CS1210 Computer Science I: Fundamentals
Homework 2: Managing Data
Due Monday, November 6 at 11:59PM

Introduction

This is the second project for CS1210. Please don't wait to start on this project, or I can guarantee you won't finish it. So, just to review:

(1) This is a challenging project, and you have been given two weeks to work on it. **If you wait to begin, you will almost surely fail to complete it.** The best strategy for success is to work on the project a little bit every day.

(2) **I will cover the Naive Bayes classifier in class**; we'll take a day to do so soon. I would start with the first part of the assignment and wait to deal with the prediction part of the assignment until we cover Naive Bayes.

(3) **The work you hand in should be only your own; you are not to work with or discuss your work with any other student.** Sharing your code or referring to code produced by others is a violation of the student honor code and will be dealt with accordingly.

(4) **Help is always available from the TAs or the instructor during their posted office hours.** You may also post general questions on the discussion board (although you should never post your Python code). **I have opened a discussion board topic specifically for HW2.**

Background

In this assignment, we will be looking at a dataset of survey responses collected from subjects asked to evaluate their risk tolerance. Each subject was asked to consider two hypothetical lotteries:

A: you have a 50% chance of success, with a payout of $100.
B: you have a 90% chance of success, with a payout of $20.

Assuming the subject was given the chance to bet $10, they were asked which lottery they would choose (we'll call this the *result* field, with possible values $\{lottery\ a, lottery\ b\}$).

In addition to their choice of lottery, subjects were asked to provide a number of demographic variables, here denoted by field name and showing possible values:

$gender = \{male, female\}$
$age = \{18-29, 30-44, 45-60, >60\}$
$income = \{\$0-\$24k, \$25k-\$49k, \$50k-\$99k, \$100k-\$149k, >\$150\}$
$education = \{high\ school, some\ college, bachelors\ degree, graduate\ degree\}$
$location = \{east\ north\ central, east\ south\ central, middle\ atlantic, mountain, new\ england, pacific, south\ atlantic, west\ north\ central, west\ south\ central\}$

as well as answer a number of additional questions:

$smoke = \{yes, no\}$
$drink = \{yes, no\}$
$gamble = \{yes, no\}$
$skydive = \{yes, no\}$
$speed = \{yes, no\}$
$cheat = \{yes, no\}$
$steak = \{yes, no\}$
$cook = \{rare, medium\ rare, medium, medium\ well, well\}$

The header of the csv file provided gives the wording of the questions; here, we've used the names I used in my Python solution.

In this project, we will be (*i*) loading the data into Python and representing it as a dictionary, (*ii*) plotting aspects of the data as a form of interactive data exploration, (*iii*) applying a machine learning technique called a *Naive Bayes* classified in an attempt to learn a relation that allows us to predict someone's preference based on their demographic and risk-tolerance variables, and (*iv*), evaluating our learning algorithm in a principled fashion.

The rest of these instructions outline the functions that you should implement, describing their input/output behaviors. As you work on each function, test your work on the document provided to make sure your code functions as expected. Feel free to upload versions of your code as you go; we only grade the last version uploaded, so this practice allows you to "lock in" working partial solutions prior to the deadline. Finally, some general guidance.

(1) Start with the template file and **immediately complete** the *hawkid*() function so that we may properly credit you for your work. **Test** *hawkid*() **to ensure it in fact returns your own hawkid as the only element in a single element tuple.** Also be sure to add your name and section to the comments at the top. Ignore this instruction at your own risk: some of you have become serial offenders and I have had to manually regrade your code too many times!

(2) The template file also contains two useful lists, *fields* and *values*, which you can use when parsing the data. Note that these lists contain (all lower-case renditions) of the data field names and possible values, respectively (see the descriptions above), so you will likely have to convert some inputs to lower case in order to compare them with the values provided.

(3) At the risk of screwing up the autograder (and your grade!), do not change the values of *fields* and *values*; use them as provided.

(4) As with HW1, you will be graded on both the **correctness** and the **quality** of your code, including the quality of your **comments**!

(5) As usual, respect the function signatures provided.

(6) Be careful with iteration; **always choose the most appropriate form of iteration (comprehension, while, or for)** as the function mandates. Poorly selected iterative forms may be graded down, even if they work!

(7) Like for HW1, we will use the autograder to determine a portion of your grade, with the TAs assigning the remining points based on style and the overall quality of your code.

def readData(filename='steak-risk-survey.csv', fields=fields, values=values):

This function returns a list, where each element of the list is a dictionary corresponding to a single line of the data from the input file. You will find the csv module and the csv.reader() function useful here; you can read about these at the following URL:

https://docs.python.org/3.5/library/csv.html

My solution makes use of a helper function that converts each data element — a row in the original csv file provided — into a dictionary, with the main function tasked with reading in each line of data and producing the list returned.

A few caveats are in order. First, we don't care about the 'RespondentID' field, and this can be safely ignored. Second, there are some rows in the input file that don't correspond to survey responses and so can also be ignored. Finally, any survey response that fails to provide a choice of lottery (the second column in the original csv file, immediately following the 'RespondentID' column, labeled 'result' in the

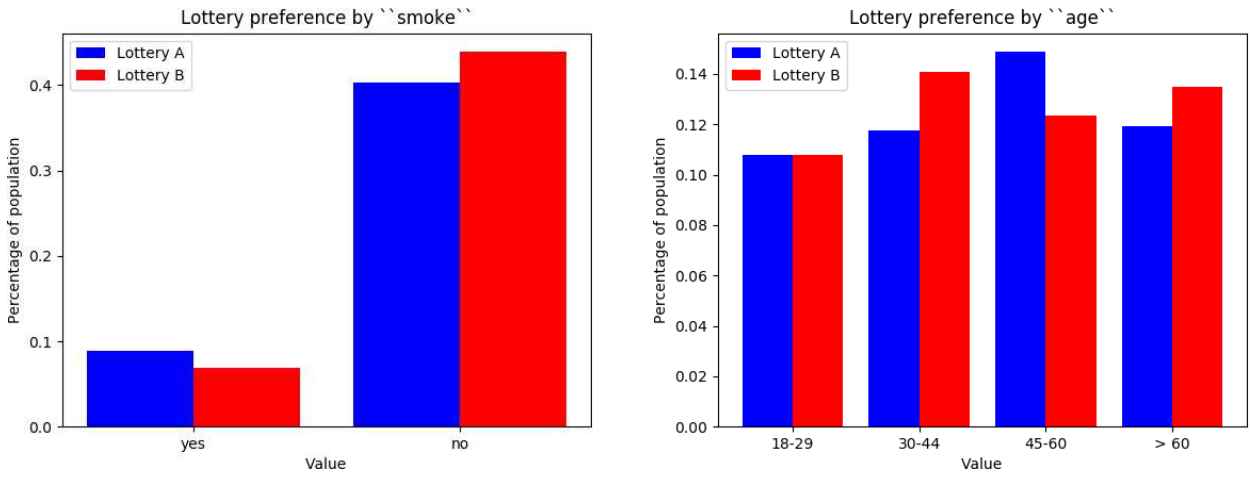
fields variable, can also be ignored. Following these caveats, my code yields:

```
>>> D=readData('steak-risk-survey.csv')
>>> len(D)
546
>>> D[0]
{'result': 'lottery b'}
>>> D[1]
{'result': 'lottery a', 'smoke': 'no', 'drink': 'yes', 'gamble': 'no',    \\
'skydive': 'no', 'speed': 'no', 'cheated': 'no', 'steak': 'yes',          \\
'cook': 'medium rare', 'gender': 'male', 'age': '> 60',                   \\
'income': '50,000-99,999', ''education': 'some college or associate degree',  \\
'location': 'east north central'}
```

where $D[0]$ corresponds to survey respondent 3237565956 and $D[1]$ corresponds to survey respondent 3234982343 in the original csv file (I've used \\ to indicate an extra line feed for readability).

def showPlot(D, field, values):

For a given field and its associated values, plots a chart showing lottery preference that looks like these for $showPlot(D, 'smoke', ('yes', 'no'))$ and $showPlot(D, 'age', ('18-29', '30-44', '45-60', '>60'))$ where D is a set of examples returned by *readData*()



Of course, your function should work for any field/value pairs from the original data. An excellent reference on using matplotlib in Python can be found here:

https://pythonspot.com/en/matplotlib-bar-chart/

train(D, fields=fields, values=values):

This function trains a Naive Bayes classifier to predict lottery preference based on the a collection of the other variables. The basic idea here is quite simple. In the absence of any other information about an individual, we can predict that individuals lottery preference simply by comparing probabilities $P('lottery\ a')$ and $P('lottery\ b')$, which can be approximated by the frequencies with which these appear in the population. In other words, if there are 1000 examples in D and 650 of these prefer Lottery B, then a new individual's lottery preference is probably also Lottery B, given that 65% of the observed individuals prefer Lottery B over Lottery A.

We can extend this idea to using conditional probabilities. If I know whether or not the individuals in D are smokers, than I can ‘‘condition’’ my prediction based on whether the new individual is a smoker. Consider the following breakdown:

	Lottery A	*Lottery B*	*Total*
smoker	200	150	350
non-smoker	150	500	650
Total	250	650	1000

Clearly, if I know nothing about the new individual, then I should predict that they will prefer Lottery B (650 to 350). But if I know the new individual is a smoker, than I should predict that they would prefer Lottery A (200 to 150). This is the key idea underlying the Naive Bayes algorithm; a really good and more complete explanation will be given in class. Now what about *train*()? First, this function should produce a table of counts analogous to the one shown above for each of the specified fields and their values. The easiest way to code such a function is to create a dictionary of dictionaries of dictionaries, where the outermost dictionary is indexed by field in fields, the next dictionary is indexed by lottery preference, and the innermost dictionary is indexed by field value. Once the table is constructed, the values within should be converted to probabilities, by an appropriate series of sums and divisions. So, for example:

```
>>> P['smoke']
{'lottery a': {'no': 0.821, 'yes': 0.179}, 'lottery b': {'no': 0.863, 'yes': 0.137}}
```

where I've reduced the number of decimals shown to make them fit on this page. This classifier (the table of probabilities, *P*) is now ready to use for prediction purposes.

predict(example, P, fields=fields, values=values):

Predicition will be explained in class. The short answer is that the probability a new individual will prefer a given lottery is the product of probabilities for that lottery over all matching field values and individuals who share them.

test(D, P, fields=fields, values=values):

The *test*() function takes a classifier, *P*, and a collection of of examples, *D*, and compares the prediction of the classifier using only the specified fields to the actual observed value, reporting a percentage of correct outcomes.

Testing Your Code

I have provided a function, *evaluate*(*fields*, *values*), that you can use to manage the process of evaluating a subset of fields as predictive elements. When invoked, this function will reserve 10% of the data for testing, train the Naive Bayes prediction algorithm with the remaining 90% of the data, and then return the percentage of correct predictions made by the system.

```
>>> evaluate()
NaiveBayes=0.5016666666666666, random guessing=0.49370370370370364
>>> evaluate(fields=['result', 'gamble'], values=[('lottery a','lottery b'),('no','yes')])
NaiveBayes=0.532962962962963, random guessing=0.5083333333333333
>>> evaluate(fields=['result', 'gamble'], values=[('lottery a','lottery b'),('no','yes')])
NaiveBayes=0.5324074074074077, random guessing=0.4981481481481481
>>> evaluate(fields=['result', 'drink', 'cheat'], \\
             values=[('lottery a','lottery b'),('no','yes'),('no','yes')])
```

```
NaiveBayes=0.5051851851851853, random guessing=0.5007407407407407
>>> evaluate(fields=['result', 'drink', 'cheat'], \\
            values=[('lottery a','lottery b'),('no','yes'),('no','yes')])
NaiveBayes=0.505, random guessing=0.5112962962962964
>>> evaluate(fields=['result','steak','gender','age'], \\
            values=[('lottery a','lottery b'),('no','yes'),('male','female'), \\
                    ('18-29','30-44','45-60','> 60')])
NaiveBayes=0.5309259259259261, random guessing=0.49907407407407406
```

which illustrates how just gambling, drinking, cheating, eating and age provide insight into risk tolerance!