README.md

# Homework 5

There are two parts to this homework. In the first part, you will use k-means clustering and the sklearn Python library to analyze congressional voting data. In the second part, you will use implement and train a decision tree to classify congress members based on their voting patterns.

In addition to the code, you will also need to submit a writeup to Gradescope in the form of a file called `written.pdf`. Part of the test suite will be made available on Gradescope next week, so you can make sure your code is running on the autograder. This assignment is due on Friday, November 22nd at 11:59 PM.

## Part 1: Clustering (40%)

In this part of the assignment, you will see how unsupervised learning can discover clusters in data that correspond to clusters in the real world. In particular, you will discover clusters in congressional voting data that cluster congresspersons by looking only at their voting records.

You must complete the starter code provided in `clusters.py`. Submit your code to Gradescope. The autograder will be enabled 1 week prior to the due date.

You must also submit a PDF containing answers to the non-coding portions of the questions to Gradescope.

### Use sklearn to Implement K-Means Clustering

`sklearn` is a standard machine learning library in `Python`. `sklearn.cluster.KMeans` is an implementation of of the k-means clustering algorithm discussed in class. Complete the `CongressionalKMeans` implementation provided in `clusters.py` using `KMeans` with k=2. You will need to view the `sklearn` documentation in order to learn how to use the `KMeans` class.

In the constructor, you will need to read in the data from `congress_data.csv` in order to load the voting record of each congressperson. In order to apply k-means, you will need to convert the votes from a string format ("Yea", "No", etc.) to a numeric (in our case, 0-1) format. See the `to_numerical` function for more information. This is a real dataset, and contains several attributes that you will not need for this portion of the assignment. To their chagrin, data scientists spend much of their time reading data files, so this will be good preparation for the real world.

When you've completed the implementation, you should be able to predict the cluster that each congressperson belongs to, as well as the most likely voting record for a congressperson in each cluster.

### Analyze the Data

Congratulations, you have trained an unsupervised learning model! In this part of the assignment, you will use the model to draw conclusions about the data. Please report your answers as part of your solution PDF.

The vote IDs in the data correspond to real congressional votes. You can find a record of these at the following url: https://www.govtrack.us/congress/votes#session=302&chamber[]=2

Answer the following questions:

1. You should find, unsurprisingly, that the clusters found correspond very well to the two major political parties (Democrats and Republicans). What percentage of congresspeople does your model appear to "correctly" classify? (Hint: You will need to write some additional code to compute the answer.)
2. Look at the cases where the congresspeople were misclassified. What went wrong? How could our clustering be improved?
3. You may have noticed that some votes were split by cluster, where as others were not. On what percentage of votes did the

"median" voter in each cluster differ? Based on the description of the votes provided by GovTrack, on what types of votes did each party vote the same? On which types of votes did they differ?

### Extra Credit: k=4

Repeat the above experiments with k=4. The clusters no longer have a one-to-one correspondence with political parties, as there are four clusters but only two parties.

1. What do the four clusters correspond to? Use descriptions of votes from GovTrack to support your answer.
2. Adding more clusters does not necessarily improve the quality of the model. In this case, is the model with 2 clusters "better," or the model with 4 clusters? Support your answer by comparing the "median" voter from each cluster.

## Part 2: Decision Trees (60%)

Now, you will implement a decision tree classifier and use it to classify members of congress based on their votes on certain issues. Due to partisanship, many of the votes in the large dataset are perfectly predictive of party affiliation by themselves. Thus, for this part, you will use the `congress_small.csv` dataset, which contains only the votes that are not strongly predictive of party alignment on their own. This way, your tree actually has some work to do, and it cannot obtain perfect accuracy by doing a single split.

### Step 1: Reading

Decision trees are one of the weaker points in Russell and Norvig, so before you start, we would like you to take a look at this overview from Provost and Fawcett's *Data Science for Business*:

https://d1b10bmlvqabco.cloudfront.net/attach/jr9anznu26h4ex/jlp7k6sk6ok4jp/juahdia38czo/provost_fawcett_dsb_entropy.pdf

This resource gives a good intuition about the entropy and information gain, which will help you understand the pseudocode in Russell and Norvig (p. 702).

### Step 2: Implementation

We have provided you with some starter code in the file `tree.py` . Start by looking over the given code to understand the basic structure of the program. There are `# TODO` comments in all the places where you need to add your own implementation. You will want to reference the pseudocode in Russell and Norvig while writing your implementation.

**The rules.** You are free to add any helper functions you want, but **DO NOT MODIFY ANY OF THE GIVEN CODE**. If you modify the function signatures or class members, the autograder will not run, and you will not receive points on the assignment. The only provided code that you are allowed to change is the code in the `main` function (and, of course, the `# TODO` comments and `pass` statements).

**Helpful hints.** Our recommendation is to start by writing code for the entropy and information gain, and test these functions independently before starting on the decision tree logic. We also encourage you to break up your decision tree training code into multiple functions, as the training function will get very big otherwise (our implementation uses several helper functions).

**Depth limit.** Unless you stop your decision tree before it's done training, it will continue to branch until every leaf is a pure set. This is almost always bad, because it usually causes your tree to overfit the training set. To prevent overfitting, your code will accept a depth-limit parameter. Once the tree reaches the depth limit, any decision node whose example set is not pure will simply predict the plurality class among its examples. A depth of zero means there should be no decision nodes in the tree; it should just make a prediction based on the plurality class in the dataset. Depth one means there should be one decision node before a prediction, etc.

**Tiebreaking.** Sometimes, your tree may encounter a node at which multiple attributes have the same information gain. If this happens, you should choose the attribute that comes first in a lexicographic ordering.

**Missing data.** You'll notice that in addition to "Yea" and "Nay", values may be "Not Voting", "Present", or missing entirely. For the purposes of this assignment, you can ignore those values by treating them like the majority value for that vote.

**Testing.** You are not required to submit any test results or pre-trained models. You will be graded based on the final performance of your decision tree code across a variety of training sets and depth limits. That said, you should run whatever tests you need to convince yourself that your code is bug-free. One good testing strategy is to make some very small versions of the dataset where you can predict what trees should result from the training process. Make sure that your code works regardless of the number of votes and the attribute to be predicted. It's natural to predict party based on voting history, but your code should be able to predict a congress member's vote on some issue given their other votes and/or their party affiliation.

## Final Submission

Submit a zip file containing:

- The file `clustering.py` with your k-means implementation.
- The file `tree.py` with your decision tree implementation.
- A file called `written.pdf` containing your answers to the written questions.