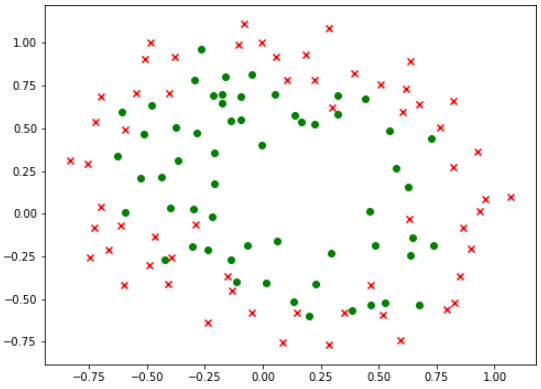
Project 3: Classification with Logistic Regression and SVM

Due: Before 11:59 pm on Mar 12, 2022

Problem Description

For this project we will apply both Logistic Regression and SVM to predict whether capacitors from a fabrication plant pass quality control based (QC) on two different tests. To train your system and determine its reliability you have a set of 118 examples. The plot of these examples is show below where a red x is a capacitor that failed QC and the green circlesrepresent capacitors that passed QC.



I have already randomized the data into two data sets: a training set of 85 examples and a test set of 33 examples. Both are formatted as

- First line: m and n, tab separated
- Each line after that has two real numbers representing the results of the two tests, followed by a 1.0 if the capacitor passed QC anda 0.0 if it failed QC—tab separated.

Assignment: Your assignment is to use what you have learned from the class slides and homework to create (from scratch in Python, not by using Logistic Regression library function!) a Logistic Regression and SVM binary classifier to predict whether each capacitor in the test set will pass QC.

Logistic Regression: You are free to use any model variation and any testing or training approach we have discussed for logistic regression. In particular, since this data is not linear, I assume you will want to add new features based on power of the original two features to create a good decision boundary. $w_0 + w_1x_1 + w_2x_2$ is not going to work!
One choice might be
$w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7 + w_8x_8$ where the new features are created as follows:

New Features	From Original Features
x_1	x_1
x_2	x_1^2
x_3	x_2
x_4	x_1x_2
x_5	$x_1x_2^2$
x_6	x_2^2
x_7	$x_1^2x_2$
x_8	$x_1^2x_2^2$

Note that it is easy to create a small Python program that reads in your original features, uses a nested loop to create the new features and then writes them to a file.

```
thePower = 2
for j  in  range(thePower+1):
    for i in range(thePower+1):
        temp = (x1**i)*(x2**j)
        if (temp != 1):
            fout1.write(str(temp)+"\t")
fout1.write(str(y)+"\n")
```

With a few additions to the code, you can make a program to create combinations of any powers of x_1 and x_2!

SVM: You need to use the original training and testing data file with kernel functions for SVM. You **can** use the svm functions in the *Scikit-learn* library and don't need to implement the algorithm from scratch.

Please refer to https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html for details

What to Upload to Canvas:

Logistic Regression:

1. ***A single py file (lastname_firstname _P3_LR.py)*** that prompts for a training file name, computes weights using gradient descent, prints out a plot of iterations vs. J, plot the decision boundary on the whole dataset and then prompts for a test filename, and using the computed weights prints out final J, FP, FN, TP, TN, accuracy, precision, recall and F1 for the test set. All values should be clearly labelled.
2. ***Your training set file (lastname_firstname_P3Train.txt).*** First line should contain integers m and n,tab separated. Each line after that should have n real numbers representing the new feature data, followed by a 1 if the capacitor passed QC and a 0 if it failed QC—tab separated.
3. ***Your test set file (lastname_firstname_P3Train.txt).*** First line should contain integers m and n, tabseparated. Each line after that should have n real numbers representing the new feature data, followed by a 1 if the capacitor passed QC and a 0 if it failed QC—tab separated.
4. ***A pdf file (lastname_firstname_P3_LR.pdf)*** that includes
   - A description of your model and testing procedure, including
     - Description of your model
     - Initial values that you chose for your weights, learning rate, and the initial value for ***J***.
     - Final values for learning rate, your weights, how many iterations your learning algorithm went through and your final value of ***J*** on your training set.
     - Include a plot of ***J*** (vertical axis) vs. number of iterations (horizontal axis).
     - Include a plot of hyperplane on the whole dataset
     - Value of ***J*** on your test set.
     - Your code
   - A confusion matrix showing your results on your test set.
   - A description of your final results that includes accuracy, precision, recall and F1 values.

SVM:

1. ***A single py file (lastname_firstname _P3_SVM.py)*** that prompts for a training file name, plot the margin and hyperplane, and then prompts for a test filename, and using the computed weights prints out final FP, FN, TP, TN, accuracy, precision, recall and F1 for the test set. All values should be clearly labelled.
2. ***A pdf file (lastname_firstname_P3_SVM.pdf)*** that includes
   - A description of your model and testing procedure, including
     - Description of your model
     - Description of your kernel function
     - Include a plot of ***margin and hyperplane***
     - Your code
   - A confusion matrix showing your results on your test set.
   - A description of your final results that includes accuracy, precision, recall and F1 values.

Note:

For undergrads (CPSC 4430) the final accuracy of both algorithms on your test set should be higher than 70%

For graduate-level (CPSC 6430) the final accuracy of both algorithms on your test set should be higher than 85%

Do not assume that any files are available to you besides files you turn in!

Zip your files into one zip file named ***lastname_firstname_P3_midterm.zip*** and upload it to Canvas.