Introduction to Machine Learning (COMP 135)
Assignment 03 (80 points)
Due on Gradescope by 9:00 AM, Wednesday, 26 February 2020

For this assignment, you will make modifications to a Python notebook (`hw03.ipynb`) that has been supplied. You will complete the various required sections outlined below in that notebook. When you are done, generate a PDF version of that notebook, with all results (figures, printed results, etc.) included, for submission to Gradescope. You will also submit the raw notebook source, two PDF images generated by your program (but not embedded in the worksheet itself), and a `COLLABORATORS.txt` file, via a separate link, as described below.

Part Zero: Adding the `graphviz` library

Before you start the assignment, you should add one additional library to your Python install; this will allow you to visualize decision trees in graphical format. You will only need to do the following once; after these steps are performed, these tools will be available, along with all the other ones we have been using, simply by activating the environment as usual.

1. Activate your environment as you normally would:

```
conda activate ml135_env
```

2. From within the environment, update the `pip` tool:

```
pip install --upgrade pip
```

3. Install the `graphviz` library:

```
pip install graphviz
```

Once these steps are completed, you will be able to use the new tools from inside the COMP 135 environment, in notebooks and other Python code.

Part One: Logistic Regression for Cancer-Risk Screening (45 points)

You have been given a data set containing some medical history information for patients at risk for cancer.* This data has been split into various training, testing, and validation sets; each set is given in CSV form, and is divided into inputs (`x`) and outputs (`y`).

Each patient in the data set has been biopsied to determine their actual cancer status. This is represented as a boolean variable, `cancer` in the `y` data sets, where `1` means the patient has cancer and `0` means they do not. You will build classifiers that seek to predict whether a patient has cancer, based on other features of that patient. (The idea is that if we could avoid painful biopsies, this would be preferred.)
Input data has three features:

- `age`: Patient age is stored as a floating-point value, to give finer-grained detail than simply number of years.
- `famhistory`: A boolean variable indicating whether or not a patient has a family history of cancer (as usual, `1` = `true`, indicating that the family does have a cancer history).
- `marker`: A measured chemical marker that clinicians believe may have some correlation with the presence of cancer.

You will be examining and comparing classifiers that use only the first two of these features, versus ones that use all three.

1. (*2 pts.*) Complete the function `calc_TP_TN_FP_FN()`. This function should take in two vectors of the same length, one consisting of known correct output values (`0` or `1`) for a classification task, and the other consisting of the actual output values for some classifier. It will then compute the number of true/false positive/negative values found in the classifier output, and return them. This function will be used in later stages of the program; as usual, you may want to write code to test it (you do not need to include this in your final submission).

2. (*2 pts.*) For each of the input sets (train, validation, test), we want to know how the proportion of patients that have cancer. Modify the relevant section of the notebook to compute these values and them print them. Results should appear in floating point form (a value from 0.0 to 1.0), formatted as already given in the notebook.

3. (*8 pts.*) Given a known-correct outputs $(y_1, y_2, \ldots, y_N)$ one simple baseline for comparison to our classifier is a simple routine that always returns the same value. For instance, we can consider the ***always-0*** classifier, which makes the same negative prediction for all input data:

$$\forall i,\ \hat{y}(x_i) = 0$$

   (a) (2) Complete the code to compute and print the accuracy of the always-0 classifier on the validation and test sets. Results should appear in floating point form (a value from 0.0 to 1.0), formatted as already given in the notebook.

   (b) (2) Print out a confusion matrix for the always-`0` classifier on the validation set. Your code should use the supplied `calc_confusion_matrix_for_threshold` function.

*Data set credit: A. Vickers, Memorial Sloan Kettering Cancer Center https://www.mskcc.org/sites/default/files/node/4509/documents/dca-tutorial-2015-2-26.pdf

(c) (2) You will see reasonable accuracy for the simple baseline classifier. Is there any reason why we wouldn't just use it for this task? Your answer, written into the notebook as text, should give some detail of the pluses and minuses of using this simple classifier.

(d) (2) Given the task of this classification experiment—determining if patients have cancer without doing a biopsy first—what are the various errors that the always-`0` classifier can make? For each such type of mistake, what would be the ***cost*** of that mistake? (Possible costs might be, for example, lost time or money, among other things.) What would you recommend about using this classifier, given these possibilities?

4. (*12 pts.*) You will now fit logistic regression model to the data, using the `sklearn` library, in particular the `sklearn.linear_model.LogisticRegression` function:

   https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

   When creating a regression model, be sure to specify that `solver='liblinear'`, and specify the inverse penalty strength `C`. Your code should explore a range of values for this last parameter, using a regularly-spaced grid of values:

   ```
   C_grid = np.logspace(-9, 6, 31)

   for C in C_grid:
       # Build and evaluate model for each value C
   ```

   (a) (5) For each value of `C`, create a logistic model and fit it to the 2-feature input data.

   - For each model, you can use the sklearn function `predict_proba` to generate the logistic's *probabilistic predictions* for the training and validation sets.
   - Once the prediction has been made, you can compute the logistic loss between it and the correct results for the data sets, using `sklearn.metrics.log_loss`.

   Once all of the models have been fit and evaluated, create a single plot for the log-loss (y-axis) versus $\log_{10}$ `C` (x-axis) for both sets of data. In addition, print out the value of `C` for which the loss is minimized on the validation set, along with that minimal loss.

   Note: When graphing results, throughout this assignment, ensure that they are clearly labeled, and presented readably with legends and distinct line colors/types to distinguish multiple plot lines if necessary.

   (b) (3) Use the logistic model with the best value `C` found in the prior step to again make the probabilistic predictions for the validation set. You can then use those predictions, along with the correct outputs for that set to plot the overall performance of that model, using the provided function `make_plot_perf_vs_threshold`; this will produce:

   - A histogram of predicted probabilities for negative data examples.
   - A histogram of predicted probabilities for positive data examples.
   - Line plots of various performance metrics that require hard decisions (see Lecture 06 for discussion of such metrics).

(c) (4) Repeat the previous two steps, this time fitting each of the models using the 3-feature input data. Again, your code will explore a range of `C`-parameter values, plotting the logistic loss for each, determining the best such value, and then plotting the performance for the best model.

5. (*8 points*) ROC curves allow us to compare the ratio of false positive and true positives for a classifier. You can use the existing tool `sklearn.metrics.roc_curve` to plot such curves.

   (a) (4) Create a single plot showing ROC curves for the two best models from prior steps (one for 2-feature data, and one for 3-feature data), using the ***validation*** data.

   (b) (2) Create a single plot showing ROC curves for the two best models from prior steps (one for 2-feature data, and one for 3-feature data), using the ***test*** data.

   (c) (2) Analyze the results to compare the two models, over both data sets. Does one dominate the other in terms of overall performance, or are there areas where one model "wins" and others where the other model does? What does this tell you?

6. (*13 pts.*) In this part of the assignment, you will use the best 3-feature logistic model from the prior steps. You will now examine how that model would perform relative to various values of the threshold probability at which we put a data item into the positive (cancerous) class. When generating confusion matrices, use `calc_confusion_matrix_for_threshold`. You can also use `print_perf_metrics_for_threshold` to print out values of various metrics.

   (a) (2) Generate a confusion matrix for the best 3-valued classifier, setting the threshold to the default 0.5 (this means that any data item getting logistic value $x \geq 0.5$ is assigned to class `1`, indicating cancer). In addition, print out the various values of the metrics. Use the **test** data here.

   (b) (4) For the same classifier, use `compute_perf_metrics_across_thresholds` to compute a range of performance metrics across various possible settings of the classifier threshold. When you do this, use the **validation** data to find the threshold. Select the threshold that maximizes true positive rate (TPR), while at the same time achieving precision—also known as positive predictive value (PPV)—that is *at least* 0.98. Print the confusion matrix and performance metrics when using this threshold for the **test** data.

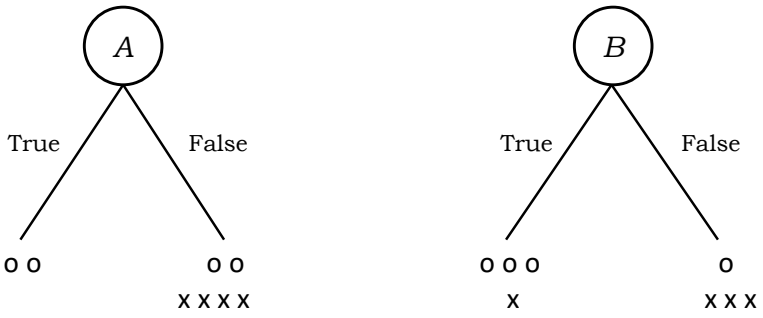
   (c) (3) Next, determine the threshold that achieves the *inverse* of the prior step. That is, we want the value that maximizes PPV, while ensuring that TPR ≥ 0.98. Again, use the validation data to find the threshold, and print the confusion matrix and performance metric values, based upon the test data.

   (d) (2) Compare the three confusion matrices just generated. Which threshold best meets the scenario of wanting to avoid life-threatening mistakes at all costs, while also eliminating unnecessary biopsy operations? Why?

   (e) (2) How many patients in the test data would be saved from unnecessary biopsies if the best threshold were used? (Remember that in this data, every patient has been biopsied to determine their cancer status, so we are asking about how many could have avoided that if your classifier was employed to screen them instead.) What fraction of biopsies would be avoided if this classifier were used by the hospital?

Part Two: Decision Trees (28 points)

You will examine the use of decision trees for classification, along with different heuristics for evaluating which features to choose when building such trees.



1. (*10 pts.*) The diagram above shows the results of splitting a simple data-set according to two binary features (A and B). The data-set consists of eight entries, of two different types (`o` and `x`). You will compute and display the results of computing the two feature-heuristics seen in the readings and in class lecture notes.

   (a) (3) Compute the values for each feature, based upon the counting heuristic discussed in the reading (Daumé). Print out the features in order from best to worst, along with the heuristic (correctness) value for that feature, using the format:

   ```
   feature_name: num_correct/total_data
   ```

   (b) (3) Compute the values for each feature, based upon the information-theoretic heuristic discussed in lecture. Print out the features in order from best to worst, along with the heuristic (gain) value for that feature, to 3 decimal places of precision, using the format:

   ```
   feature_name: information_gain
   ```

   (c) (4) Discuss the results: if we built a tree using each of these heuristics, what would happen? What does this mean?

2. (*6 pts.*) We have provided some data on abalone, a widespread shellfish.[†] The input data consists of a number of features of abalone, as shown in the following table, while the output is the number of rings found in the abalone shell:

column	type	unit	description
`is_male`	binary		`1 ==` male; `0 ==` female
`length_mm`	numeric	mm	longest shell measurement
`diam_mm`	numeric	mm	shell diameter, perpendicular
`height_mm`	numeric	mm	height of shell
`whole_weight_g`	numeric	gram	weight (entire)
`shucked_weight_g`	numeric	gram	weight (meat)
`viscera_weight_g`	numeric	gram	weight (guts)
`shell_weight_g`	numeric	gram	weight (dried shell)

[†]Original data: Warwick J. Nash, et al. (1994) https://archive.ics.uci.edu/ml/datasets/Abalone

In addition, we have supplied a simplified version of the data, where each input-feature has been converted to a binary value (either above average value for that feature (`1`), or not `0`), and the output value $y \in \{0, 1, 2\}$ no signifies a *Small*, *Medium*, or *Large* number of rings; the data has also been simplified down to only four features of the original eight. Each data-set is broken up into `x` and `y` sets already, for both training and testing.

Your code will explore these data sets, computing the two heuristics for the simplified data, and classifying both sets using decision trees.

(a) (3) Compute the counting-based heuristic for the features of the ***simplified*** abalone data. Print out the features in order, using the same format as before.

(b) (3) Compute the information-theoretic heuristic for the features of the ***simplified*** abalone data. Print out the features in order, using the same format as before.

3. (*12* pts.)

You will use the `sklearn` library for decision-trees on both versions of the abalone data-set:

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

(a) (8) For each data-set, create the classifier using the `criterion='entropy'` option, which uses the same information-theoretic heuristic discussed in lecture. After building the model, you can use its `score()` function to get its accuracy on each of the testing and training portions of the data. Print out these values, being clear which value is which.

In addition, export the two trees as PDF images, using the `export_graphviz()` and `render()` functions.[‡] When done, you should be able to open those images (they will be in the directory with your active notebook file) to examine them.

(b) (4) Discuss the results you have just seen. What do the various accuracy-score values tell you? How do the two trees that are produced differ? Looking at the outputs (leaves) of the simplified-data tree, what sorts of errors does that tree make?

[‡]See the user guide for sample code: https://scikit-learn.org/stable/modules/tree.html

Part Three: Code submission (7 points)

1. (*2 pts.*) Submit the source code (`hw03.ipynb`) to Gradescope.

2. (*2 pts.*) Submit the two PDF images of trees that were generated by your decision-tree visualizations. Each PDF file should be named to indicate which tree it shows (the full data-set or the simplified one).

3. (*3 pts.*) Along with your code, submit a completed version of the `COLLABORATORS.txt` file. An example has been provided, which you should edit appropriately to include:

   - Your name.
   - The time it took you to compete the assignment.
   - Any resources you used to complete the assignment, including discussions with the instructor, TA's, or fellow students, and any online or offline resources consulted. If you did not need to consult any outside resources, you can say so.
   - A brief description of what parts, if any, of the assignment caused you to seek help.

Submission details: Your code must work properly with the versions of Python and other libraries that are part of the COMP 135 standard environment. The class website contains instructions for installing and using this environment.

http://www.cs.tufts.edu/comp/135/resources/

You can write the Python code using whatever tools you like, but you should ensure that the code executes properly.