HOMEWORK 1: DECISION STUMPS

10-601 Introduction to Machine Learning (Summer 2020)
https://www.cs.cmu.edu/~10601/
DUE: Thursday, September 10, 2020 11:59 PM

START HERE: Instructions

- **Collaboration policy:** Please read the collaboration policy here: https://www.cs.cmu.edu/~10601
- **Late Submission Policy:** See the late submission policy here: https://www.cs.cmu.edu/~10601
- **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 permitted libraries match those used on Gradescope, specifically OpenJDK 11.0.5, g++ 7.4.0 (C++14), Python 3.6.9, NumPy 1.17.0, and SciPy 1.4.1. If you have minor variations of these versions in your development environment, you will probably be fine; just be sure to check that your code works when submitted to Gradescope. Built-in libraries included in Python, Java, and C++ are available, but unless otherwise mentioned, additional libraries are not supported. For example in Python, the built-in csv module is available but libraries like pandas and scikit-learn are not.
  - **Written:** In addition to this programming assignment, there is a written portion of HW1 on Gradescope. On Gradescope, there is an assignment for HW1 programming where you will submit your code, and a separate assignment for HW1 written, which you will complete online. Both the programming and written portions are part of your HW1 grade, with the programming assignment accounting for 30% of your HW1 grade, and the written assignment accounting for 70% of your HW1 grade.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the write-up on Piazza.

Programming: (30 points)

1 Introduction

In this homework you have to choose Python, Java, or C++ 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.

2 Decision Stump

2.1 Algorithm

This simple algorithm acts a precursor to 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.

This assignment requires you to implement a Decision Stump. You may assume that the attribute on which the Decision Stump splits is always provided (on the command line). The algorithm should partition the provided training data based on the attribute that you are splitting on. You may assume that the attributes are always binary and that the output class is always binary. As such, the left branch of the stump should perform a majority vote on the subset of the training data corresponding to that branch's value for the given attribute. The right branch should do likewise for the other value of the attribute. The training procedure should store the decision stump data structure for use at test time.

At test time, each example should be passed down through the stump. Its label becomes the label (i.e. the stored majority vote) of the corresponding branch in which it lands.

2.2 The Datasets

Materials Download the zip file from Piazza, 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 line of each `.tsv` file contains the name of each attribute, and *the class 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`.

3. **small:** We also include `small_train.tsv` and `small_test.tsv`—a small, purely for demonstration version of the politicians dataset, with *only* attributes *Anti_satellite_test_ban* and *Export_south_africa*.

The handout zip file also contains the predictions and metrics from a reference implementation of a Decision Stump for the **politician** (splitting on feature 3), **education** (splitting on feature 5) and **small** (splitting on feature 0) 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 Decision Stump.

2.3 Command Line Arguments

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

For Python:
```
$ python decisionStump.py [args...]
```

For Java:
```
$ javac decisionStump.java; java decisionStump [args...]
```

For C++:
```
$ g++ -g decisionStump.cpp; ./a.out [args...]
```

Where above `[args...]` is a placeholder for six command-line arguments: `<train input> <test input> <split index> <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. `<split index>`: the index of feature at which we split the dataset. The first column has index 0, the second column index 1, and so on.
4. `<train out>`: path of output `.labels` file to which the predictions on the *training* data should be written
5. `<test out>`: path of output `.labels` file to which the predictions on the *test* data should be written
6. `<metrics out>`: path of the output `.txt` file to which metrics such as train and test error should be written

As an example, if you implemented your program in Python, the following command line would run your program on the politicians dataset and split the dateset by the first feature (Remember that the index of feature starts from zero). The train predictions would be written to `pol_0_train.labels`, the test predictions to `pol_0_test.labels`, and the metrics to `pol_0_metrics.txt`.

```
$ python decisionStump.py politicians_train.tsv politicians_test.tsv \
        0 pol_0_train.labels pol_0_test.labels pol_0_metrics.txt
```

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

Linear Algebra Libraries When implementing machine learning algorithms, it is often convenient to have a linear algebra library at your disposal. In this assignment, Java users may use EJML[a] or ND4J[b] and C++ users Eigen[c]. Details below. (As usual, Python users have NumPy.)

EJML for Java EJML is a pure Java linear algebra package with three interfaces. We strongly recommend using the SimpleMatrix interface. The autograder will use EJML version 0.38. When compiling and running your code, we will add the additional command line argument `-cp "linalg_lib/ejml-v0.38-libs/*:linalg_lib/nd4j-v1.0.0-beta7-libs/*:./"` to ensure that all the EJML jars are on the classpath as well as your code.

ND4J for Java ND4J is a library for multidimensional tensors with an interface akin to Python's NumPy. The autograder will use ND4J version 1.0.0-beta7. When compiling and running your code, we will add the additional command line argument `-cp "linalg_lib/ejml-v0.38-libs/*:linalg_lib/nd4j-v1.0.0-beta7-libs/*:./"` to ensure that all the ND4J jars are on the classpath as well as your code.

Eigen for C++ Eigen is a header-only library, so there is no linking to worry about—just `#include` whatever components you need. The autograder will use Eigen version 3.3.7. The command line arguments above demonstrate how we will call you code. When compiling your code we will include, the argument `-I./linalg_lib` in order to include the `linalg_lib/Eigen` subdirectory, which contains all the headers.

We have included the correct versions of EJML/ND4J/Eigen in the `linalg_lib.zip` posted on the Piazza Resources page for your convenience. It contains the same `linalg_lib/` directory that we will include in the current working directory when running your tests. Do **not** include EJML, ND4J, or Eigen in your homework submission; the autograder will ensure that they are in place.

[a] https://ejml.org
[b] https://deeplearning4j.org/docs/latest/nd4j-overview
[c] http://eigen.tuxfamily.org/

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 decision stump 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:

```
republican
republican
democrat
democrat
democrat
democrat
democrat
...
```

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 0.001 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): 0.241611
error(test): 0.228916
```

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 each of the programming languages you can use in the course. In the examples below, suppose your program takes two arguments: a file with your training data, and a file with your test data.

Python:

```
import sys

if __name__ == '__main__':
    traininput = sys.argv[1]
    testinput = sys.argv[2]
    print("The train input file is: %s" % (traininput))
    print("The test input file is: %s" % (testinput))
```

Java:

```
public class myclass {
   public static void main(String[] args) {
        String traininput = args[0];
        String testinput = args[1];
        System.out.println("The train input file is: " + traininput);
        System.out.println("The test input file is: " + testinput);
   }
}
```

C++:

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char **argv){
    if (argc >= 3) {
        string traininput = string(argv[1]);
        string testinput = string(argv[2]);
        cout << "The train input file is: " << traininput << endl;
        cout << "The test input file is: " << testinput << endl;
    }
    return 0;
}
```

4 Code Submission

You must submit a file named `decisionStump.{py|java|cpp}`. The autograder is case sensitive, so observe that all your files should be named in **camelCase**. You must submit this file to the corresponding homework link on Gradescope.

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