Introduction to Machine Learning (CS 135) Assignment 03 (50-point scale)

For this assignment, you will complete `hw3.py` and present solutions to a Python notebook (`hw3.ipynb`), both skeletons provided. This consists of various sections outlined below. When complete, generate a PDF version of that notebook with all results (figures, printed results, etc.) and submit it to Gradescope. You will also submit the raw notebook source and a `COLLABORATORS.txt` via a separate link, as described below.

Simple Classifiers for Cancer-Risk Screening (48 points)

We are given a data set containing some medical history information of patients tested for cancer [source: A. Vickers. DCA Tutorial. Memorial Sloan Kettering Cancer Center/].

The data is split into training and testing sets and divided into inputs `x` and outputs `y` in CSV format.

Each patient in the data had a biopsy to test for cancer: the target variable `y` (i.e., cancer) is represented as a boolean, where 1 means the patient has cancer and 0 means benign. The goal is to build classifiers to predict whether a patient has cancer-based on other features provided for each 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 the number of years.
- `famhistory` : A boolean variable indicating whether 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. (3 pts.) Complete the function `calc_binary_metrics()` .
   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 predictions for some classifier. It will then compute and return the number of true/false positive/negative values. This function will be used in later stages of the program.
2. (2 pts.) We want to know the proportion of cancer patients for each input set (train, test). Modify `calc_percent_cancer()` to compute this. Results should appear in floating point form as a percentage from 0.0 to 100.0,. The `print` statement is formatted already in the notebook.
3. (8 pts.) Given known-correct outputs (y1, y2, . . . , yN), 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:
   ∀i, yˆ(xi) = 0

- (2) Complete the code to classify `predict_0_always_classifier()` and calculate the accuracy `calc_accuracy()` . Results should appear in floating point form (a value from 0.0 to 1.0), formatted as already given in the notebook.
- (2) Print out a confusion matrix for the always-0 classifier on the test set via the function provided `calc_confusion_matrix_for_threshold()` .
- (2) You will see reasonable accuracy for the simple baseline classifier. Why not just use it for this task? Your answer, written into the notebook as text, should detail the pluses and minuses of using this simple classifier.
- (2) Given the task of this classification experiment—determining if patients have cancer without doing a biopsy—what are the errors 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? Be sure to speak about the binary metrics (i.e., TP, FP, TN, FN).

4. (21 pts.) Using the sklearn's sklearn.linear_model.Perceptron we will now fit a perceptron model to the data

- (1) A model performs poorly because the variables making up the data set have very different ranges, so some variables have much stronger initial effects on potential solutions than others. Looking at our data, this is likely the case. If you have this issue, you should re-scale the data to eliminate this issue. Using the MinMaxScaler from sklearn, a min-max scaling solution should help rectify things. Implement function `standardize_data()` for this.
- (3) Create a basic version of the model using default parameter values, 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.
- (4) How does the basic perceptron model compare to the always-0 classifier regarding the accuracy and binary metrics? What are the consequences of the performance of the perceptron model?
  /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).
- (8) 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 model's accuracy 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 alpha values plotted on a logarithmic scale. Make sure to show the title, legends, and axis labels.

- (5) What does the performance plot tell you? It will look quite different from the ones you would have seen; think about these differences and address them in your answer (hints: Is there a specific trend? Does it look normal? How does the performance look compared to the performance of the always-0 classifier? Is regularization helpful?).

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 confident 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. They can be used as the basis for methods that examine different thresholds, classify data points based on our confidence, and make 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).
Convert a Perceptron classifier into one of these probabilistic classifiers, then plot ROC data for both versions (ROC curves work with probabilities and confidence values, among other things). Plot such curves using the existing tool `sklearn.metrics.roc_curve` .

- (6) Create a new basic 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. Please print out the outputs of the two functions to compare them and help you see what you are working with.
  Generate a plot containing the ROC curves for each model labeled correctly. You can generate the TPR and FPR values using sklearn.metrics.roc_curve.
  Plot the two curves in the same plot, labeling each correctly to distinguish them. Following the plot, print out each model's area under the curve (AUC) values using: sklearn.metrics.roc_auc_score.
- (2) Discuss the results 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?
- (4) Once we have generated the output of a probabilistic classifier on some data, we can compute its performance relative to probability threshold T, where we decide to classify any data with output P ≥ T as a 1, else a 0. (So, for instance, setting T = 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.
- (4) Test a range of probability thresholds in: `thresholds = np.linspace(0, 1.001, 51)`
  For each threshold, compute the TPR and PPV on the test data. Record the threshold with the highest-possible TPR while also achieving the highest PPV possible (if there is a tie between two thresholds in terms of TPR, use a higher PPV to break the tie), and record the performance metrics at that best threshold. Do the same for the threshold with the highest PPV (breaking ties using TPR).
- (8) Compare different values of the threshold for classification:
  1. (2) Use the default threshold, under which a data point is classified positive if and only if its output probability is P ≥ 0.5. Print the confusion matrix and performance metrics for that threshold on test data.
  2. (2) Do the same for the threshold with the best TPR and the highest possible associated PPV.
  3. (2) Do the same for the threshold you found at which we have the best PPV and the highest possible associated TPR.
  4. (2) Discuss these results. What would the effects be if we decided to apply each of these thresholds to our data and then used that process to classify patients and decide whether to do biopsies?

Code submission (2 points)

1. (2 pts.) Submit the source code (`hw3.ipynb` and `hw3.py`) to Gradescope, along with the generated PDF.
2. (+1 pts. BONUS) Along with your code, submit a completed version of the `COLLABORATORS.txt` file with:
   - Your name.
   - The time it took you to complete the assignment.
   - Any resources you used to complete the assignment, including discussions with the instructor, Tasks, or fellow students, and any online or offline resources consulted. You could say that you did not need to consult any outside resources.
   - A brief description of what parts, if any, of the assignment caused you to seek help.
3. Submission details: Your code must work properly with the versions of Python and other libraries that are part of the CS 135 standard environment. 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 the code executes correctly.