regionVar = np.var(X, axis=0)
    print("The region {} has the highest mean of {} "
          .format(names[np.argmax(regionMean)],np.max(regionMean)))
    print("The region {} has the lowest mean of {} "
          .format(names[np.argmin(regionMean)],np.min(regionMean)))
    print("The region {} has the highest variance of {} "
          .format(names[np.argmax(regionVar)],np.max(regionVar)))
    print("The region {} has the lowest mean of {} "
          .format(names[np.argmin(regionVar)],np.min(regionVar)))
```

5.2 Data Visualization

Consider the figure below.

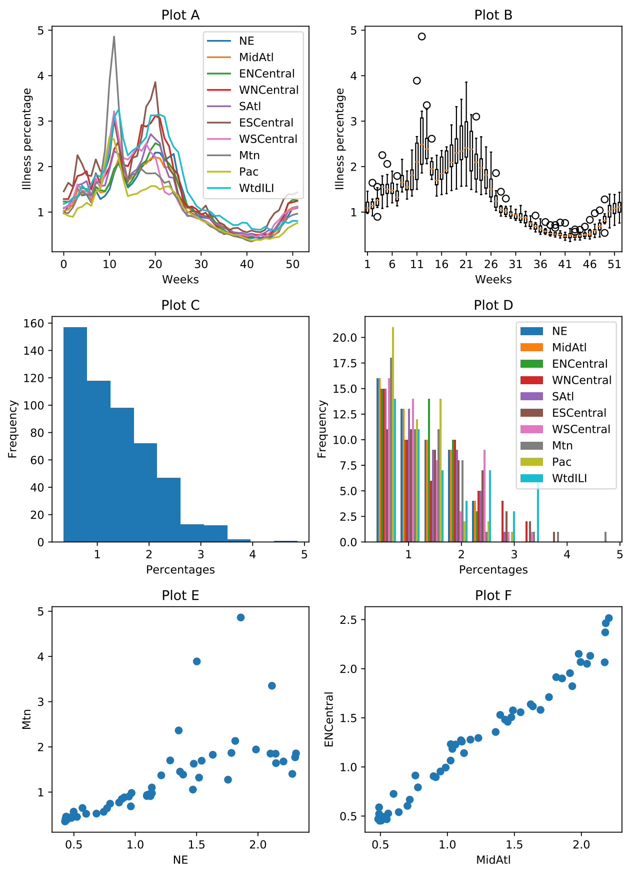

Figure 1: Unlabeled visualization

1. A single histogram showing the distribution of each column in X.
   Plot D since the columns in X are regions and can be found on the plot.

2. A histogram showing the distribution of each the values in the matrix X.
   Plot C shows the distribution of each values in matrix X since values range from 0.352 to 4.862 and it also shows the frequency for values in this range in y axis.

3. A boxplot grouping data by weeks, showing the distribution across regions for each week.
   Plot A shows illness percentage for each region by legends over time (weeks).

4. A plot showing the illness percentages over time.
   Plot B shows the illeness percentages over time without dividing data into different regions.

5. A scatterplot between the two regions with highest correlation.
   Plot F shows high correlation between MidAtl and ENCentral, the correlation is approximately y=x

6. A scatterplot between the two regions with lowest correlation.
   Plot E shows two regions with lowest correlation. If the correlation line is drawn, there are several outliers far away from the line.

6 Decision Trees

6.1 Splitting rule

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?
With categorical features, it makes more sense to use an equality-based splitting rule.

6.2 Decision Stump Implementation

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 modifi- cations from there.
Decision Stump with inequality rule error: 0.253
For code implementation, please refer to DecisionStumpErrorRate class in Appendix A

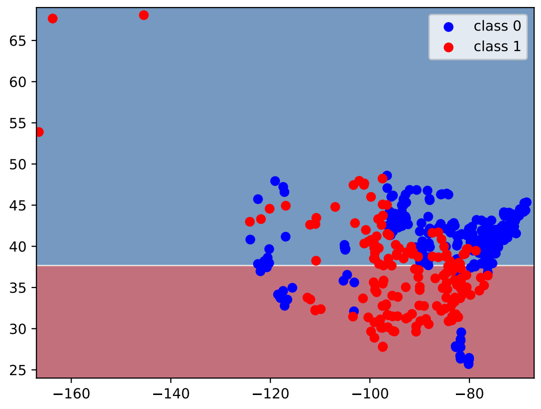

Figure 2: Classification boundary plot of inequality with depth of 1

6.3 Decision Stump Info Gain Implementation

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 ar- gument 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.

Decision Stump with info gain rule error: 0.325

For code implementation, please refer to DecisionStumpInfoGain class in Appendix A

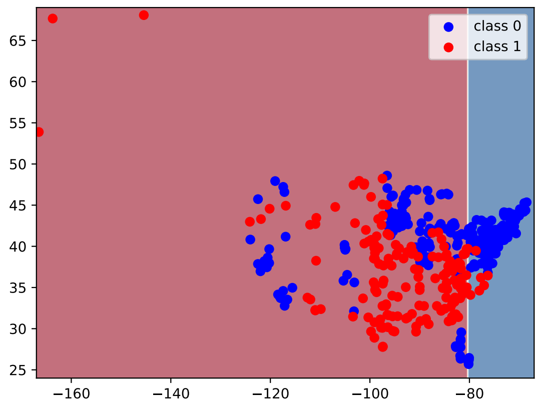

Figure 3: Classification boundary plot of infoGain with depth of 1

6.4 Constructing Decision Trees

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.

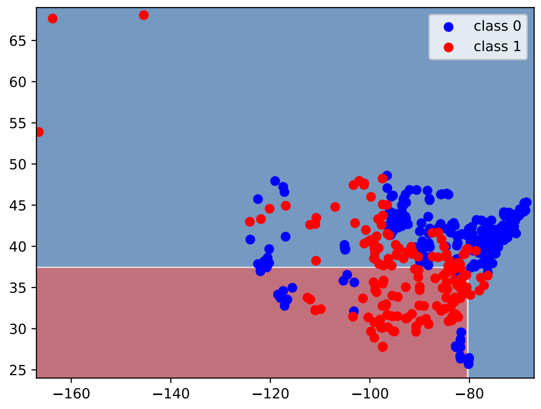

Figure 4: Classification boundary plot of infoGain with depth of 2

```
import numpy as np

def predict(X):
    N, D = X.shape
    y = np.zeros(N)

    for n in range(N):
        if X[n, 0] > -80.305106:
            if X[n, 1] > 36.453576:
                y[n] = 0
            else:
                y[n] = 0
        else:
            if X[n, 1] > 37.669007:
                y[n] = 0
            else:
                y[n] = 1

    return y
```

6.5 Decision Tree Training Error

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(n2d) decision stump learning algorithm and sklearn's implementation presumably uses the faster O(ndlogn) decision stump learning algorithm that we discussed in lecture.

The classification error vs. depth of the tree is shown in the following figure:

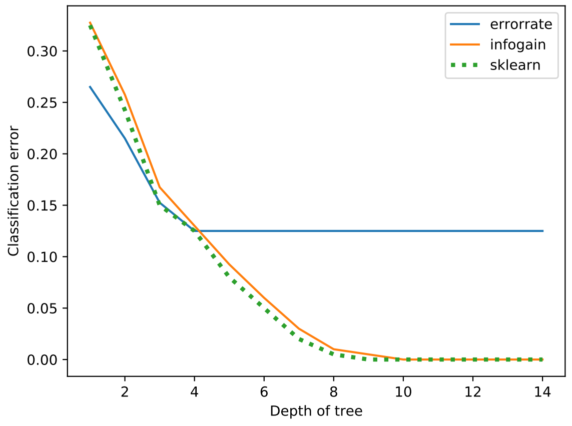

Figure 5: The classification error vs. depth of the tree

The reason why approach 1 is so disappointing is that after depth of 4, increasing depth won't lower the error rate any more and the lowest error rate is around 0.12 for this approach. Approach 2 and approach 3 yield much lower error rate compared to approach 1.

I also generated the classification boundary plot of lowest training error for each three approaches

For code to generate classification boundary plot, please refer to Appendix B

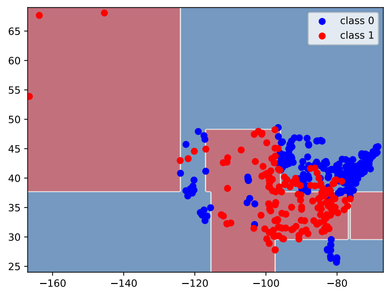

Figure 6: Classification boundary plot of lowest error on DecisionStump

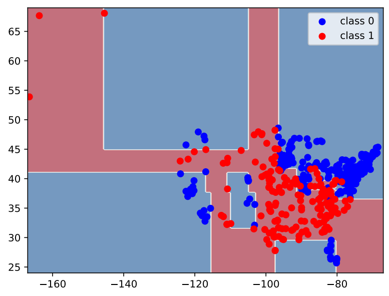

Figure 7: Classification boundary plot of lowest error on DecisionStumpInfoGain

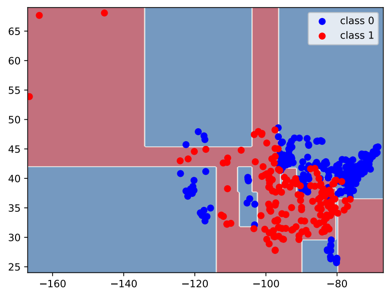

Figure 8: Classification boundary plot of lowest error on DecisionTreeClassifier

6.6 Comparing implementations

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?
With exact same curve of classification error rate vs. tree depth, we cannot conclude that two implementations are the same. If both implementations are using different splitting rules while depths and examples are very limited, there will be a high chance that both implementations yield the same result. Also if two different implementations are both overfitting to the given datasets, they will probably produce the same curve.
To test if implementations are equivalent, we can print out the splitting sequence of the implementations and see if the sequences are exactly the same. We can also apply the implementations to new dataset to see if they produce same curve or not.

6.7 Cost of Fitting Decision Trees

In class, we discussed how in general the decision stump minimizing the classification error can be found in $O(ndlogn)$ 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 $2m - 1$ 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 $2m-1$ decision stumps.

Although there could be $2m - 1$ decision stumps, the number of with the depth increases, the size of examples decreases by roughly $\frac{1}{2}$. With depth of m, the total cost will be O(mndlogn)

Appendices

A decision_stump.py

```
import numpy as np
import utils


class DecisionStumpEquality:

    def __init__(self):
        pass

    def fit(self, X, y):
        N, D = X.shape

        # Get an array with the number of 0's, number of 1's, etc.
        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)

        self.splitSat = y_mode
        self.splitNot = None
        self.splitVariable = None
        self.splitValue = None

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

        minError = np.sum(y != y_mode)

        # Loop over features looking for the best split
        X = np.round(X)

        for d in range(D):
            for n in range(N):
                # Choose value to equate to
                value = X[n, d]

                # Find most likely class for each split
                y_sat = utils.mode(y[X[:,d] == value])
                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
    def predict(self, X):

        M, D = X.shape
        X = np.round(X)

        if self.splitVariable is None:
            return self.splitSat * np.ones(M)

        yhat = np.zeros(M)

        for m in range(M):
            if X[m, self.splitVariable] == self.splitValue:
                yhat[m] = self.splitSat
            else:
                yhat[m] = self.splitNot

        return yhat


class DecisionStumpErrorRate:

    def __init__(self):
        pass

    def fit(self, X, y):
        N, D = X.shape

        # Get an array with the number of 0's, number of 1's, etc.
        count = np.bincount(y, minlength=2)

        # 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)

        self.splitSat = y_mode
        self.splitNot = None
        self.splitVariable = None
        self.splitValue = None

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

        minError = np.sum(y != y_mode)

        # Loop over features looking for the best split

        for d in range(D):
            for n in range(N):
                # Choose value to equate to
                value = X[n, d]

                # Find most likely class for each split
                y_sat = utils.mode(y[X[:,d] > value])
                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

    def predict(self, X):

        M, D = X.shape

        if self.splitVariable is None:
            return self.splitSat * np.ones(M)

        yhat = np.zeros(M)

        for m in range(M):
            if X[m, self.splitVariable] > self.splitValue:
                yhat[m] = self.splitSat
            else:
                yhat[m] = self.splitNot

        return yhat


"""
A helper function that computes the entropy of the
discrete distribution p (stored in a 1D numpy array).
The elements of p should add up to 1.
This function ensures lim p-->0 of p log(p) = 0
which is mathematically true (you can show this with l'Hopital's rule),
but numerically results in NaN because log(0) returns -Inf.
"""
```

```
def entropy(p):
    plogp = 0*p # initialize full of zeros
    plogp[p>0] = p[p>0]*np.log(p[p>0]) # only do the computation when p>0
    return -np.sum(plogp)


# 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?
class DecisionStumpInfoGain(DecisionStumpErrorRate):

    def fit(self, X, y):
        N, D = X.shape

        # Get an array with the number of 0's, number of 1's, etc.
        count = np.bincount(y, minlength=2)

        # 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)

        self.splitSat = y_mode
        self.splitNot = None
        self.splitVariable = None
        self.splitValue = None

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

        maxInfoGain = 0

        # Loop over features looking for the best split
        for d in range(D):
            for n in range(N):
                # Choose value to equate to
                value = X[n, d]

                # Find most likely class for each split
                y_sat = utils.mode(y[X[:,d] > value])
                y_not = utils.mode(y[X[:,d] <= value])

                # Make predictions
                y_pred = y_sat * np.ones(N)
                y_pred[X[:, d] <= value] = y_not

                # Create smaller datasets
                y_sat_vec = y[X[:,d] > value]
                y_not_vec = y[X[:,d] <= value]
                # Compute information gain
                p_before = np.bincount(y, minlength=len(count)) / len(y)
                p_yes = np.bincount(y_sat_vec, minlength=len(count)) / len(y_sat_vec)
                p_not = np.bincount(y_not_vec, minlength=len(count)) / len(y_not_vec)
                n_yes = len(y_sat_vec)
                n_before = len(y)
                n_no = n_before - n_yes
                infoGain = entropy(p_before) - n_yes/n_before * entropy(p_yes) \
                           - n_no/n_before * entropy(p_not)

                # Compare infoGain to minimum infoGain so far
                if infoGain > maxInfoGain:
                    maxInfoGain = infoGain
                    self.splitVariable = d
                    self.splitValue = value
                    self.splitSat = y_sat
                    self.splitNot = y_not
        # print("splitting stump by variable {} and > {} to {} and < {} to {}"
        #        .format(self.splitVariable, self.splitValue, y_sat, self.splitValue,
```

B main.py for Q6.5

```
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)

    min_err_err_rate = 1;
    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)

        if min_err_err_rate > my_tree_errors[i]:
            min_err_err_rate = my_tree_errors[i]
            # generate plot
            print("Generating plot")

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

    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)

    min_err_info_gain = 1
    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)

        if min_err_info_gain > my_tree_errors_infogain[i]:
            min_err_info_gain = my_tree_errors_infogain[i]
            # generate plot
            print("Generating plot")

            utils.plotClassifier(model, X, y)

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

    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)

    min_err_sklearn = 1
    for i, max_depth in enumerate(depths):
        model = DecisionTreeClassifier(max_depth=max_depth,
                                        criterion='entropy',
                                        random_state=1)
        model.fit(X, y)
        y_pred = model.predict(X)
```

```
        sklearn_tree_errors[i] = np.mean(y_pred != y)

        if min_err_sklearn > sklearn_tree_errors[i]:
            min_err_sklearn = sklearn_tree_errors[i]
            # generate plot
            print("Generating plot")

            utils.plotClassifier(model, X, y)

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

    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)

else:
    print("No code to run for question", question)
```