Introduction to Machine Learning

CSCE 478/878

Programming Assignment 1

Fall 2019

K Nearest Neighbors Model for Binary Classification

Basic Info

Late submission is not allowed on the 1st programming assignment.

You will work in teams of maximum two students.

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

The written report will be graded on content, conclusions, and presentation. It must be formatted according to the given template (posted on Canvas). The report will be graded as if the values obtained from the code portion were correct. The report should be short and to the point. The length should be between 2 to 4 pages of text plus any tables and graphs.

Assignment Goals and Tasks

This assignment is intended to build the following skills:

1. Implementation of the brute force K-NN algorithm for **binary classification**
2. Data pre-processing and feature selection techniques
3. Model evaluation techniques

Furthermore, the functions you develop for the data pre-processing and model evaluation will form a basis for future assignments.

In particular, you will be evaluating various K-NN models on the **white wine portion** of the Wine Quality dataset from UCI's repository. In order to carry out this evaluation, you will:

1. Implement two distance metrics (Euclidean and Manhattan)
2. Implement the brute-sforce K-NN algorithm
3. Implement a cross-validation function
4. Apply scaling to the dataset
5. Perform feature selection
6. Evaluate your model
7. Write a short report

Assignment Instructions

Note: you are not allowed to use any Scikit-Learn models or functions.

i) The code should be written in a Jupyter notebook and the report should be prepared as a PDF file.
   a. Name the notebook `<lastname1>_<firstname1>_<lastname2>_<firstname2>_assignment1.ipynb`
   b. Name the PDF `<lastname1>_<firstname1>_<lastname2>_<firstname2>_assignment1.pdf`
ii) The Jupyter notebook and the report should be submitted via webhandin. Only one submission is required for each group.
iii) Use the cover page (posted on Canvas) as the front page of your report.
iv) Download the wine quality dataset from: http://archive.ics.uci.edu/ml/datasets/Wine+Quality
You will be using the white wine dataset: "winequality-white.csv"

Score Distribution

Part A (Model Code): 478 (50 pts) & 878 (65 pts)
Part B (Data Processing): 478 (20 pts) & 878 (25 pts)
Part C (Model Evaluation): 478 (40 pts) & 878 (50 pts)
Part D (Written Report): 478 & 878 (30 pts)

Total: 478 (140 pts) & 878 (170 pts)

Part A: Model Code (478: 50 pts & 878: 65 pts)

1) Write a function to calculate and return the Euclidean distance of two vectors. **[2 pts]**
2) Write a function to calculate and return the Manhattan distance of two vectors. **[2 pts]**
3) Write a function to calculate and return the accuracy and generalization error of two vectors. **[4 pts]**
4) Write three functions to compute: precision, recall and F1 score. **[6 pts]**
5) Write a function to compute the confusion matrix of two vectors. **[4 pts]**
6) Write a function to generate the Receiver Operating Characteristic (ROC) curve. **[8 pts]**
7) Write a function to compute area under curve (AUC) for the ROC curve. **[4 pts]**
8) [**Extra Credit for 478 and Mandatory for 878**] Write a function to generate the precision-recall curve. **[10 pts]**

9) Implement a **KNN_Classifier** model class. It should have the following three methods. **[20 pts]**

a)
fit(self, X, Y, n_neighbors, weights, kwargs)

This method simply needs to store the relevant values as instance variables.

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.

n_neighbors : int
    The number of nearest neighbors.

weights : string, optional (default = ‘uniform’)

The weight function used in prediction. Possible values:

- 'uniform': uniform weights. All points in each neighborhood are weighted equally.
- 'distance': weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
  [**Extra Credit for 478 and Mandatory for 878**]

[5 pts]

kwargs :

Dictionary of arguments to be passed to the distance function (this will not be used with our simple distance functions, but is important for more complex functions). If you are not familiar, look up using the ** operator to unpack dictionaries as arguments.

Returns:

No return value necessary.

b)

predict(self, X)

This method will use the instance variables stored by the ***fit*** method.

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

c)

__init__(self)

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

Part B: Data Processing (478: 20 pts & 878: 25 pts)

10) Read in the **winequality-white.csv** file as a Pandas data frame.
11) The target will be the "quality" column which represents rating of wine and ranges from 3 to 8. You will need to convert it into a two-category variable consisting of "good" (quality > 5) & "bad" (quality <= 5). Your target vector should have 0s (representing "bad" quality wine) and 1s (representing "good" quality wine).
12) Use the techniques from the first recitation to summarize each of the variables in the dataset in terms of mean, standard deviation, and quartiles. Include this in your report. [**3 pts**]
13) 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**]
14) Generate pair plots using the seaborn package (see 2nd recitation notebook). This will be used to identify and report the redundant features, if there is any. [**2 pts**]
15) Drop the redundant features. [**1 pts**]
16) Write a function named "**partition**" to split your data into train and test set. The function should take 3 arguments: feature matrix (numpy array with rows representing data samples and columns representing features.), target vector (numpy array with labels corresponding to each row of the feature matrix), t. Here t is a real number to determine the size of partition. For example, if t is set to 0.2, then 80% of the data will be used for training and 20% for testing. This function should return two feature matrices for train and test data, and two target vectors for train and test data. [**6 pts**]

17) Naively run your **KNN_Classifier** model on the train dataset with *n_neighbors* = 5 and using Euclidean distance. [**6 pts**]

    a. Use accuracy and F1 score to compare your predictions to the expected labels.
    b. Now standardize each feature of your training set (subtract mean and divide by standard deviation). Use the mean and standard deviation values for each feature in the training set to scale the test data.
    c. Re-run the **KNN_Classifier** model on the standardized data, find the accuracy and F1 score with the expected labels.
    d. Compare the two accuracy values and the F1 scores; and decide whether you should use standardized data or unscaled data for the remainder of the assignment. This will go in the report
    e. [**Extra Credit for 478 and Mandatory for 878**] Perform a similar test for inverse distance weighting in the **KNN_Classifier** model and determine whether or not to use it. This will go in the report. [**5 pts**]

Part C: Model Evaluation (478: 40 pts & 878: 50 pts)

18) **Evaluation of an estimator performance via cross-validation**: Implement the S-fold cross-validation function. [**10 pts**]
   a. sFold(folds, data, labels, model, model_args, error_fuction)
      i. folds is an integer number of folds.
      ii. data is a numpy array with rows representing data samples and columns representing features.
      iii. labels is a numpy array with labels corresponding to each row of training_features.
      iv. model is an object with the fit and predict methods.
      v. model args is a dictionary of arguments to pass to the classification algorithm. If you are unfamiliar, look up using the ** operator to unpack dictionaries as arguments
      vi. error_function
         1. Returns error value between predicted and true labels. For example, mean squared error (mse) function could be used as error_function.
   b. How it should work:
      i. Use a helper function to calculate an s-partition of the data (i.e., partition the data into s equally sized portions)
      ii. For each partition
         1. Make a model using the model class
         2. Fit the data to all other partitions (1 – folds)
         3. Make prediction on current partition
         4. Store expected labels and predicted labels for current partition
      iii. Calculate the average error (for all partitions) using the **error_function** on stored expected and predicted labels.
   c. It should return:
      i. A Python dictionary with the following elements
         1. Expected labels
         2. Predicted labels
         3. Average error

19) Use your **sfold** function to evaluate the performance of your model over each combination of k and distance metrics from the following sets: [**5 pts**]
   a. k = [1, 5, 9, 11]
   b. distance = [Euclidean, Manhattan]

c. [**Extra Credit for 478 and Mandatory for 878**] weights = [uniform, distance] **[5 pts]**

d. Store the returned dictionary for each. We will need these for the report.

e. Determine the best model based on the overall performance (lowest average error). For the error_function of the S-fold function argument use the F1 score function.

20) Evaluate your model on the test data and report the performance measures. **[10 pts]**

a. Precision
b. Recall
c. F1 score
d. Confusion matrix
e. Accuracy & Generalization Error

21) Generate the ROC curve and determine the optimal threshold. This will go in your report. **[8 pts]**

22) Compute the AUC score. **[2 pts]**

23) [**Extra Credit for 478 and Mandatory for 878**] Generate the precision-recall curve and determine the optimal threshold. **[5 pts]**

24) Calculate and report the 95% confidence interval on the generalization error estimate. **[5 pts]**

Part D: Written Report (30 pts)

25) Write a "Data Summary" section. **[10 pts]**

a. Describe the dataset and the variables. What is the target? What are you calculating it from?
b. Include a table of each variables' descriptive statistics (mean, standard deviation, quartiles).
c. Describe whether or not you used feature scaling and why or why not.
d. Describe whether or not you dropped any feature and why or why not.

26) Write a "Methods" section. **[10 pts]**

a. Describe the runtime complexity of the **KNN_Classifier** model.
b. Explain the effects of increasing k. When is and isn't it (increasing k) effective?
c. **878 students:** Describe whether or not you used inverse distance weighting in the features and why.

27) Write a "Results" section.

[**10 pts**]

a. Describe the performance of the model with respect to the different levels of k and the different distance metrics. Include a table of performances, bolding the best.
b. Characterize the overall performance of your model.
c. Discuss on which quality values your model performed well, and on which it performed poorly. Include a table of average error (e.g., F1 score) to support your claims.
d. Give any final conclusions.