Introduction to Machine Learning (CS 135)
Assignment 03 (50 points)
Due on Gradescope by 11:59 PM, Saturday, 25 June 2022

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 and a `COLLABORATORS.txt` file, via a separate link, as described below.

Simple Classifiers for Cancer-Risk Screening (46 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 and testing 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.

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, 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.

*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

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 train 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 test set. Your code should use the supplied `calc_confusion_matrix_for_threshold` function.

   (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. (*10 pts.*) You will now fit a perceptron model to the data, using the `sklearn` library, in particular the `sklearn.linear_model.Perceptron`:

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

   (a) (2) Create a basic version of the model, fitting it to the training data, and making predictions on the test data. Report its accuracy on the test data, and show the confusion matrix for that data.

   (b) (2) How does the basic perceptron model compare to the always-0 classifier? What does the performance of the model indicate?

   Note: If, after doing these first two steps, the performance is essentially the same as with the always-0 classifier, then something would seem to have gone wrong (since that would mean that the perceptron model is no more effective than a "model" that requires no complex learning at all).

   One main reason that a model performs poorly is that the variables making up the data set have very different ranges, so that some variables have much stronger initial effects on potential solutions than others. Looking at our data, this is likely to be the case. If you have this issue, you should *re-scale* the data to eliminate this issue. A min-max scaling solution, using the `MinMaxScaler` from `sklearn` should help rectify things:

   https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

(c) (4) As you did in the prior assignment, you will explore regularization options with the perceptron. Create a series of such models: each should use the L2 penalty for regularization, and different values of the `alpha` parameter, selected from:

```
alphas = np.logspace(-5, 5, base=10, num=100)
```

For each such model, record the accuracy of the model on the training and test data. (Note that the model's `score()` function will do this for you, although there are other ways of getting the result.) Plot the accuracy values, relative to the various `alpha` values, plotted on a logarithmic scale.

(d) (2) What does the performance plot tell you? It will look quite different than the ones you would have seen in the prior assignment, for regression; think about these differences, and address them in your answer.

5. (*24 points*) Rather than use the basic `predict()` function of the `Perceptron`, we can use its `decision_function()` to generate *confidence scores*. While the basic predictions are always in 0/1 form, confidence scores are more nuanced—they effectively convey how certain the model is that each data-point is in the assigned class, relative to other data points.

   We can use confidence scores in a few ways. By themselves, they can be used as the basis for methods that examine different *thresholds*, classifying data-points based upon how confident we are, and making different decisions about when we should treat a particular confidence score as indicating a positive (1) class output.

   We can also convert confidence scores like this to *probabilities* of belonging to the positive (1) class, and *then* examine different thresholds. While some models, like logistic regression, can handle both discrete and probabilistic predictions, the `Perceptron` requires that we use another classifier to do so, `sklearn.calibration.CalibratedClassifierCV` (`CCCV`):

   https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

   You will convert a `Perceptron` classifier into one of these probabilistic classifiers, and then plot ROC data for both versions (ROC curves work with both probabilities and confidence values, among other things).

   positives for a classifier. You can use the existing tool `sklearn.metrics.roc_curve` to plot such curves.

   (a) (6) Create a new `Perceptron` model, and generate its `decision_function()` for the test data. Also create a `CCCV` model, using a `Perceptron` as its basis for estimation, and using the `'isotonic'` method option for best results when converting to probabilities—after building that model, generate its `predict_proba()` output for the test data.[†]

   Generate a plot that contains the ROC curves for each of these models, labeled correctly. You can generate the TPR and FPR values you need using `sklearn.metrics.roc_curve`

   https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html

[†]You may want to print out the outputs of the two functions to compare them and help you see what you are working with.

Plot the two curves in the same plot, labeling each correctly to distinguish them. Following the plot, print out the area under the curve (AUC) values for each model, using: `sklearn.metrics.roc_auc_score`:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

(b) (2) Discuss the results you see in the plots and AUC values. How do the two versions of the model differ? How are they similar? Which might you prefer, and why?

(c) (4) Once we have generated the output of a probabilistic classifier on some data, we can compute its performance relative to probability threshold T_P, where we decide to classify any data with output $P \geq T_P$ as a 1, else a 0. (So, for instance, setting $T_P = 0.0$ means that *all* data is classified as part of the positive (1) class.) Complete the function `calc_perf_metrics_for_threshold()`, which should return the various performance metrics for a probabilistic classification, given the correct outputs, and a particular probability threshold to employ.

(d) (4) Test a range of probability thresholds in:

```
thresholds = np.linspace(0, 1.001, 51)
```

For each such threshold, compute the TPR and PPV on the test data. Record the threshold at which you get highest-possible TPR while *also* achieving the highest PPV possible (that is, if there is a tie between two thresholds in terms of TPR, use higher PPV to break the tie), and also record the performance metrics at that best threshold. Do the same for the threshold with highest PPV (breaking ties using TPR).

(e) (8) Compare different values of the threshold for classification:

i. (2) Use the *default* threshold, under which a data-point is classified positive if and only if its output probability is $P \geq 0.5$. Print the confusion matrix and performance metrics for that threshold on test data.

ii. (2) Do the same for the threshold you found at which we have best TPR and highest possible associated PPV.

iii. (2) Do the same for the threshold you found at which we have best PPV and highest possible associated TPR.

iv. (2) Discuss these results. If we decided to apply each of these thresholds to our data, and then used that process to classify patients, and decide upon whether or not to do biopsies, what would the effects be, exactly?

Code submission (4 points)

1. (*2 pts.*) Submit the source code (`hw03.ipynb`) to Gradescope.
2. (*2 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 CS 135 standard environment. The class Canvas site contains instructions for installing and using this environment. You can write the Python code using whatever tools you like, but you should ensure that the code executes properly.