CSc 740 Spring 2022
Assignment 1
Due Date: 11:30 pm, March 8, 2022

Read and Follow Assignment Instructions Carefully

1. This is an individual assignment. All work submitted must be your own.

2. First read the entire assignment description to get the big picture; create your own notes about the control flow, expected functionality of the various methods and why you are being asked to implement specific items. To repeat: **First, read the entire assignment completely and think about how the different parts are connected.**

3. You are **Not allowed** to use the Scikit-Learn library to **implement any of the functionality outlined in Section A**. Otherwise Scikit-Learn library is permitted.

4. Deliverables: the code and answers should be written in a Jupyter notebook named `<lastname>_<firstname>_assignment1.ipynb. The notebook should also include short write-ups using markdown (2-5 sentences) summarizing results

5. Make sure you copy each question with the question number as a Markdown Cell in your Jupyter notebook and have the code response right below it. Points will be deducted if it is difficult to locate the question and response.

6. Make sure you comment your code. Points will be deducted if code logic is not apparent.

7. The written sections will be graded on correctness and preciseness while code will be graded on structure, implementation and correctness.

About the assignment: This assignment is intended to build the following skills:

1. Implementation of the brute force K-NN algorithm for **binary classification**
2. Data Processing, Feature Selection, and Initial Estimation
3. Model evaluation techniques and Results Summary

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-force 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

Part A: Model Code (60 pts)

1. Write a function to calculate and return the Minkowski distance with optional argument p defaulting to 'p=2' (Euclidean) of two vectors where a vector represents a data point.**[6 pts]**
2. Write a function to calculate and return the accuracy of two vectors. **[4 pts]**
3. Write three functions to compute: precision, recall and F1 score. **[6 pts]**
4. Write a function to compute the confusion matrix of two vectors. **[4 pts]**
5. Write a function to generate the Receiver Operating Characteristic (ROC) curve. **[5 pts]**
6. Write a function to compute area under curve (AUC) for the ROC curve. **[5 pts]**
7. Write a function to generate the precision-recall curve. **[5 pts]**
8. Implement a **KNN_Classifier** model class. It should have the following three methods. **[20 pts]**

a) __init__(self,) It's a standard python initialization function so we can instantiate the class. Just "pass" this. **[5 pts]**

Arguments:
n_neighbors : int, optional (default = 5) 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

p: int, optional (default = 2) Minkowski distance.
Returns: No return value necessary.

b) fit(self, X, Y) This method simply needs to store the relevant values as instance variables. **[5 pts]**

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.
Returns: No return value necessary.

c) predict(self, X,threshold=.5) This method will use the instance variables stored by the ***fit*** method. **[2 pts]** **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 class labels for each row in X. The 1D array should be designed as a column vector.

d) predict_proba(self, X) Same as c) but for probabilities **[3 pts]** **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 prediction probabilities for positive class for each row in X. The 1D array should be designed as a column vector.

e) get_params(self) Get parameters for this estimator. **[3 pts]**

Arguments: N/A

Returns: *dict* Model parameter names mapped to their values.

f) set_params(self, **params) **[2 pts]** **Arguments:**

***params : dict* A dictionary with the model parameter names to change mapped to their values

Returns: No return value necessary.

9. Write a function named "**partition**" to split your data into training and test sets. The function should take 4 arguments: [**5 pts**]

- 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 where 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.
- shuffle (default=True) where shuffle is a boolean whether to shuffle the data prior to partitioning. You will be required to use "shuffle=True" for this assignment
- This function should return two feature matrices for training and test data, and two target vectors for training and test data (4 tuple).

Part B: Data Processing, Feature Selection, and Initial Estimation (40 pts)

10. Read in the **winequality-white.csv** file as a Pandas data frame.
11. The target will be the "quality" column which represents the 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). [**2 pts**]
12. Provide a table with univariate statistics of your data (mean, standard deviation, and quartiles, min, max, missing count, number of unique values). [**4 pts**]
13. Generate pair plots using the seaborn package to help identify redundant features. For any redundant features(?), report, drop, and explain your logic (w/ markdown). [**4 pts**]
14. Use your "**partition**" function to split the data into 80% train and 20% test. [**5 pts**]
15. Naively run your **KNN_Classifier** model on the training dataset with *n_neighbors* = 5 and using Euclidean distance. [**15 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) and apply trained standardization to the test set. Use the mean and standard deviation values for each feature in the training set to scale the test data (you can use sklearn.preprocessing.StandardScaler)

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.

e. Perform a similar test for inverse distance weighting in the **KNN_Classifier** model and determine whether or not to use it. [**5 pts**]

16. Repeat #15 a-d, but using a logistic regression with 'elasticnet' or 'l2' penalty (feel free to use sklearn.linear_model.LogisticRegression) [**10 pts**]

Part C: Model Evaluation and Results Summary (100 pts)

17) **Evaluation of an estimator performance via cross-validation**: Implement the S-fold cross validation function. [**15 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 :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). You may use sklearn.model_selection.KFold if you wish and assume data is already shuffled.
ii. For each partition
    a. Make a model using the model class
    b. Fit the data to all other partitions (1 – folds)
    c. Make prediction on current partition
    d. 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 a Python tuple with the following

i. Expected labels
ii. Predicted labels
iii. Average error

18) **Only using the training portion of your data**, use your **sfold** function to evaluate the performance of your model over each combination of k and distance metrics from the following sets: [**10 pts**]

i. k=[1,5,9,11]
   b. distance = [Euclidean, Manhattan]
ii. weights = [uniform, distance]
iii. From the returned tuple store as a row in a pandas DataFrame with headers: Experiment name, k, distance, weights, Average F1
iv. Determine the best model based on the overall performance. For the error_function of the S-fold function argument use the F1 score function.

19) Repeat #18 for at least 3 experiments for the regularized logistic regression from #16 and discuss why you optimized over you selected hyper-parameters [**10 pts**]

20) Based on the results above, use the full training portion (80%), to re-estimate your best model. Discuss your model choice. [**5 pts**]

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

i. Precision
ii. Recall
iii. F1 score
iv. Confusion matrix
v. Accuracy & Generalization Error

22) Generate the ROC curve and determine the optimal threshold that maximizes the F1 score. [**10 pts**]

23) Compute the AUC score. [**5 pts**]

24) Generate the precision-recall curve and determine the optimal threshold. **[5 pts]**

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

26) Write a "Summary and Methods" section. [**10 pts**] No more than 2-5 sentences for each question below

i. Provide a summary of the project and what you completed in the assignment.

ii. Describe the dataset and features. What is the target? What are you calculating it from?

iii. Describe the differences in *fit* and *predict* between the regularized logistic regression vs **KNN_Classifier.** In particular, discuss training time vs prediction time for large data. Also discuss the hyperparameters of each and why they are used.

27) Write a "Results" section. [**15 pts**] No more than 2-5 sentences for each question below

a) Describe the performance of the KNN 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 which quality values led to good performance of your model and those that resulted in poor performance. Include a table of average error (e.g., F1 score) to support your claims.

d) Give any final conclusions.