Project#2: Online Shopping

Write an AI to predict whether online shopping customers will complete a purchase.

```
$ python shopping.py shopping.csv
Correct: 4088
Incorrect: 844
True Positive Rate: 41.02%
True Negative Rate: 90.55%
```

Background

When users are shopping online, not all will end up purchasing something. Most visitors to an online shopping website, in fact, likely don't end up going through with a purchase during that web browsing session. It might be useful, though, for a shopping website to be able to predict whether a user intends to make a purchase or not: perhaps displaying different content to the user, like showing the user a discount offer if the website believes the user isn't planning to complete the purchase. How could a website determine a user's purchasing intent? That's where machine learning will come in.

Your task in this problem is to build a nearest-neighbor classifier to solve this problem. Given information about a user — how many pages they've visited, whether they're shopping on a weekend, what web browser they're using, etc. — your classifier will predict whether or not the user will make a purchase. Your classifier won't be perfectly accurate — perfectly modeling human behavior is a task well beyond the scope of this class — but it should be better than guessing randomly. To train your classifier, we'll provide you with some data from a shopping website from about 12,000 users sessions.

How do we measure the accuracy of a system like this? If we have a testing data set, we could run our classifier on the data, and compute what proportion of the time we correctly classify the user's intent. This would give us a single accuracy percentage. But that number might be a little misleading. Imagine, for example, if about 15% of all users end up going through with a purchase. A classifier that always predicted that the user would not go through with a purchase, then, we would measure as being 85% accurate: the only users it classifies incorrectly are the 15% of users who do go through with a purchase. And while 85% accuracy sounds pretty good, that doesn't seem like a very useful classifier.

Instead, we'll measure two values: sensitivity (also known as the "true positive rate") and specificity (also known as the "true negative rate"). Sensitivity refers to the proportion of positive examples that were correctly identified: in other words, the proportion of users who did go through with a purchase who were correctly identified. Specificity refers to the proportion of negative examples that were correctly identified: in this case, the proportion of users who did not go through with a purchase who were correctly identified. So our "always guess no" classifier

from before would have perfect specificity (1.0) but no sensitivity (0.0). Our goal is to build a classifier that performs reasonably on both metrics.

Getting Started

- Download the distribution code from https://cdn.cs50.net/ai/2020/x/projects/4/shopping.zip and unzip it.
- Run `pip3 install scikit-learn` to install the `scikit-learn` package if it isn't already installed, which you'll need for this project.

Understanding

First, open up `shopping.csv`, the data set provided to you for this project. You can open it in a text editor, but you may find it easier to understand visually in a spreadsheet application like Microsoft Excel, Apple Numbers, or Google Sheets.

There are about 12,000 user sessions represented in this spreadsheet: represented as one row for each user session. The first six columns measure the different types of pages users have visited in the session: the `Administrative`, `Informational`, and `ProductRelated` columns measure how many of those types of pages the user visited, and their corresponding `_Duration` columns measure how much time the user spent on any of those pages. The `BounceRates`, `ExitRates`, and `PageValues` columns measure information from Google Analytics about the page the user visited. `SpecialDay` is a value that measures how close the date of the user's session is to a special day (like Valentine's Day or Mother's Day). `Month` is an abbreviation of the month the user visited. `OperatingSystems`, `Browser`, `Region`, and `TrafficType` are all integers describing information about the user themself. `VisitorType` will take on the value `Returning_Visitor` for returning visitors and some other string value for non-returning visitors. `Weekend` is `TRUE` or `FALSE` depending on whether or not the user is visiting on a weekend.

Perhaps the most important column, though, is the last one: the `Revenue` column. This is the column that indicates whether the user ultimately made a purchase or not: `TRUE` if they did, `FALSE` if they didn't. This is the column that we'd like to learn to predict (the "label"), based on the values for all of the other columns (the "evidence").

Next, take a look at `shopping.py`. The `main` function loads data from a CSV spreadsheet by calling the `load_data` function and splits the data into a training and testing set. The `train_model` function is then called to train a machine learning model on the training data. Then, the model is used to make predictions on the testing data set. Finally, the `evaluate` function determines the sensitivity and specificity of the model, before the results are ultimately printed to the terminal.

The functions `load_data`, `train_model`, and `evaluate` are left blank. That's where you come in!

Specification

An automated tool assists the staff in enforcing the constraints in the below specification. Your submission will fail if any of these are not handled properly, if you import modules other than those explicitly allowed, or if you modify functions other than as permitted.

Complete the implementation of `load_data`, `train_model`, and `evaluate` in `shopping.py`.

The `load_data` function should accept a CSV filename as its argument, open that file, and return a tuple `(evidence, labels)`. `evidence` should be a list of all of the evidence for each of the data points, and `labels` should be a list of all of the labels for each data point.

- Since you'll have one piece of evidence and one label for each row of the spreadsheet, the length of the `evidence` list and the length of the `labels` list should ultimately be equal to the number of rows in the CSV spreadsheet (excluding the header row). The lists should be ordered according to the order the users appear in the spreadsheet. That is to say, `evidence[0]` should be the evidence for the first user, and `labels[0]` should be the label for the first user.
- Each element in the `evidence` list should itself be a list. The list should be of length 17: the number of columns in the spreadsheet excluding the final column (the label column).
- The values in each `evidence` list should be in the same order as the columns that appear in the evidence spreadsheet. You may assume that the order of columns in `shopping.csv` will always be presented in that order.
- Note that, to build a nearest-neighbor classifier, all of our data needs to be numeric. Be sure that your values have the following types:
    - `Administrative`, `Informational`, `ProductRelated`, `Month`, `OperatingSystems`, `Browser`, `Region`, `TrafficType`, `VisitorType`, and `Weekend` should all be of type `int`
    - `Administrative_Duration`, `Informational_Duration`, `ProductRelated_Duration`, `BounceRates`, `ExitRates`, `PageValues`, and `SpecialDay` should all be of type `float`.
    - `Month` should be `0` for January, `1` for February, `2` for March, etc. up to `11` for December.
    - `VisitorType` should be `1` for returning visitors and `0` for non-returning visitors.
    - `Weekend` should be `1` if the user visited on a weekend and `0` otherwise.
- Each value of `labels` should either be the integer `1`, if the user did go through with a purchase, or `0` otherwise.
- For example, the value of the first evidence list should be `[0, 0.0, 0, 0.0, 1, 0.0, 0.2, 0.2, 0.0, 0.0, 1, 1, 1, 1, 1, 1, 0]` and the value of the first label should be `0`.

The `train_model` function should accept a list of evidence and a list of labels, and return a `scikit-learn` nearest-neighbor classifier (a ***k-nearest-neighbor classifier where `k = 1`***) fitted on that training data.

- Notice that we've already imported for you `from sklearn.neighbors import KNeighborsClassifier`. You'll want to use a `KNeighborsClassifier` in this function.

The `evaluate` function should accept a list of `labels` (the true labels for the users in the testing set) and a list of `predictions` (the labels predicted by your classifier), and return two floating-point values `(sensitivity, specificity)`.

- `sensitivity` should be a floating-point value from 0 to 1 representing the "true positive rate": the proportion of actual positive labels that were accurately identified.
- `specificity` should be a floating-point value from 0 to 1 representing the "true negative rate": the proportion of actual negative labels that were accurately identified.
- You may assume each label will be `1` for positive results (users who did go through with a purchase) or `0` for negative results (users who did not go through with a purchase).
- You may assume that the list of true labels will contain at least one positive label and at least one negative label.

You should not modify anything else in `shopping.py` other than the functions the specification calls for you to implement, though you may write additional functions and/or import other Python standard library modules. You may also import `numpy` or `pandas` or anything from `scikit-learn`, if familiar with them, but you should not use any other third-party Python modules. You should not modify `shopping.csv`.

Deliverables

Submit the completed file `shopping.py`.

Grading (out of 100):

- 30pts for the correct implementation of `load_data`,
- 35pts for the correct implementation of `train_model`,
- 35pts for the correct implementation of `evaluate`.
- 5pts (bonus) show the confusion matrices for k=3 and k=1 and discuss the accuracy of each one and discuss your results? This part should be submitted as a separate file from the `shopping.py` file. Combine `shopping.py` and the pdf file in one zip file and submit it.

IMPORTAT:

1. ALL Submission will be checked for plagiarism. Plagiarized submissions receive Zero scores.
2. Any submitted implementation that is not following the given specification will not be accepted.

Credit: https://cs50.harvard.edu/ai/2020/projects/4/shopping/