Introduction to Machine Learning
CSCE 478/878

Programming Assignment 3

Fall 2020

Naïve Bayes Classifier and Logistic Regression

Basic Info

You will work in teams of maximum two students from the previous assignment.

The programming code will be graded on **both implementation and correctness**.

This assignment **doesn't require a written report**.

Assignment Goals

This assignment is intended to build the following skills:

1. Text classicization using Naïve Bayes Classifier
2. Multi-class classification using Logistic Regression (Softmax Regression)

Assignment Instructions

Note: you are not allowed to use any Scikit-Learn library for building Naïve Bayes and Logistic Regression models. However, for text preprocessing and feature extraction you may use Scikit-Learn and NLTK libraries as specified in this document.

i. The code should be written in a Jupyter notebook. Use the following naming convention.
<lastname1>_<firstname1>_<lastname2>_<firstname2>_assignment4.ipynb
ii. The Jupyter notebook should be submitted via webhandin.

Score Distribution

Naïve Bayes
Part A (Model Code): 478 (30 pts) & 878 (35 pts)
Part B (Exploratory Data Analysis): 478 & 878 (5 pts)
Part C (Feature Extraction): 478 & 878 (15 pts)
Pert D (Model Evaluation): 478 (10 pts) & 878 (20 pts)
Extra credit (BONUS) tasks for both 478 & 878: 15 pts

Logistic Regression
Part A (Model Code): 478 (45 pts) & 878 (40 pts)
Part B (Exploratory Data Analysis): 478 & 878 (10 pts)
Pert C (Model Evaluation): 478 (15 pts) & 878 (25 pts)
Extra credit (BONUS) tasks for both 478 & 878: 20 pts

Total: 478 (130 pts) & 878 (150 pts)

Naïve Bayes Classifier

Dataset: You will use the UCI SMS Spam Collection Data Set (from the "*SMSSpamCollection.csv*" file) that contains a set of SMS labeled messages.

URL: https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

Part A: Model Code (478: 30 pts & 878: 35 pts)

Design a **Multinomial Naïve Bayes** classifier for performing **binary classification** on the SMS Spam collection dataset. Implement the following methods for the Multinomial_NB model class. The model uses one hyperparameter "alpha" which represents the Additive or Laplace smoothing parameter (0 for no smoothing).

1. Implement a **Multinomial_NB** model class. It should have the following methods. **[25 pts]**

   a)
   __init__(self, alpha=1.0)

Initialization function to instantiate the class.

b)

fit(self, X, Y)
Arguments:
  X : ndarray
    A numpy array with rows representing data samples and columns representing **numerical features**.

  Y : ndarray
    A 1D numpy array with labels corresponding to each row of the feature matrix X.

Returns:
  No return value necessary.

c)
predict(self, X)

This method performs classification on an array of test vectors X. Use the predict_log_proba() to generate **log probabilities for avoiding overflow**.

Arguments:
  X : ndarray
    A numpy array containing samples to be used for prediction. Its rows represent data samples and columns represent numerical features.

Returns:
  1D array of predictions for each row in X.
  The 1D array should be designed as a column vector.

This method returns log-probability estimates for the test matrix X.

d) **[5 pts]**
predict_log_proba(self, X)

This method returns log-probability estimates for the test matrix X.

Arguments:
  X : ndarray
    A numpy array containing samples to be used for prediction. Its rows represent data samples and columns represent numerical features.

Returns:

A numpy array that contains log-probability of the samples (unnormalized log posteriors) for each class in the model. The number rows are equal to the rows in X and number of columns are equal to the number of classes.

e) [**Extra Credit for 478 and Mandatory for 878**] [**5 pts**]

predict_proba(self, X)

This method returns probability estimates for the test matrix X.

Arguments:

X : ndarray

A numpy array containing samples to be used for prediction. Its rows represent data samples and columns represent numerical features.

Returns:

A numpy array that contains probability of the samples (unnormalized posterior) for each class in the model. The number rows are equal to the rows in X and number of columns are equal to the number of classes.

Part B: Exploratory Data Analysis (478 & 878: 5 pts)

2. Read in the "***SMSSpamCollection.csv***" as a pandas data frame.
3. Use the techniques from the first recitation to summarize each of the variables in the dataset in terms of mean, standard deviation, and quartiles. [**3 pts**]

4. Generate a bar plot to display the class distribution. You may use "seaborn"s barplot function. [**2 pts**]

Part C: Feature Extraction (478 & 878: 15 pts)

5. Normalize the "text" by performing stemming and lemmatization. You should do experimentation with both stemming and lemmatization and see whether stemming/lemmatization or a combination of both improves the accuracy of classification. Finally use the best performing normalization. For text normalization you may use the NLTK library. [**4 pts**]

6. Generate word clouds for both the spam and ham emails. You may use the NLTK library. [**2 pts**]
7. Remove the stop words from the text and convert the text content into numerical feature vectors. Note that for the multinomial Naïve Bayes classifier you need to count word occurrences as feature values. You may use Scikit-Learn's CountVectorizer object for text preprocessing and feature vectorization. [**3 pts**]
8. Create data or feature matrix X and the target vector Y. The number of columns in X is equal to the number of features. [**2 pts**]
9. Shuffle the rows of your data. You can use def = df.sample(frac=1) as an idiomatic way to shuffle the data in Pandas without losing column names. [**2 pts**]
10. Partition the data into train and test set (80%-20%). Use the "**Partition**" function from your previous assignment. [**2 pts**]

Part D: Model Evaluation (478: 60 pts & 878: 70 pts)

11. **Model selection via Hyper-parameter tuning**: Use the **kFold** function from the previous assignment to evaluate the performance of your model for the following values of the hyperparameter alpha = [0.0001, 0.001, 0.01, 0.1, 0.5, 1.0, 1.5, 2.0]. Determine the **best model** (model selection) based on the overall performance (lowest average error). For the error_function of the kFold function argument use the "F1 Score" function from previous assignment. [**5 pts**]

12. [**Extra Credit for 478 and Mandatory for 878**]: Generate the Receiver Operating Characteristic (ROC) curve and compute the area under curve (AUC) score. You may reuse the functions from previous assignment. [**10 pts**]

13. Evaluate your model on the **test data** and report the following performance measures. You may reuse the functions from your previous assignment.
    a. Precision
    b. Recall
    c. F1 score
    d. Confusion matrix

e. Accuracy

[5 pts]

14. [**Extra Credit for both 478 & 878**] Implement the **Multivariate Bernoulli Naïve Bayes** model. The hyperparameter should be the Additive or Laplace smoothing parameter alpha. Using cross-validation determine the best model. Evaluate your model on test data as specified in the previous question. **[15 pts]**

Logistic Regression: Multi-Class Classification

Dataset: You will use the Iris dataset for **multi-class classification**.

URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

Part A: Model Code (478 & 878 45 pts)

Design a **Softmax Regression** classifier for performing **multi-class classification** on the Iris dataset.

15. Implement the following function to convert the vector of class indices into a matrix containing a one-hot vector for each instance. **[5 pts]**

one_hot_labels(Y)

Arguments:

Y : ndarray

1D array containing data with "int" type that represents class indices/labels.

Returns:

Y_one_hot : ndarray

A matrix containing a one-hot vector for the Y of each instance. The number of rows is equal to the number of rows in Y. The number of columns is equal to the number of unique class indices/labels in Y (i.e., the number of classes).

16. Implement the following function that computes the softmax score or the normalized exponential of the score of a feature. [**5 pts**]

softmax(score):

Arguments:
score : ndarray
Score of a sample belonging to various classes.

Returns:
Y_proba : ndarray
Probability of a sample belonging to various classes.

17. Implement the following function to compute the cross-entropy loss. [**5 pts**]

cross_entropy_loss(Y_one_hot, Y_proba)

Arguments:
Y_one_hot : ndarray
A matrix containing a one-hot vector of class indices/labels for each instance.

Y_proba : ndarray
Probability of a sample belonging to various classes.

Returns:
cost : float

18. Implement a **Softmax_Regression** model class. It should have the following three methods. Note the that "fit" method should implement the **batch gradient descent** algorithm. Also, use 1st order derivative of the loss in the gradient descent.

[**30 pts**]

a)

fit(self, X, Y, learning_rate=0.01, epochs=1000, tol=None, regularizer=None, lambd=0.0, early_stopping=False, validation_fraction=0.1, **kwargs)

Arguments:

X : ndarray
: A numpy array with rows representing data samples and columns representing features.

Y : ndarray
: A 1D numpy array with labels corresponding to each row of the feature matrix X.

learning_rate : float
: It provides the step size for parameter update.

epochs : int
: The maximum number of passes over the training data for updating the weight vector.

tol : float or None
: The stopping criterion. If it is not None, the iterations will stop when (error > previous_error - tol). If it is None, the number of iterations will be set by the "epochs".

regularizer : string
: The string value could be one of the following: l1, l2, None.
If it's set to None, the cost function without the regularization term will be used for computing the gradient and updating the weight vector. However, if it's set to l1 or l2, the appropriate regularized cost function needs to be used for computing the gradient and updating the weight vector.

Note: you may define two helper functions for computing the regularized cost for "l1" and "l2" regularizers.

lambd : float
: It provides the regularization coefficient. It is used only when the "regularizer" is set to l1 or l2.

early_stopping : Boolean, default=False
: Whether to use early stopping to terminate training when validation score is not improving. If set to True, it will automatically set aside a fraction of training data as validation and terminate training when validation score is not improving.

validation_fraction : float, default=0.1
: The proportion of training data to set aside as validation set for early stopping. Must be between 0 and 1. Only used if early_stopping is True.

Note: the "fit" method should use a weight matrix "Theta_hat" that contains the parameters for the model (features and bias terms). The "Theta_hat" should be a matrix with dimension: *no. of features (including bias) x no. of classes*

Finally, it should update the model parameter "Theta" to be used in "predict" method as follows.

self.Theta = Theta_hat

b)

predict(self, X)

Arguments:

X : ndarray

A numpy array containing samples to be used for prediction. Its rows represent data samples and columns represent features.

Returns:

1D array of predicted class labels for each row in X.

Note: the "predict" method uses the **self.Theta** to make predictions.

c)

__init__(self)

It's a standard python initialization function so we can instantiate the class. Just "pass" this.

Part B: Exploratory Data Analysis (478 & 878: 10 pts)

19. Read the Iris data using the sklearn.datasets.load_iris method.
20. Use the techniques from the second recitation to summarize each of the variables in the dataset in terms of mean, standard deviation, and quartiles. **[3 pts]**
21. Shuffle the rows of your data. You can use def = df.sample(frac=1) as an idiomatic way to shuffle the data in Pandas without losing column names. **[2 pts]**
22. Generate pair plots using the seaborn package (see second recitation notebook). This will be used to identify and report the redundant features, if there is any. **[2 pts]**
23. Scale the features. **[1 pts]**

24. Partition the data into train and test set. Use the "**Partition**" function from your previous assignment. [**2 pts**]

Part C: Model Evaluation (478: 15 pts & 878: 25 pts)

25. **Model selection via Hyper-parameter tuning**: Use the **kFold** function from previous assignment to evaluate the performance of your model over each combination of parameters from the following sets. You can increase the range of values, if needed and also for more experimentation. [**10 pts**]
    a. lambd = [0.1, 0.01, 0.001, 0.0001]
    b. tol = [0.001, 0.0001, 0.00001, 0.000001, 0.0000001]
    c. learning_rate = [0.1, 0.01, 0.001]
    d. regularizer = [l1, l2]
    e. Store the returned dictionary for each and present it in the notebook.
    f. Determine the **best model** (model selection) based on the overall performance (lowest average error). For the error_function of the kFold function argument use accuracy.

26. Evaluate your model on the **test data** and report the accuracy and confusion matrix. [**5 pts**]

27. [**Extra Credit for 478 and Mandatory for 878**] Implement early stopping in the "fit" method of the Softmax_Regression model. You will have to use the following two parameters of the model: early_stopping and validation_fraction. Also note that when training the model using early stopping it should generate an early stopping curve. [**10 pts**]

28. [**Extra Credit for both 478 & 878**] Implement the Stochastic Gradient Descent Logistic Regression algorithm. Using cross-validation determine the best model. Evaluate your model on test data and report the accuracy and confusion matrix. [**20 pts**]