HMC CS 158, Spring 2018
Problem Set 4 Programming: Logistic Regression, Perceptron

Goals:
- To investigate what is and is not guaranteed by perceptron theory.
- To statistically compare the performance of different classifiers.
- To investigate how feature preprocessing might affect performance.

For this assignment, you can work individually though you are encouraged to work with a partner. You should sign up for partners on Canvas (People → PS4 Groups). If you are looking for a partner, try Piazza. If, after trying Piazza, you are having trouble finding a partner, e-mail Jessica.

Submission

You should submit any answers to the exercises in a single file `writeup.pdf`. This writeup should include your name and the assignment number at the top of the first page, and it should clearly label all problems. Additionally, cite any collaborators and sources of help you received (excluding course staff), and if you are using slip days, please also indicate this at the top of your document.

Your code should be commented appropriately (though you do not need to go overboard). The most important things:
- Your name and the assignment number should be at the top of each file.
- Each class and method should have an appropriate doctsring.
- If anything is complicated, it should include some comments.

There are many possible ways to approach the programming portion of this assignment, which makes code style and comments very important so that staff can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

When you are ready to submit, make sure that your code compiles and remove any debugging statements. Then rename the top-level directory as `<username1>_<username2>_titanic`, with the usernames in alphabetical order (e.g. `hadas_yjw_titanic`). This directory should include the electronic version of your writeup and main code, any files necessary to run your code (including any code and data provided by staff), and follow the same structure as the assignment directory provided by staff. So, for this assignment, your directory should have the following structure:
- `<username1>_<username2>_ps4/`
  - `data/`
    - `perceptron_data.csv`
    - `phoneme_train.csv`
  - `source/`
    - `phoneme.py` (with your modifications)
    - `perceptron.py` (with your modifications)
    - `util.py`
  - `writeup.pdf`
  - `phoneme.pdf` (pdf printout of `phoneme.py`)
  - `perceptron.pdf` (pdf printout of `perceptron.py`)

Parts of this assignment are adapted from course material by Tommi Jaakola (MIT).

Package this directory as a single `<username1>_<username2>_ps4.zip` file, and submit the archive. Additionally, to aid the staff in grading, submit your `pdf`'s as separate files.

1 Perceptron [8+2 pts]

In this problem, we will investigate properties of the perceptron, in particular, exploring how the perceptron algorithm behaves depending on various algorithm options.

---

code and data

- code : `perceptron.py`
- data : `perceptron_data.csv`

The perceptron algorithm is remarkably simple yet it comes with a mistake guarantee. The guarantee is not statistical in nature but applies to all sequences of examples (and labels) provided that there exists a linear reference classifier (the maximum-margin classifier, whose parameters are unknown to us) that can correctly classify all the examples in the sequence with the specific margin.

(a) **(4 pts)** We are interested in investigating the number of mistakes made by the perceptron algorithm. Since `scikit-learn`'s `Perceptron` class does not keep track of mistakes, we will have to roll out our own classifier. Implement the missing portions of `Perceptron.fit(...)`.

Hint: To check your implementation, you may want to work out the small example by hand and compare the results against your code.[1]

(b) **(2 pts)** Let us now consider a larger data set (`perceptron_data.csv`). Train the perceptron classifier (without offset) on the provided data set using two different initializations: $\boldsymbol{\theta}^{(0)} = (0,0)^T$ and $\boldsymbol{\theta}^{(0)} = (1,0)^T$. The two training procedures should traverse the data points in the same order and run until convergence.

What are your trained coefficients $\boldsymbol{\theta}^*$? Do the two training procedures converge to the same solution? Why or why not? Will both classifiers have the same performance on the training set? What about on a new, held-out test set?

(c) **(2 pts)** Compare the value of the mistake bound to the number of mistakes that you are getting *on this data set*. Explain why there might be a difference.

Hint: The mistake bound only depends on the margin γ^* and $R = \max_{i=1,\ldots,n} ||\boldsymbol{x}^{(i)}||_2$. (We have provided code to compute the margin γ^* for a maximum-margin classifier on this data.)

(d) **(+2 pts)** *Extra Credit*: Describe an adversarial procedure for selecting the order and value of labeled data points so as to increase the number of mistakes the perceptron algorithm makes before converging. Assume that the data are linearly separable.

Hint: There are several approaches here. You will get full credit if you can provide one way to increase the number of mistakes and justify your approach.

[1]Alternatively, you can compare against `scikit-learn`'s `Perceptron` class. If you use `scikit-learn`, make sure to refer to the documentation and use the correct model parameters!

2 Perceptron vs Logistic Regression [12 pts]

In this problem, we will apply two different classifiers to the problem of phoneme classification, an important sub-problem of Automatic Speech Recognition (ASR). The task of phoneme classification is to predict the phoneme label of a phoneme articulation given its corresponding voice signal. In this problem, we will try to use linear classifiers to distinguish the two kinds of vowel phonemes "aa" (as in "bot") and "ae" (as in "bat") in an acoustic-phonetic corpus called TIMIT[2].

---

code and data

- code : `phoneme.py`
- data : `phoneme_train.csv`

documentation

- Dummy (Baseline) :
  http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html
- Perceptron :
  http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html
- Logistic Regression :
  http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- K-Fold Cross-Validation :
  http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
- Preprocessing :
  http://scikit-learn.org/stable/modules/preprocessing.html

We are given 300 training examples of "aa" and "ae" (150 from each class) from TIMIT, and for each training example, a 61-dimensional real-valued feature vector[3] is computed from the voice signal. The examples of "aa" are labeled as $+1$ and examples of "ae" are labeled as -1.

In this problem, we would like to compare the performance of different classifiers and preprocessing methods. Note that for this problem, the starter code purposely has less helper text to encourage you to explore `numpy` and `scikit-learn` documentation. For all classifiers, **be sure to use the correct parameters**! (The default settings may not be what you want.)

(a) **(2 pts)** Before we start analyzing performance, let us understand the data (at a very high level). Run the perceptron algorithm (with offset) on this data. Is the training data linearly separable? How do you know? Why is this not surprising?

(b) **(1 pts)** To obtain a good estimate of the each classifier's performance, we will average over multiple trials of k-fold cross-validation. As a reminder of the experimental procedure:

> For each trial, split the training data randomly into k folds, choose one to be the "test" fold and train the classifier on the remaining $k-1$ folds. Then, evaluate the trained model on the held-out "test" fold to obtain its performance. Repeat this

[2] https://catalog.ldc.upenn.edu/LDC93S1

[3] Unfortunately, the large dimension of the data set prevents easy visualization, but we will learn dimensionality reduction techniques later in the course to aid us in this task.

process until each fold has been the test fold exactly once, then advance to the next trial.

Implement `cv_performance_one_trial(...)` and `cv_performance(...)` according to the provided specifications.

We will compare the performances of three classifiers:

- (D) dummy that predicts the most frequent label in the training set
- (P) perceptron, with offset, without regularization
- (L) logistic regression, with offset, without regularization

To compare performance, we will compute statistics across 10 trials[4] of 10-fold cross-validation:

accuracies

classifier	μ	σ
no preprocessing		
P		
L		
R		
with standardization		
P		
L		
R		

***p*-values**

		no preprocessing			standardization		
		P	L	R	P	L	R
no preprocessing	P	-					
	L	-	-				
	R	-	-	-			
standardization	P	-	-	-	-		
	L	-	-	-	-	-	
	R	-	-	-	-	-	-

(c) **(3 pts)** Start by comparing the classifiers on the raw features (i.e. without feature preprocessing).

  i. **(1 pts)** To properly make comparisons, we have to make certain that they are trained and tested on exactly the same subsets of the data on each trial/fold. Create these folds, being certain to shuffle the data at the start of each trial but never within a trial.

  ii. **(1 pts)** Report the mean and standard deviation of prediction accuracy for the different classifiers (up to three significant figures, computing statistics across all trials and folds).

  iii. **(1 pts)** For each pairwise combination of classifiers, calculate their t-test score to figure out if the differences are significant. Remember to use a two-tailed paired t-test (which can be computed using `scipy.stats.ttest_rel(...)`). To ease visualization, highlight any significant differences[5].

[4] In practice, we typically use many more trials, e.g. $n = 100$ or $n = 1000$ trials. But, for the sake of time, we will use $n = 10$ here.

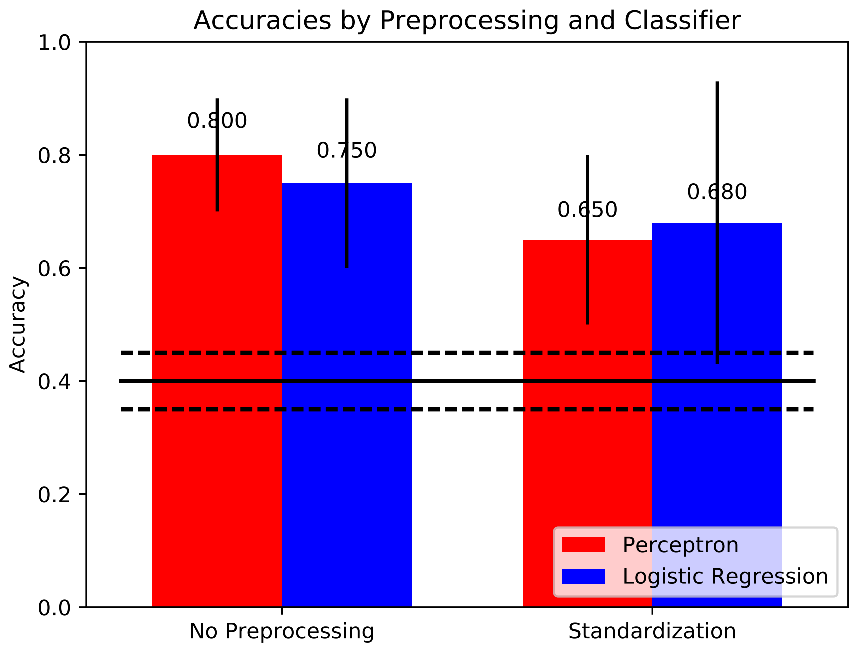
[5] Use a threshold of $\alpha = 0.05/15 = 3.33 \times 10^{-4}$. Because we have multiple comparisons, it becomes more likely that any comparison will differ by random choice alone. Thus, we must perform *multiple hypothesis correction*. Here, we use simple Bonferroni correction: If we are testing m hypotheses, then we test each individual hypothesis at a statistical significance level of $1/m$ times what it would be if only one hypothesis were tested. Though technically, since the multiple t-tests are not independent from one another, we should be using ANOVA, which generalizes the t-test to more than two groups.

(d) **(2 pts)** Next, let us investigate the effect of data preprocessing, in particular *feature standardization*, on classifier performance. Feature standardization transforms the data by removing the mean value of each feature, then scaling it by dividing non-constant features by their standard deviation.[6]

Use the `preprocessing` package from `scikit-learn` to standardize the features. Then repeat your experiments above.

(e) **(4 pts)** Finally, using the pyplot documentation, make a bar graph summarizing your findings. Your graph should use the template below (dummy values here; your results will be different). Your graph does not have to look exactly like the template – the goal is to visualize the differences between the performances of your various classifiers. The horizontal lines represent the mean and error bar ($\mu \pm \sigma$) for the dummy classifier. Be sure to label your axes and provide a legend.

Which data preprocessing method and classifier performs best? Were there any surprises?



[6] *Implementation Note*: When standardizing the features, it is important to store the values used for standardization – the mean value and the standard deviation used for the computations. After learning the parameters from the model, we often want to predict on a held-out test dataset. Given new features, we must first normalize these features using the mean and standard deviation that we had previously computed from the training set.