LAHORE UNIVERSITY OF MANAGEMENT SCIENCES
Syed Babar Ali School of Science and Engineering

EE514/CS535 Machine Learning
Spring Semester 2021

Programming Assignment 4 – Logistic Regression and Naïve Bayes

Issued: Monday, 29th March, 2021

Total Marks: 100

Submission: 11:55 pm, Sunday, 11th April, 2021

Contribution to Final Percentage: 8%

Goal

The aim of this assignment is to give you an initial hands-on experience regarding real-life machine learning application. You will be using logistic regression for sentiment analysis of tweets.

Instructions

- Submit your code BOTH as notebook file (.ipynb) and python script (.py) on LMS. The name of both files should be your roll number. Failing to submit any one of them will result in the deduction of marks.
- All tasks should be implemented in the same file.
- All tasks must be implemented in different cells.
- The code MUST be implemented independently. Any plagiarism or cheating of work from others or the internet will be immediately referred to the DC.
- Late submissions: 10% penalty per day for 3 days after due date.

Importing Libraries

Import the following libraries with the given commands:

```
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import numpy as np
import math
import os
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
import re
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
```

Let the TA know if any if there is an error in running this code or if any of the libraries are not installed.

Dataset

You are provided with US Airlines Twitter Sentiment Analysis dataset which contains tweets extracted using the twitter API. There are 5856 examples in the training set, 1464 examples in the test set. The dataset contains the following columns

- **Sentiment**: The tweet can be classified as either positive negative, or neutral.
- **Text**: The text of the tweet.

Downloading the Dataset

If you are working on Google Colab, run the following command on your notebook to download the data. TO BE ADDED:

```
!gdown --id 1Pjr69ChG81L2KEQLPs-dA6eW4RUCkBsD
```

Once the download is complete, run the following command to unzip:

```
!unzip PA4_dataset.zip
```

If you are working on Jupyter, you can download the dataset locally with this link.

Part-1: Data Preprocessing (10 Marks)

Read the .csv file and perform the following on your dataset. You may find the **string** and **regex** module useful for this purpose. In the preprocessing step, you are required to remove stop words, unwanted symbols, punctuation marks, hyperlinks and usernames from the tweets and lower case them. The list of stop words is provided to you along with the dataset.

Part-2: Bag of Words (10 Marks)

In this task, you'll represent each tweet as a bag-of-words (BoW), that is, an unordered set of words with their position ignored, keeping only their frequency in the tweet.
For example, consider the below tweets:
T1 = Welcome to machine learning, Machine!
T2 = Machine learning is fun.

The bag-of-words representation (ignoring case and punctuation) for the above tweets are:

Vocabulary	Welcome	to	Machine	Learning	Is	Fun
T1	1	1	2	1	0	0
T2	0	0	1	1	1	1

Basically, it will be a sparse matrix of size **Total Tweets x Size of Vocabulary**.
Note: We only use the training set to construct the vocabulary for the BoW representation.

Part-3: Implementation of Logistic Regression From Scratch (30 Marks)

You are required to implement **One-vs-Rest Multi-class Logistic Regression Model** from scratch keeping in view all the discussions from the class lectures. In One-vs-Rest strategy, there are N total binary classifiers where N is equal to total number of classes. In this part, you will implement the following three classifiers:

- **Classifier-1**: [Negative] vs [Positive, Neutral]
- **Classifier-2**: [Neutral] vs [Negative, Positive]
- **Classifier-3**: [Positive] vs [Neutral, Negative]

The sigmoid function is used on each classifier to compute the probability of the feature vector x belonging to that class. Then the label of the classifier which returns the highest probability is assigned to x. Feel free to read this article to get in-depth insight of One-vs-Rest classification. **If Scikit-Learn is used in this part, you will NOT get any credit**. It is HIGHLY recommended that you use matrix multiplication in your implementation instead of for loops to avoid unnecessary runtime. Specifically, you'll need to implement the following:

- *Sigmoid function* using the formula:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

  If we take z as the hypothesis, $\boldsymbol{\theta}^T \boldsymbol{x}$, where $\boldsymbol{\theta}$ is the weight vector, and $\boldsymbol{x}$ is the festure vector (with bias added). Then the sigmoid function gives us the probability $\boldsymbol{x}$ belonging to that class:

$$P(y = m|\boldsymbol{x}) = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{\boldsymbol{\theta}^T \boldsymbol{x}}},$$

  You will need to use this function three times, for each binary logistic regression model.

- *Cross-entropy loss function* using the formula:

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{n} y_i \log(h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) + (1 - y_i)\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)),$$

  where $L(\boldsymbol{\theta})$ is binary cross-entropy loss.

- *Batch Gradient Descent function* which updates weights of the model using the derivative of the loss function. The gradient descent function will be called by every classifier individually. You are also required to use L_2 regularization to update weights. You will compute the derivative of the following new loss expression:

$$L_\lambda(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda \|\boldsymbol{x}\|_2^2,$$

  where $L(\boldsymbol{\theta})$ is cross-entropy loss and λ is the regularization parameter. Run your batch gradient descent function for large number of epochs, e.g. around 1000, to get good F1-scores.

- *Prediction function* that predicts the labels of test data.
- Plot graphs for different values of learning rate and regularization parameter with no of epochs on the x-axis and training loss on the y-axis. The overall loss is computed by adding the loss of all classifiers.
- *Evaluation function* that calculates classification accuracy, F1 score and confusion matrix. Pass the labels of test data in this function and report your results.

Part-4: Logistic Regression using Scikit-Learn (10 Marks)

Use Scikit-Learn's Logistic Regression implementation to train and test the dataset. The model should be similar to the one you made from scratch in part-3. Remember to implement one vs rest model with the in-built classifier in binary classification mode.You are not required to plot graphs in this part. Report the accuracy, F1 score, and confusion matrix of test data using Scikit-Learn.

Part-5: Implementation of Naïve Bayes classifier from scratch (30 marks)

Using the same bag of words that has been used earlier, implement a naïve Bayes classifier. Do not forget to apply Laplace (Add-1) smoothing as learned in class. Report the accuracy, F1 score, and confusion matrix of test data using the evaluation function you implemented earlier. You can use the evaluation function that you implemented in Part-3. **If Scikit-Learn is used in this part, you will NOT get any credit**.

For implementing naïve Bayes, it may be helpful to split the model into two parts:

- **Training the Naïve Bayes Classifier**
  We need to find the prior probabilities and likelihoods for all words and all classes in the training set. The prior probability for any class c is relatively easy to find, it is given by:
  $$P(c) = \frac{N_c}{N_{doc}},$$
  where N_c is the number of documents in class c, and N_{doc} is the total number of documents. To find likelihood $P(w_i|c)$ of any word w_i in class c, we first need to divide our data-set with respect to class, and determine the frequency of w_i in all documents of class c (with add-one Laplace smoothing applied) as shown:
  $$P(w_i|c) = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|},$$
  where vocabulary V consists of the union of all word types in all classes.
- **Testing the Naïve Bayes Classifier**
  Given a test data point, and the set of prior probabilities and likelihoods, we need to return the 'best' class c. We will create a vector 'sum' of length equal to the number of classes. For each class c, we will initially add our prior probability. Then for each word in the test data in our vocabulary, we will add the corresponding likelihood. Finally, the maximum index of our 'sum' vector will be the predicted class for the test data point.

More detail can be found here in Chapter 4 (Section 4.1, 4.2, 4.3) of Speech and Language Processing.

Part-6: Implementation of Naïve Bayes classifier using Scikit-Learn (10 marks)

Use Scikit-Learn's implementation of the naïve Bayes classifier on the bag of words. Remember to implement one vs rest model with the in-built classifier in binary classification mode. Report the accuracy, F1 score, and confusion matrix of test using the library's implementation.
Compare and analyse your results from Part-4 and Part-6, and justify your statements using concepts learned in class.