EE 559 **Homework 4** Posted: Thur., 2/16/2023
Jenkins/Chugg Due: Fri., 2/24/2023, 11:59 PM

Notes for computer problems:

(1) For a given dataset in a classification problem:

- **Accuracy** {**error rate**} is defined as number of correctly {incorrectly} classified data points divided by the total number of data points, respectively.
- **Unclassified rate** is defined as number of unclassified data points divided by the total number of data points.

By these definitions [accuracy + error rate + unclassified rate] = 100%.

(2) Please refer to Discussion 5 notes for tips on structuring your code generally.

(3) For the dry bean dataset (Pr. 1-2) and the synthetic datasets (Pr. 4), **standardize the data set**s before using them. For this you may use:

sklearn.preprocessing.StandardScaler()

(4) For this homework assignment, you are to **write the code yourself.** You may use only Python built-in functions, NumPy, matplotlib; you may use the PlotDecBoundaries.py or similar functions provided with previous homeworks; the provided PlotNonlinear.py function for Pr. 4, and you may use pandas only for reading and/or writing csv files. In addition, any sklearn functions that are stated in this assignment may also be used.

(5) Also see the piazza post use of sample class code that describes using code from lecture, discussion, or Prof. Chugg's GitHub postings. This applies to all homeworks and the class project.

Notation for all problems: underbars are omitted in this homework assignment; all *x* and all *w* below are vectors. W is a matrix consisting of the C weight vectors as columns ($w_k = k^{th}$ column of W). And boldface **C** is a confusion matrix.

---

1. Multiclass perceptron. In this problem you will code and run a multiclass perceptron classifier on a "Dry Beans" dataset. More information on the dataset can be found here. For this problem, be sure to use the provided train and test datasets so that everyone is using the same dataset split; do not start from the UCI website and divide it yourself into train and test sets. To read the data provided, you can use:

```
df_train = pd.read_csv('Dry_Bean_train.csv')
X_train = df_train.drop("Class", axis=1)
Y_train = df_train['Class']
```

Tip: in general, it's a good idea to do some exploratory data analysis (EDA) on the training data before running or coding classifiers or regressors. For example, EDA on Dry Bean

Dataset is a notebook that was created by Thanos and used in Discussion 5. It is already set up to read the Dry Bean dataset (but extracts its own train and test sets).

Nothing is required to turn in from the EDA; it's for your own information and experience.

For this homework problem, code a multiclass perceptron learning and prediction algorithm, for *D* dimensions and *C* classes. Use the true multiclass perceptron algorithm given in lecture. (Do not use OvR or OvO for this problem.)

Algorithm notes: use stochastic GD variant 1 (shuffling data at the beginning of each epoch). For initial weight vector use $w(0) = (1,1,1\ldots,1)$, and for the halting condition use a simple one that always halts after 100 epochs For the final weight vectors $\widehat{W}$, compute $J(W)$ for each of the last 100 iterations in the last epoch, and pick among those the best W (corresponding to minimum $J(W)$). If there is a tie for minimum value of *J*, pick the last W of those that tied.

Tip to help you verify your code: You should expect the test accuracy to be approximately 90%.

(a) **Run your Multiclass Perceptron algorithm** on the Dry Beans dataset.

Report the following:

- classification accuracy on the training set and test set.
- final (augmented) weight vectors w_k, $k = 1,2,\cdots,C$ and their magnitudes $||w_k||$, $k = 1,2,\cdots,C$.
- the confusion matrix[1] on the test set.

Note:

[1] A confusion matrix C for a C-class problem is a $C \times C$ matrix with entries $C_{ij} = \{$number of data points labeled as class i that are classified by the ML system as class $j\}$.

Tip: For the confusion matrix you can use sklearn (see sample code provided below):

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(<labels>, <your predictions>)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.figure()
plt.show()
```

(b) Repeat part (a) 10 times. Report only the mean and standard deviation of: classification accuracies, weight vector norms $||w_k||$, $k = 1,2,\cdots,C$, and confusion matrix. (For the confusion matrix, give a mean matrix (each entry is the mean of 10 runs), and a standard deviation matrix (each entry is the standard deviation of 10 runs). No need to report individual results from each run.

2. Code up a one vs. rest (OvR) classifier, for a *C*-class problem with *D* features, that can call 2-class fit and predict methods for each 2-class classifier.

Use a 2-class perceptron to train each 2-class problem. Use the same parameters given in Problem 1 "Algorithm notes" above. For each 2-class problem, store the optimal weight vector $\widehat{w}_k$ and the class assignment for each data point (e.g., +1 for S_k' or -1 for $\overline{S_k'}$).

(a) Run the 2-class problems. Store the resulting weight vectors w_k (will be used in (b)-(d) below). For the result of each 2-class problem, report the following:

– A histogram plot (one plot for each 2-class problem). For the problem S_k' *vs.* $\overline{S_k'}$, the histogram plot should show the number of test data points oringinally labeled S_k in one color, and the number of test data points oringinally labeled S_j, for all $j \neq k$, in a different color. (Translucent colors work best.) Both of these histogram values should be plotted *vs.* value of $g(x)$ (using non-reflected *x*), on the same plot. See Notebook, Notebook2 for examples.

– The classification accuracy for the 2-class problem on the training set and on the test set.

The 2-class results (weight vectors stored above) will be used for all 3 combining methods below.

For combining the results of the *C* 2-class classifiers, you will use 3 different methods:

(i) After OvR training, use the final default decision rule given in lecture:

$$x \in \Gamma_k \text{ iff } x \in \Gamma_k' \text{ and } x \in \overline{\Gamma_j'} \ \forall j \neq k$$

and points in indeterminate regions are left unclassified.

(ii) After OvR training, use MVM decision rule 1 as follows, to classify all data points:

$$\text{Assign } x_n \text{ to class } S_k \text{ iff } k = \text{argmax}_k \left(g_k(x)\right)$$

(iii) After OvR training, use MVM decision rule 2 as follows, to classify all data points:

$$\text{Assign } x_n \text{ to class } S_k \text{ iff } k = \text{argmax}_k \left(\frac{g_k(x)}{||w_k||}\right)$$

in which w_k in the denominator is non-augmented.

(b)-(d) Code and run each combining method.

Tip: use the stored weight vectors from (a) to apply rules (i), (ii), (iii).

For each method, report the following performance measures separately on the training set and testing set:

– classification accuracy

– error rate

– unclassified rate

(e) Compare the results of (b)-(d). Try to explain similarities and differences you observe.

3. This problem is to be done by hand.

In this problem, parts (a) and (b) ask about different scenarios, but both parts are on the topic of OvR and multiclass classification.

(a) In a *C*-class problem, suppose we use an OvR approach to train the 2-class sub-problems, and then use the following decision (fusion) rule:

$$x \in \Gamma_k \text{ iff } x \in \Gamma_k^{'} \text{ and } x \in \overline{\Gamma_j^{'}} \quad \forall j \neq k \qquad (*)$$

which generally results in some indeterminate regions.

To classify points x_n that are in indeterminate regions given by $x_n \in \overline{\Gamma_j^{'}} \quad \forall j$ (no winning class), suppose we then apply the following decision (fusion) rule:

$$x_n \text{ is in } \Gamma_k \quad \text{iff} \quad k = \text{argmax}_k \left(\frac{g_k(x)}{||w_k||} \right).$$

in which non-augmented notation is used.

Will this assign x_n to the same class as the decision region closest to x_n (in feature space)? If yes, prove it; if no, justify why not.

Tip: use non-augmented notation for consistency with our distance expressions.

(b) In this part you will compare 2 different approaches to decision (fusion) rules in OvR.

Approach (i): After training the 2-class subproblems using an OvR approach, you use the decision (fusion) rule given in Eq. (*) in part (a). Then, you classify points x_n that are in any indeterminate region by applying the following decision (fusion) rule:

$$x_n \text{ is in } \Gamma_k \quad \text{iff} \quad k = \text{argmax}_k \left(g_k(x) \right).$$

Approach (ii): After training the 2-class subproblems using an OvR approach, you apply this one decision (fusion) rule for points anywhere in feature space:

$$x_n \text{ is in } \Gamma_k \quad \text{iff} \quad k = \text{argmax}_k \left(g_k(x) \right)$$

Yes, it's the same decision rule as the previous one, except this time it is applied to all of feature space.

Are Approach (i) and Approach (ii) equivalent (in other words, will they result in the same final decision regions and boundaries)? Justify your answer. Your justification should be based on reasoning and theory, not computational experiments. (Although feel free to follow up your answer with computational trials on your own if you like.)

Hint: compare the rule last decision rule above, to decision rule (*), in the regions that (*) does result in a final class assignment ("determinate" regions).

4. Code up a nonilnear polynomial transformation that can be performed on a dataset file with *N* data points and *D* (original) features. Do this for:

(i) A general quadratic polymonial transformation (including all terms)

(ii) A general cubic polymonial transformation (including all terms)

Tip 1: You may consider simply using for loops to obtain all the combinations of features. For example:


```
for i in range(Dim_feature):
    for j in range(i):
        …
```

You are able to get all quadratic combinations of features without duplicates, namely

$$(x_0, x_0), (x_0, x_1), (x_0, x_2), \cdots, (x_1, x_1), (x_1, x_2), \cdots.$$

For this problem you will use dataaset1, dataset2 and dataset3 from HW1. These are 2-class sythetic datasets each with 2 features.

Tip 2: for plotting points and the resulting decision regions in the original feature space, you may use the provied routine PlotNonlinear.py.

(a) Linear classification. Run a linear 2-class perceptron classifier on each training dataset. Use the same parameters given in Problem 1 "Algorithm notes" above. For each dataset, give the resulting classification accuracy on both the training and testing sets. Plot the resulting decision regions along with the training data points, and in a separate plot show the decision regions along with the testing data points.

(b) Repeat part (a), except run it 10 times. Report only the mean and standard deviation of the training-set accuracy and testing-set accuracy. No plots.

(c) Nonlinear-quadratic classification. Now for each dataset: run the quadratic polynomial transformation on the dataset, and then run the 2-class (linear) perceptron learning on the transformed training data. Give the resulting classification accuracy on both the training and testing sets. Plot in the original 2D feature space, the resulting decision regions along with the training data points, and in a separate plot show the resulting decision regions along with the testing data points.

Tip: refer to Discussion 5 for a simple numerical example (done by hand).

(d) Repeat part (c), except run it 10 times. Report only the mean and standard deviation of the training-set accuracy and testing-set accuracy. No plots.

(e) Nonlinear-cubic classification. Repeat part (c) except for a cubic polynomial transformation.

(f) Repeat part (e), except run it 10 times. Report only the mean and standard deviation of the training-set accuracy and testing-set accuracy. No plots.

(g) Explain using words and mathematical expressions, how the PlotNonlinear.py routine finds the final decision regions as mapped back into the original feature space.

(h) Compare your results of (a)-(f) (e.g., compare plots to plots; mean (and std.) accuracies to mean (and std.) accuracies). You can use standard deviations to gauge whether small differences in mean accuracies are statistically significant. Try to explain what you observe.