# Csc 59866 Senior Design I Assignment 1

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

**Read and Follow Assignment Instructions Carefully**

1. This is team-based grade assignment, but All members of the team need to submit the work according to your own understanding, codes must be your own.

2. First read the entire assignment description to get the big picture; make notes down the control flow, expected functionality of the various methods and why you are being asked to implement specific items.

3. You are Not **allowed** to use any Scikit-Learn library for your work, but you are allowed to use it to test your result

**4.** Deliverables:

**a.** The code and answers should be written in a Jupyter notebook named `<lastname>_<firstname>_assignment1.ipynb.

**b.** You should have a summary file written in a pdf file named `<lastname>_<firstname>_assignment1.pdf` which documents your result and analysis.

**c.** submit your hw to github, then put the hw link to Blackboard, make sure I could find all of your hw1 by your link.

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

## Part A: Model Code

1. Write a function to calculate and return the Euclidean distance of two vectors.
2. Write a function to calculate and return the Manhattan distance of two vectors
3. Write a function to calculate and return the accuracy and generalization error of two vectors.
4. Write three functions to compute: precision, recall and F1 score.
5. Write a function to compute the confusion matrix of two vectors.
6. Write a function to generate the Receiver Operating Characteristic (ROC) curve.
7. Write a function to compute area under curve (AUC) for the ROC curve.
8. Write a function to generate the precision-recall curve.
9. Implement a **KNN_Classifier** model class. It should have the following three methods.

**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.

*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.

## Part B: Data Processing

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.
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.
14. Generate pair plots using the seaborn package. This will be used to identify and report the redundant features, if there is any.
15. Drop the redundant features.

16. Write a function named "**partition**" to split your data into training 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 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.
- This function should return two feature matrices for training and test data, and two target vectors for training and test data.

17. Naively run your **KNN_Classifier** model on the training dataset with *n_neighbors* = 5 and using Euclidean distance.

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 described in the report
e. Perform a similar test for inverse distance weighting in the **KNN_Classifier** model and determine whether or not to use it.

## Part C: Model Evaluation

18. evaluate the performance of your model over each combination of k and distance metrics from the following sets:

i. k=[1,5,9,11]
distance = [Euclidean, Manhattan]
weights = [uniform, distance]

19 Evaluate your model based on your own precision… functions on the test data and report the performance measures.

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