Hacettepe University

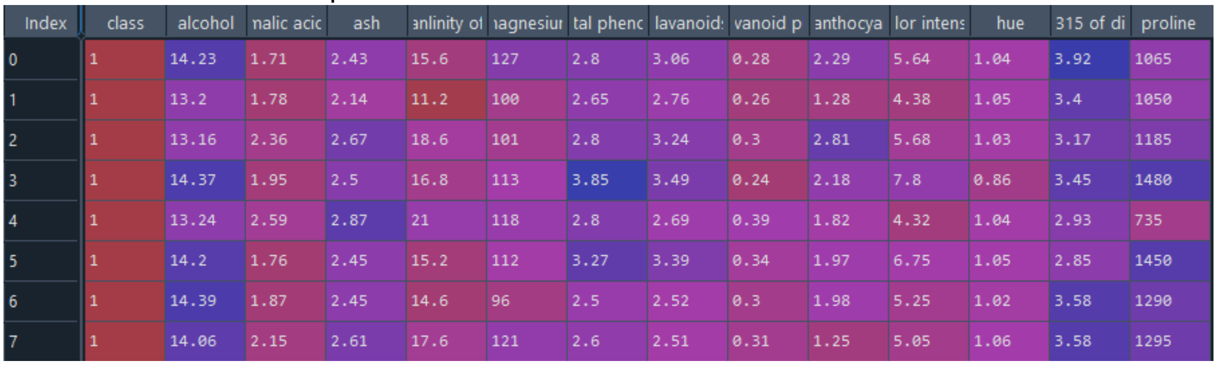
Department of Electrical and Electronics Engineering

ELE 489: Fundamentals of Machine Learning

Name: Zülal Kübra Nur Yüksel
Student ID: 2200357083
Date: 03.04.2025

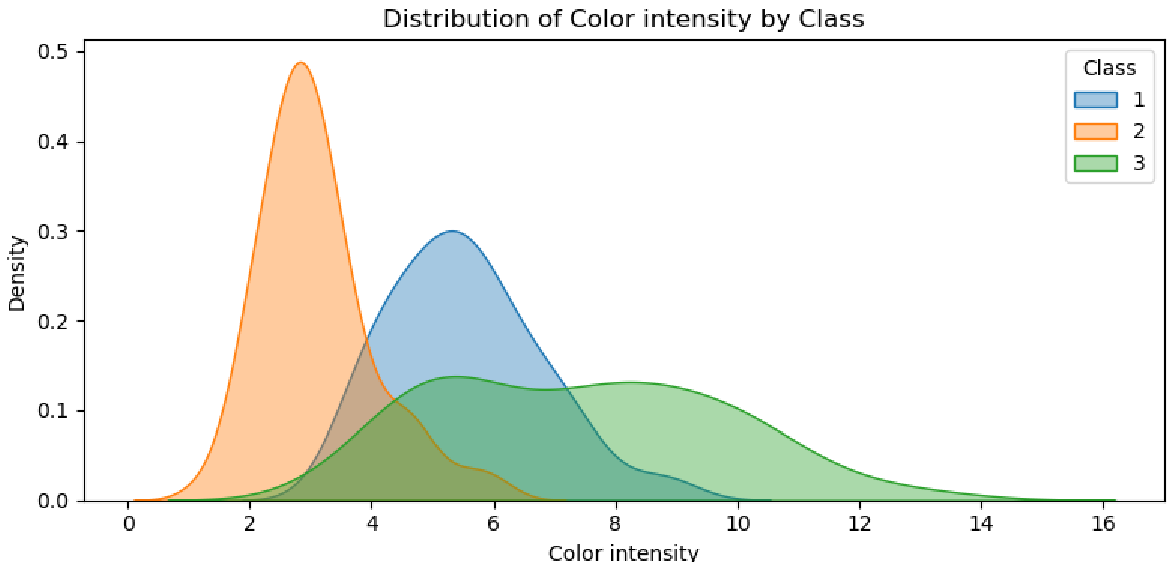
HW-1(https://github.com/zulalyuksel/ELE489_HW1)

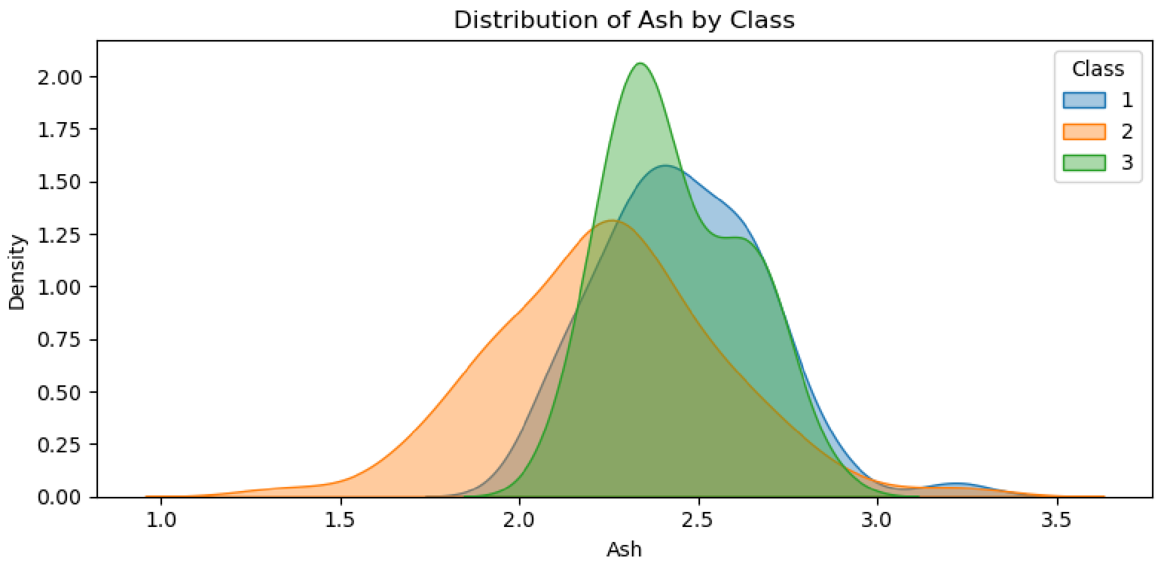
1) Download the dataset from the link. Load the dataset, and visualize some of the features of your choice. See if the features overlap from different classes.

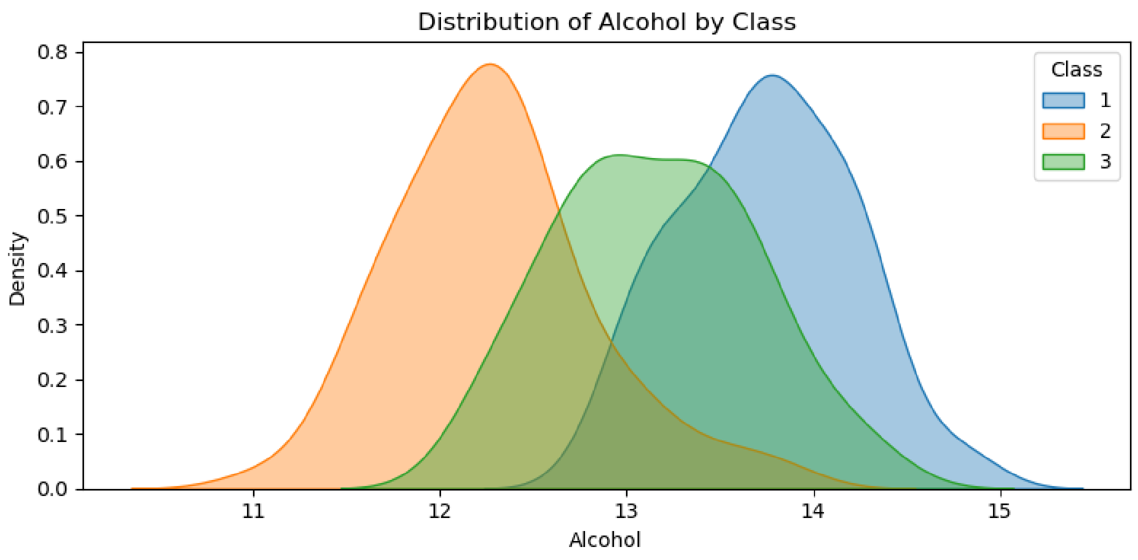


Index	class	alcohol	nalic acic	ash	anlinity ol	nagnesiur	tal phenc	lavanoid:	vanoid p	anthocya	lor intens	hue	315 of di	proline
0	1	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050
2	1	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86	3.45	1480
4	1	13.24	2.59	2.87	21	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735
5	1	14.2	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
6	1	14.39	1.87	2.45	14.6	96	2.5	2.52	0.3	1.98	5.25	1.02	3.58	1290
7	1	14.06	2.15	2.61	17.6	121	2.6	2.51	0.31	1.25	5.05	1.06	3.58	1295

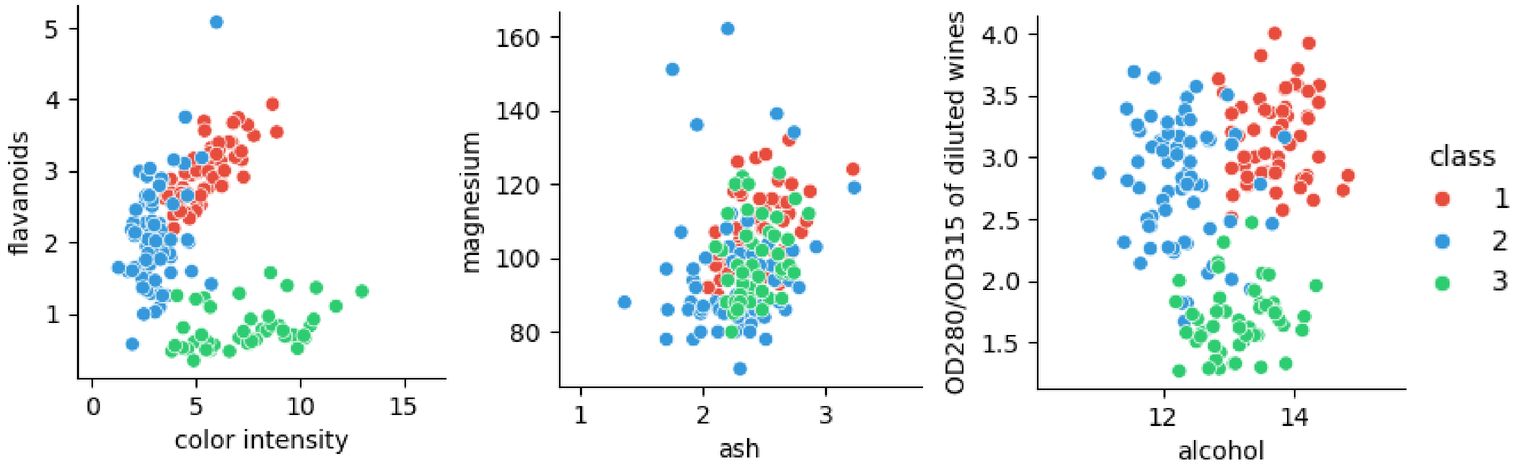
Here's the first 8 instances of 13 numerical features of the loaded dataset, which actually consists of 178 instances with 13 numerical features.







Above are the visualizations of the distribution of different features by classes. It is clear that one feature is not enough for the separation of classes. The three classes are overlapping in each case. In addition to individual feature distributions, I also plotted feature pairs to investigate if combinations of features provide better class separation.



We can observe different graphs with and without overlapping features from different classes. For instance, the second graph has all overlapped features whereas the third one is more scattered.

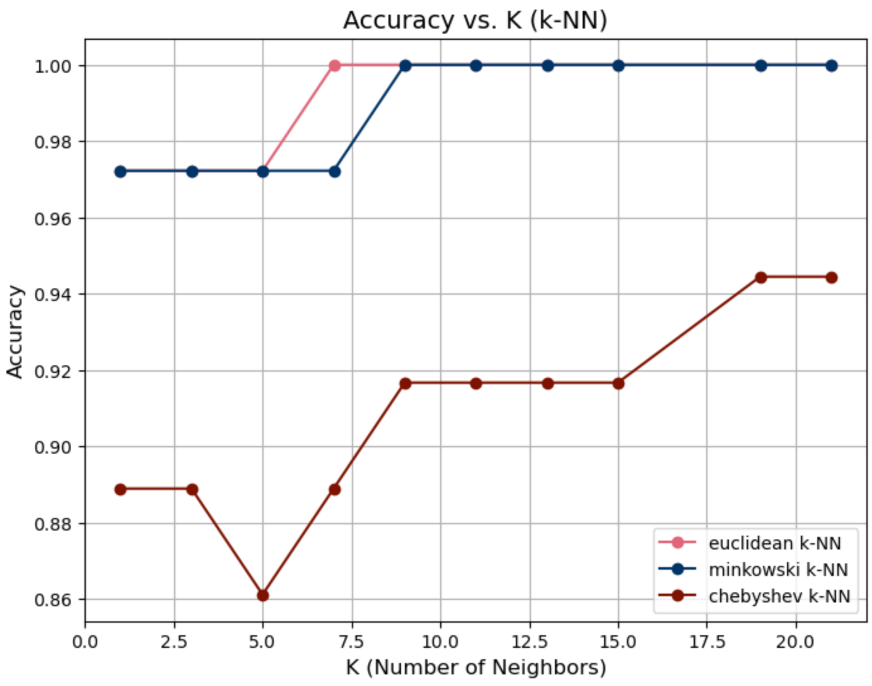
2) Perform any necessary preprocessing (e.g., normalization, handling missing values). Split the data into training (80%) and testing (20%) sets using train_test_split() from sklearn.model_selection.

The dataset doesn't have any missing value, as all the columns are filled. Since the "class" column is already in numeric form, there's no need to do any extra encoding or transformation. Also, we don’t need to worry about outliers right now because there aren’t any extreme values that could seriously affect the model. Based on this, the main preprocessing step needed is normalization to make sure all the features are on the same scale, which is important for distance-based models like KNN.

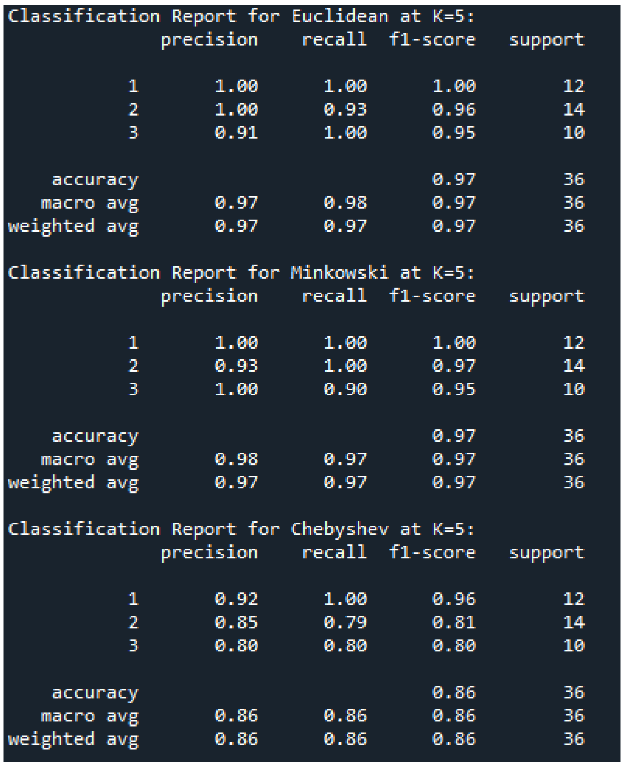
```
#spliting the data into training (80%) and testing (20%) sets and normalization
theFeatures = theData.iloc[:,1:14].values
X_1 = pd.DataFrame(data = theFeatures, index = range(178), columns = [features])
theClass = theData.iloc[:,0:1].values
Y_1 = pd.DataFrame(data = theClass, index = range(178), columns = ['class'])

x_train, x_test,y_train,y_test = train_test_split(X_1,Y_1,test_size=0.8,
random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(x_train)
X_test = sc.transform(x_test)
```

3) Implement the k-NN algorithm from scratch (without using sklearn.neighbors.KNeighborsClassifier). Allow for different values of K (e.g., 1, 3, 5, 7, 9). Compare at least two distance metrics (e.g., Euclidean, Manhattan). Calculate classification accuracy for each value of K. Plot the accuracy vs. K, and analyze the impact of K on model performance. Provide a confusion matrix and classification report.

I started the implementation by calculating the distances first. I used three different distance metrics. The first one was **Euclidean distance** which is computed by finding the square root of the sum of the squared differences between corresponding features of the test and training points. The second one is **Minkowski distance** calculated by taking the absolute difference between the test and training points for each feature, raising these differences to the power of p, summing them up, and then taking the p-th root of the total. I chose p=4 in my code. As the third one I employed **Chebyshev distance** which is calculated by taking the absolute differences between the test and training points for each feature and then selecting the maximum of these absolute differences. Later, I defined a function named NN, which identifies the K nearest neighbors by sorting the computed distances and selecting the indices of the smallest K values. After identifying the nearest neighbors, the **voting()** function performs majority voting: it counts how many times each label appears in the K nearest neighbors and returns the most common label as the predicted class for the test point. Finally, the overall KNN function applies this process iteratively for each test point.



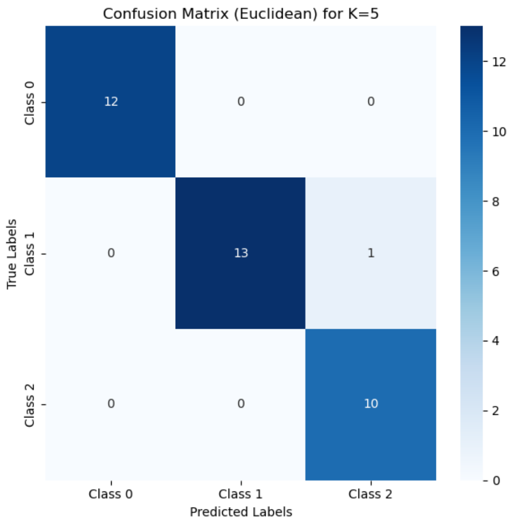
After adding the related codes for accuracy calculations, I obtained the accuracy vs. K graph. As can be seen from the graph on the side, both Euclidean and Minkowski had the best accuracies of 1 when K is greater than 9. On the contrary, the accuracy of the algorithm with Chebyshev metrics could not reach 1, the best it could get was 0.94. Unfortunately, the accuracies were not to my expectations. They were supposed to be small with smaller K values, better at medium values and again get worse with increasing K. That is because, with small values , the model becomes sensitive to individual data points, and with very large values of K, the model becomes too simple, and it may start underfitting. Thus, medium level K values should be the best ones.

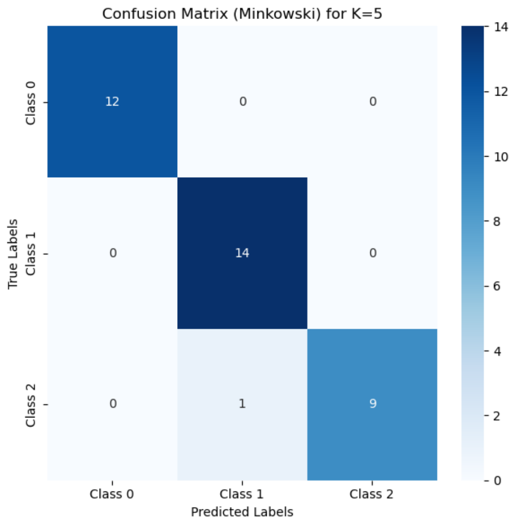


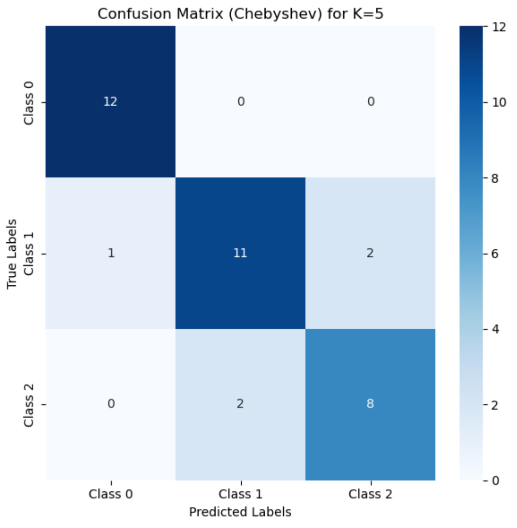
```
Classification Report for Euclidean at K=5:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      0.93      0.96        14
           3       0.91      1.00      0.95        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36

Classification Report for Minkowski at K=5:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       0.93      1.00      0.97        14
           3       1.00      0.90      0.95        10

    accuracy                           0.97        36
   macro avg       0.98      0.97      0.97        36
weighted avg       0.97      0.97      0.97        36

Classification Report for Chebyshev at K=5:
              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       0.85      0.79      0.81        14
           3       0.80      0.80      0.80        10

    accuracy                           0.86        36
   macro avg       0.86      0.86      0.86        36
weighted avg       0.86      0.86      0.86        36
```

Here's the classification report for all three metrices when K=5.
 Euclidean and Minkowski distances perform similarly, with high accuracy and good classification for all classes. Chebyshev distance has a lower accuracy, especially for class 2 and class 3, indicating it might not be the best choice for this dataset.

Finally, the confusion matrix when K=5 for Euclidean, Minkowski and Chebyshev distances is above. From the accuracy vs. K graph, we know that Euclidean and Minkowski performed the best at K=5. We can further observe the deviations in classes from the confusion matrices. It can be seen that Euclidean distance performs best with the least misclassification. Minkowski distance performs similarly but with a slight increase in misclassifications, especially for class 2. Chebyshev distance causes more misclassifications, particularly for class 2. So, Euclidean distance can be the best choice for this dataset in terms of accuracy, while the Chebyshev distance might need further tuning or a different approach to improve its performance.


Confusion Matrix (Euclidean) for K=5
True Labels
Class 0
Class 1
Class 2
Predicted Labels


Confusion Matrix (Minkowski) for K=5
True Labels
Class 0
Class 1
Class 2
Predicted Labels


Confusion Matrix (Chebyshev) for K=5
True Labels
Class 0
Class 1
Class 2
Predicted Labels