CS57300: Assignment 3

Due date: Friday October 30, 2020, 11:59pm (submit via Brightspace)

Comparing Methods for Speed Dating Classification

In this programming assignment, you will be asked to implement Logistic Regression and Linear SVM for the classification task that you explored in Assignment 2, and then compare the performance of different classifiers.

Similar as that in Assignment 2, you must design and implement your **own versions** of the algorithm in Python for this assignment. **DO NOT** use any publicly available code including libraries such as **sklearn**. Your code will be checked against public implementations. In addition, we will not provide separate testing data to you. You are asked to design your own tests to ensure that your code runs correctly and meets the specifications below. **Note:** You may use the **pandas, numpy, scipy** libraries for data processing purposes. The only restriction is that you have to write your **own version** of data mining algorithms; you can not use any built-in functions for your algorithm. This is a general rule for this assignment and all the upcoming ones as well. As before, you should submit your typed assignment report as a pdf along with your source code file.

In the following sections, we specify a number of steps you are asked to complete for this assignment. **Note that all results in sample outputs are fictitious and for representation only**.

1 Preprocessing (4 pts)

Consider the data file **dating-full.csv** that you used in Assignment 2. For this assignment, we will only consider the first 6500 speed dating events in this file. That is, you can discard the last 244 lines of the file. Write a Python script named **preprocess-assg3.py** which reads the first 6500 speed dating events in **dating-full.csv** as input and performs the following operations.

(i) Repeat the preprocessing steps 1(i), 1(ii) and 1(iv) that you did in Assignment 2. (You can reuse the code there and you are not required to print any outputs.)

(ii) For the categorical attributes **gender**, **race**, **race_o** and **field**, apply one-hot encoding. Sort the values of each categorical attribute **lexicographically/alphabetically** before you start the encoding process, and set the last value of that attribute as the reference (i.e., the last value of that attribute will be mapped to a vector of all zeros).

You are then asked to print as outputs the mapped vectors for 'female' in the **gender** column, for 'Black/African American' in the **race** column, for 'Other' in the **race_o** column, and for 'economics' in the **field** column.

- Expected output lines:
  Mapped vector for female in column gender: [vector-for-female].
  Mapped vector for Black/African American in column race: [vector-for-Black/African American].
  Mapped vector for Other in column race_o: [vector-for-other].
  Mapped vector for economics in column field: [vector-for-economics].

Additional note on one-hot encoding: The key point is that you will transform a categorical variable with n unique values into $n-1$ binary variables (and you can think of this as mapping one value to a vector of binary values). The vector we ask you to print is precisely the sequence of $n-1$ binary variable values that correspond to the specified value of a categorical variable. You don't need to do anything with the reference vector. It is just a way for us to make sure that all students use the same categorical variable value as the reference, so the encoding result will be the same across all students.

(iii) Use the **sample** function from **pandas** with the parameters initialized as **random_state = 25, frac = 0.2** to take a random 20% sample from the entire dataset. This sample will serve as your test dataset, which you should output in **testSet.csv**; the rest will be your training dataset, which you should output in **trainingSet.csv**. (Note: The use of the random_state will ensure all students have the same training and test datasets; incorrect or no initialization of this parameter will lead to non-reproducible results).

In summary, below are the sample inputs and outputs we expect to see. We expect 4 lines of outputs (the outputs below are fictitious) as well as two new .csv files (**trainingSet.csv** and **testSet.csv**) produced:

```
$python preprocess-assg3.py
Mapped vector for female in column gender:  [1]
Mapped vector for Black/African American in column race:  [0 0 1 0 0]
Mapped vector for Other in column race_o:  [0 0 0 1 0]
Mapped vector for economics in column field:  [0 0 0 0 0 0 0 0]
```

2 Implement Logistic Regression and Linear SVM (16 pts)

Please put your code for this question in a file called **lr_svm.py**. This script should take three command-line-arguments as input as described below:

1. *trainingDataFilename*: the set of data that will be used to train your algorithms (e.g., **trainingSet.csv**).
2. *testDataFilename*: the set of data that will be used to test your algorithms (e.g., **testSet.csv**).
3. *modelIdx*: an integer to specify the model to use for classification (LR= 1 and SVM= 2).

Note: Please, refer to the lecture slides on Brightspace for the pseudocode of these algorithms rather than referring to other online sources. Also, when implementing Gradient Descent, DO NOT implement stochastic gradient descent, and make sure to follow the values given for the parameters to be used by each algorithm as described below:

(i) Write a function named ***lr(trainingSet, testSet)*** which takes the training dataset and the testing dataset as input parameters. The purpose of this function is to train a logistic regression classifier using the data in the training dataset, and then test the classifier's performance on the testing dataset.
Use the following setup for training the logistic regression classifier: (1) Use L2 regularization, with $\lambda = 0.01$. Optimize with gradient descent, using an initial weight vector of all zeros and a step size of 0.01. (2) Stop optimization after a maximum number of iterations $max = 500$, or

when the L2 norm of the difference between new and old weights is smaller than the threshold $tol = 1e-6$, whichever is reached first. Print the classifier's accuracy on both the training dataset and the testing dataset (rounded to two decimals).

(ii) Write a function named ***svm(trainingSet, testSet)*** which takes the training dataset and the testing dataset as input parameters. The purpose of this function is to train a linear SVM classifier using the data in the training dataset, and then test the classifier's performance on the testing dataset.
Use the following setup for training the SVM: (1) Use hinge loss. Optimize with subgradient descent, using an initial weight of all zeros, a step size of 0.5 and a regularization parameter of $\lambda = 0.01$. (2) Stop optimization after a maximum number of iterations $max = 500$, or when the L2 norm of the difference between new and old weights is smaller than the threshold $tol = 1e-6$, whichever is reached first. Print the classifier's accuracy on both the training dataset and the testing dataset (rounded to two decimals).

The sample inputs and outputs we expect to see are as follows (the numbers are fictitious):

```
$python lr_svm.py trainingSet.csv testSet.csv 1
Training Accuracy LR: 0.71
Testing Accuracy LR: 0.68

$python lr_svm.py trainingSet.csv testSet.csv 2
Training Accuracy SVM: 0.75
Testing Accuracy SVM: 0.74
```

3 Learning Curves and Performance Comparison (10 pts)

In this part, you are asked to use incremental 10-fold cross validation to plot learning curves for different classifiers, with training sets of varying size but constant test set size. You are then asked to compare the performance of different classifiers given the learning curves. The only dataset you should use in this part is **trainingSet.csv**. Put your code for this question in a file named **cv.py**.

(i) Use the **sample** function from **pandas** with the parameters initialized as **random_state = 18, frac = 1** to shuffle the training data (i.e., data in **trainingSet.csv**). Then partition the training data into 10 disjoint sets $\mathbf{S} = [S_1, ..., S_{10}]$, where S_1 contains training samples with index from 1 to 520 (i.e., the first 520 lines of training samples after shuffling), and S_2 contains samples with index from 521 to 1040 (i.e., the second 520 lines of training samples after shuffling) and so on. Each set has 520 examples.

(ii) For each $t\_frac \in \{0.025, 0.05, 0.075, 0.1, 0.15, 0.2\}$:

(a) For $idx = [1..10]$

i. Let $test\_set = S_{idx}$.

ii. Let $\mathbf{S}_C = \bigcup \mathbf{S}_{i=[1..10], i \neq idx}$.

iii. Construct $train\_set$ by taking a random t_frac fraction of training examples from $\mathbf{S}_C$. Use the **sample** function from **pandas** with the parameters initialized as **random_state = 32, frac = t_frac** to generate this training set.

iv. Learn each model (i.e., NBC, LR, SVM) from *train_set*. Feel free to use your NBC implementation in Assignment 2 here (Just use your own version from assignment 2).

v. Apply each of the learned model to *test_set* and measure the model's accuracy.

(b) For each model (i.e., NBC, LR, SVM), compute the average accuracy over the ten trials and its *standard error*. Standard error is the standard deviation divided by the square root of the number of trials (in our case it's 10). For example, for a sequence of numbers $\mathbf{L} = [0.16, 0.18, 0.19, 0.15, 0.19, 0.21, 0.21, 0.16, 0.18, 0.16]$, the standard deviation of $\mathbf{L}$ is $\sigma_{\mathbf{L}} = 0.021$, and the standard error is:

$$sterr_{\mathbf{L}} = \frac{\sigma_{\mathbf{L}}}{sqrt(num\_trials)} = 0.007$$

(iii) Plot the learning curves for each of the three models in the same plot based on the incremental 10-fold cross validation results you have obtained above. Use x-axis to represent the size of the training data (i.e., $t\_frac * |S_C|$) and y-axis to represent the model accuracy. Use error bars on the learning curves to indicate ± 1 standard error.

(iv) Formulate a hypothesis about the performance difference between at least two of the models (Any pair of the 3 models can be used to form your hypothesis).

(v) Test your hypothesis and discuss whether the observed data support your hypothesis (i.e., are the observed differences significant).

Submission Instructions:

Submit through Brightspace. Please submit the report file and the source code files separately. You should submit one pdf file and one zip file through Brightspace.

1. Include in your report which version of Python you are using.
2. Make sure you include in your report all the output / results you get from running your code for all sub-questions. You may include screen shots to show them.
3. Make a directory named $yourFirstName\_yourLastName\_HW3$ and copy all of your files to this directory.
4. **DO NOT** put the datasets into your directory.
5. Make sure you compress your directory into a **zip folder** with the same name as described above, and then upload your zip folder to Brightspace.
6. Make sure to use any extension days used in your report
7. We should not need to make any edits to be able to run your codes (including file paths, assume everything will reside in the root folder where your code files are).
8. Do not make any changes in your dataset, you are not submitting your dataset. We will be using the original version of the dataset to run your codes.
9. Stick to the command line arguments required to run each python file.

10. Your README file should include any specifics regarding running your code, if you made any necessary changes mention them there.

Your submission should include the following files:

1. The source code in python.
2. Your evaluation & analysis in .pdf format. Note that your analysis should include visualization plots as well as a discussion of results, as described in details in the questions above.
3. A README file containing your name, instructions to run your code and anything you would like us to know about your program (like errors, special conditions, etc).