HOMEWORK 1: BACKGROUND

10-301/10-601 Introduction to Machine Learning (Spring 2022)
http://www.cs.cmu.edu/~mgormley/courses/10601/
OUT: Wednesday, January 19th
DUE: Wednesday, January 26th
TAs: Sana, Abhi, Sami, Zachary, Brendon, Mukund

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 2.1"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html#7-academic-integrity-policies
- **Late Submission Policy:** See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html#late-homework-policy
- **Submitting your work:**
  - **Programming:** You will submit your code for programming questions on the homework to Gradescope (https://gradescope.com). After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). When you are developing, check that the version number of the programming language environment (e.g. Python 3.9.6) and versions of permitted libraries (e.g. `numpy` 1.21.2 and `scipy` 1.7.1) match those used on Gradescope. You have a **total of 10 Gradescope programming submissions.** Use them wisely. In order to not waste code submissions, we recommend debugging your implementation on your local machine (or the linux servers) and making sure your code is running correctly first before any Gradescope coding submission. The above is true for future assignments, but this one allows **unlimited submissions.**
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using Gradescope (https://gradescope.com/). Please use the provided template. Submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Regrade requests can be made, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. Each derivation/proof should be completed on a separate page. For short answer questions you **should not** include your work in your solution. If you include your work in your solutions, your assignment may not be graded correctly by our AI assisted grader. For this assignment only, if you answer at least 90% of the written questions correctly, you get full marks on the written portion of this assignment. For this assignment only, **we will offer two**

1 Programming: Majority Vote Classifier [30 Points]

1.1 Introduction

In this homework you have to choose Python as your programming language. Submitting code for more than one language may result in undefined behavior.

The goal of this assignment is to ensure that you:

1. Have a way to edit and test your code (i.e. a text editor and compiler/interpreter)
2. Are familiar with submitting to Gradescope
3. Are familiar with file I/O and standard output in the language of your choice

Warning: This handout assumes that you are using a unix command prompt (with `zsh, bash, csh` or similar). Windows commands may differ slightly.

1.2 Majority Vote Classifier

1.2.1 Algorithm

This assignment requires you to implement a Majority Vote Classifier. Your algorithm should calculate the most common label in the data, "predict" that label for each given point in the dataset, and calculate the error rate for the classifier's predictions. You may assume that the output class label is always binary.

The training procedure should store the label used for prediction at test time. In the case of a tie, output the value that comes *last* alphabetically. At test time, each example should be passed through the classifier. Its predicted label becomes the label most commonly occurring in the train set.

Looking ahead: This simple algorithm acts as a small component of the Decision *Tree* that you will implement in the next homework assignment. We hope that you will employ best practices when coding so that you can re-use your own code here in the next assignment. A Majority Vote Classifier is simply a decision tree of depth zero (it predicts a class label for the input instance based on the most commonly occurring label present in the data).

1.2.2 The Datasets

Materials Download the zip file from course website, which contains all the data that you will need in order to complete this assignment.

Datasets The handout contains three datasets. Each one contains attributes and labels and is already split into training and testing data. The first row of each `.tsv` file contains the name of each attribute, and *the class label is always the last column*.

1. **politician:** The first task is to predict whether a US politician is a member of the Democrat or Republican party, based on their past voting history. Attributes (aka. features) are short descriptions of bills that were voted on, such as *Aid_to_nicaraguan_contras* or *Duty_free_exports*. Values are given as *'y'* for yes votes and *'n'* for no votes. The training data is in `politicians_train.tsv`, and the test data in `politicians_test.tsv`.
2. **education:** The second task is to predict the final *grade* (A, not A) for high school students. The attributes (covariates, predictors) are student grades on 5 multiple choice assignments *M1* through *M5*, 4 programming assignments *P1* through *P4*, and the final exam *F*. The training data is in `education_train.tsv`, and the test data in `education_test.tsv`.

The handout zip file also contains the predictions and metrics from a reference implementation of a Majority

Vote Classifier for the **politician** and **education** datasets (see subfolder *example_output*). You can check your own output against these to see if your implementation is correct.[1]

Note: For simplicity, all attributes are discretized into just two categories. This applies to all the datasets in the handout, as well as the additional datasets on which we will evaluate your Majority Vote Classifier.

1.2.3 Command Line Arguments

The autograder runs and evaluates the output from the files generated, using the following command:

```
$ python majority_vote.py [args...]
```

Where above `[args...]` is a placeholder for five command-line arguments: `<train input> <test input> <train out> <test out> <metrics out>`. These arguments are described in detail below:

1. `<train input>`: path to the training input `.tsv` file
2. `<test input>`: path to the test input `.tsv` file
3. `<train out>`: path of output `.labels` file to which the predictions on the *training* data should be written
4. `<test out>`: path of output `.labels` file to which the predictions on the *test* data should be written
5. `<metrics out>`: path of the output `.txt` file to which metrics such as train and test error should be written

As an example, the following command line would run your program on the politicians dataset. The train predictions would be written to `pol_train.labels`, the test predictions to `pol_test.labels`, and the metrics to `pol_metrics.txt`.

```
$ python majority_vote.py politicians_train.tsv politicians_test.tsv \
          pol_train.labels pol_test.labels pol_metrics.txt
```

1.2.4 Output: Labels Files

Your program should write two output `.labels` files containing the predictions of your model on training data (`<train out>`) and test data (`<test out>`). Each should contain the predicted labels for each example printed on a new line. Use '\n' to create a new line.

Your labels should exactly match those of a reference majority vote classifier implementation—this will be checked by the autograder by running your program and evaluating your output file against the reference solution.

Note: You should output your predicted labels using the same string identifiers as the original training data: e.g., for the politicians dataset you should output democrat/republican and for the education dataset you should output A/notA. The first few lines of an example output file is given below for the politician dataset:

```
democrat
democrat
```

[1] Yes, you read that correctly: we are giving you the correct answers.

```
democrat
democrat
democrat
democrat
democrat
...
```

1.2.5 Output: Metrics File

Generate another file where you should report the training error and testing error. This file should be written to the path specified by the command line argument `<metrics out>`. Your reported numbers should be within 0.01 of the reference solution. You do not need to round your reported numbers! The Autograder will automatically incorporate the right tolerance for float comparisons. The file should be formatted as follows:

```
error(train): 0.442953
error(test): 0.506024
```

1.3 Command Line Arguments

In this and future programming assignments, we will use command line arguments to run your programs with different parameters. Below, we provide some simple examples for how to do this in Python. In the examples below, suppose your program takes two arguments: an input file and an output file.

Python:

```
import sys

if __name__ == '__main__':
    infile = sys.argv[1]
    outfile = sys.argv[2]
    print("The input file is: %s" % (infile))
    print("The output file is: %s" % (output))
```

1.4 Code Submission

You must submit a file named `majority_vote.py`. The autograder is case sensitive. You must submit this file to the corresponding homework link on Gradescope.

Note: For this assignment, you may make arbitrarily many submissions to the autograder before the deadline, but only your last submission will be graded.