TW5: Naive Bayesian

- Full names of your team members who work on the assignment.
    - Xiaomei Xie
    - Lili Hao
- URL links to the notebook of each student on GitHub repo.
    - https://github.com/xiaomeiX/NaiveBayes_TW5
    - https://github.com/lhaoSeattleu/TW5-NB
- A summary of what you learned from the teamwork assignment.

Learning objectives:

- Be able to understand Bayesian classifier.
- Be able to develop a ML framework and test it.
- Be able to evaluate classification models.
- Be able to understand parameters of classification models.

Part 1: Construct a ML framework and develop classification models.

- Dataset: ***iris.csv*** is stored in a folder
- Your modeling analysis should be done on two different datasets:
    - The original dataset
    - Normalized data using min-max normalization.
- Apply Naive Bayes classifiers
- Apply KNN classifiers
- A framework of the k-cross validation (k = 10)
- Display confusion matrix (a matrix with numbers)
- Print a summary of the performance metrics.
- Plot ROC curves

Submission(s)

Each student should make individual submissions.

- **Part 2**:
    - Submit a summary of your learning to Canvas. Your document should include:
        - Full names of your team members who work on the assignment.
        - URL links to the notebook of each student on GitHub repo.
        - A summary of what you learned from the teamwork assignment.

Your summary should include the comparisons of the two models and the model performance based on parameters (e.g., k value in k-NN classifier).

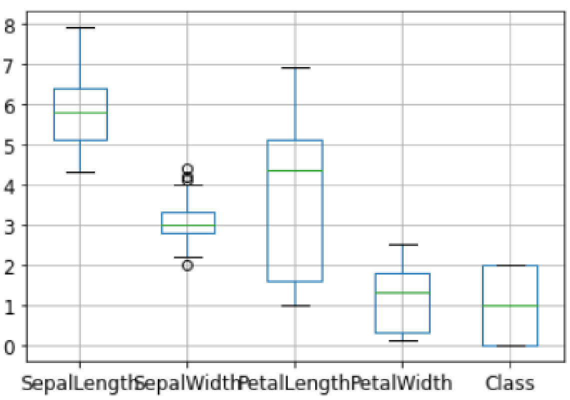
Iris Dataset:

This dataset includes 4 columns as floats, 1 column as an int and 1 column as a string:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   SepalLength  150 non-null    float64
 1   SepalWidth   150 non-null    float64
 2   PetalLength  150 non-null    float64
 3   PetalWidth   150 non-null    float64
 4   Name         150 non-null    object
 5   Class        150 non-null    int64
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

The dataset has no missing data, and a few outliers.

```
SepalLength    0
SepalWidth     0
PetalLength    0
PetalWidth     0
Name           0
Class          0
dtype: int64
```



The numeric columns range from 0 to 8.

	SepalLength	SepalWidth	PetalLength	PetalWidth	Class
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667	1.000000
std	0.828066	0.433594	1.764420	0.763161	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name	Class
0	5.1	3.5	1.4	0.2	Iris-setosa	0
1	4.9	3.0	1.4	0.2	Iris-setosa	0
2	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5.0	3.6	1.4	0.2	Iris-setosa	0

Naive Bayes classifiers

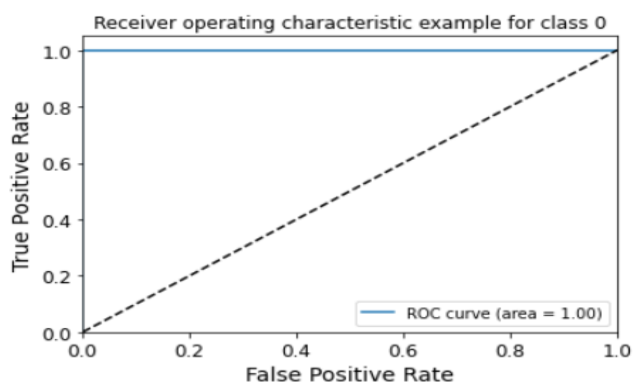
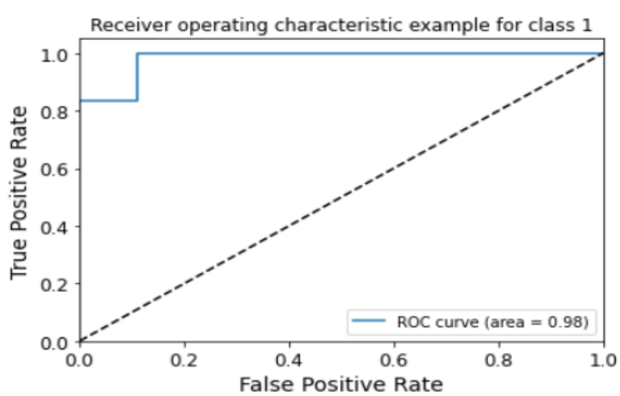
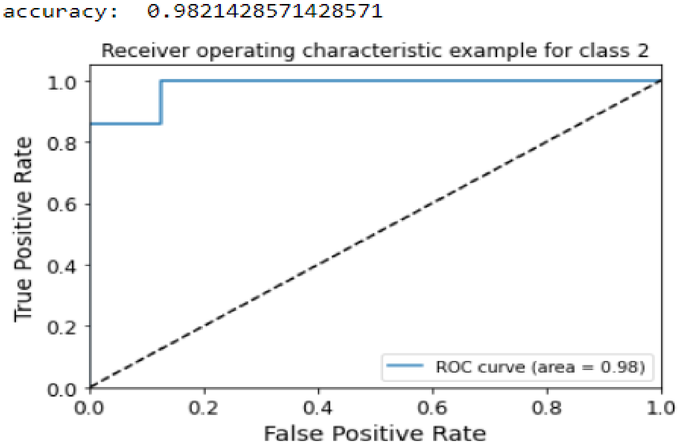
KFold(n_splits=10): we first applied Naïve Bayes classifiers. We used 10 splits for K fold cross validation: 6 of them have 100% accuracy, 1 of them has 86.7% accuracy, and the remaining 3 of them are all above 90% accuracy.

k	Confusion matrix		K	Confusion matrix	
1	[[6 0 0] [0 6 0] [0 0 3]]	precision recall f1-score support Class 0 1.00 1.00 1.00 6 Class 1 1.00 1.00 1.00 6 Class 2 1.00 1.00 1.00 3 accuracy 1.00 15 macro avg 1.00 1.00 1.00 15 weighted avg 1.00 1.00 1.00 15 precision (weighted): 1.0 recall avg (weighted): 1.0 accuracy: 1.0	6	[[6 0 0] [0 4 2] [0 0 3]]	precision recall f1-score support Class 0 1.00 1.00 1.00 6 Class 1 1.00 0.67 0.80 6 Class 2 0.60 1.00 0.75 3 accuracy 0.87 15 macro avg 0.87 0.89 0.85 15 weighted avg 0.92 0.87 0.87 15 precision (weighted): 0.92 recall avg (weighted): 0.8666666666666667 accuracy: 0.8666666666666667
2	[[4 0 0] [0 3 0] [0 0 8]]	precision recall f1-score support Class 0 1.00 1.00 1.00 4 Class 1 1.00 1.00 1.00 3 Class 2 1.00 1.00 1.00 8 accuracy 1.00 15 macro avg 1.00 1.00 1.00 15 weighted avg 1.00 1.00 1.00 15 precision (weighted): 1.0 recall avg (weighted): 1.0 accuracy: 1.0	7	[[5 0 0] [0 4 0] [0 1 5]]	precision recall f1-score support Class 0 1.00 1.00 1.00 5 Class 1 0.80 1.00 0.89 4 Class 2 1.00 0.83 0.91 6 accuracy 0.93 15 macro avg 0.93 0.94 0.93 15 weighted avg 0.95 0.93 0.93 15 precision (weighted): 0.9466666666666667 recall avg (weighted): 0.9333333333333333 accuracy: 0.9333333333333333
3	[[9 0 0] [0 4 0] [0 0 2]]	precision recall f1-score support Class 0 1.00 1.00 1.00 9 Class 1 1.00 1.00 1.00 4 Class 2 1.00 1.00 1.00 2 accuracy 1.00 15 macro avg 1.00 1.00 1.00 15 weighted avg 1.00 1.00 1.00 15 precision (weighted): 1.0 recall avg (weighted): 1.0 accuracy: 1.0	8	[[3 0 0] [0 6 0] [0 0 6]]	precision recall f1-score support Class 0 1.00 1.00 1.00 3 Class 1 1.00 1.00 1.00 6 Class 2 1.00 1.00 1.00 6 accuracy 1.00 15 macro avg 1.00 1.00 1.00 15 weighted avg 1.00 1.00 1.00 15 precision (weighted): 1.0 recall avg (weighted): 1.0 accuracy: 1.0
4	[[4 0 0] [0 6 0] [0 1 4]]	precision recall f1-score support Class 0 1.00 1.00 1.00 4 Class 1 0.86 1.00 0.92 6 Class 2 1.00 0.80 0.89 5 accuracy 0.93 15 macro avg 0.95 0.93 0.94 15 weighted avg 0.94 0.93 0.93 15 precision (weighted): 0.9428571428571428 recall avg (weighted): 0.9333333333333333 accuracy: 0.9333333333333333	9	[[5 0 0] [0 5 0] [0 0 5]]	precision recall f1-score support Class 0 1.00 1.00 1.00 5 Class 1 1.00 1.00 1.00 5 Class 2 1.00 1.00 1.00 5 accuracy 1.00 15 macro avg 1.00 1.00 1.00 15 weighted avg 1.00 1.00 1.00 15 precision (weighted): 1.0 recall avg (weighted): 1.0 accuracy: 1.0
5	[[6 0 0] [0 4 0] [0 0 5]]	precision recall f1-score support Class 0 1.00 1.00 1.00 6 Class 1 1.00 1.00 1.00 4 Class 2 1.00 1.00 1.00 5 accuracy 1.00 15 macro avg 1.00 1.00 1.00 15 weighted avg 1.00 1.00 1.00 15 precision (weighted): 1.0 recall avg (weighted): 1.0 accuracy: 1.0	10	[[2 0 0] [0 5 1] [0 1 6]]	precision recall f1-score support Class 0 1.00 1.00 1.00 2 Class 1 0.83 0.83 0.83 6 Class 2 0.86 0.86 0.86 7 accuracy 0.87 15 macro avg 0.90 0.90 0.90 15 weighted avg 0.87 0.87 0.87 15 precision (weighted): 0.8666666666666667 recall avg (weighted): 0.8666666666666667 accuracy: 0.8666666666666667

ROC Curve:

We use binarize three class labels for the Roc Curve. The receiver operating characteristic curve shows the performance of a classification model at all thresholds using True Positive Rate and False Positive rate. We have the accuracy for each class as:

- Class_0: 1.0
- Class_1: 0.98148
- Class_1: 0.98214

	Roc Curve
Class_0	accuracy: 1.0 
Class_1	accuracy: 0.9814814814814814 
Class_2	accuracy: 0.9821428571428571 

KNN Classifier

We tried different values for n_neighbors, such as 1, 5, 15, 20. When n_neighbors = 5, the model has the best training and testing accuracy.

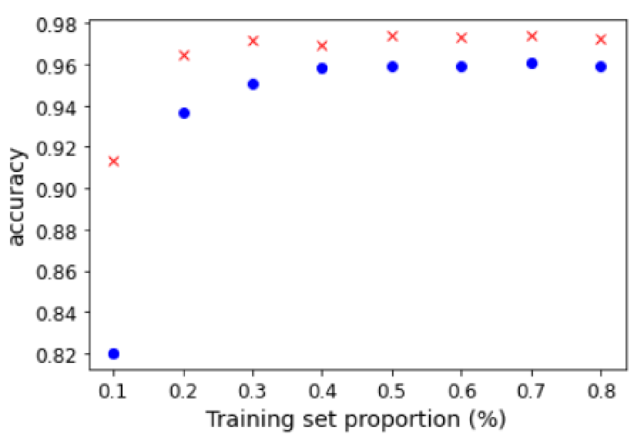
```
k=1
training score:  1.0
testing score:  0.9736842105263158
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        13
     class_1       1.00      0.94      0.97        16
     class_2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

```
k=5
training score:  0.9732142857142857
testing score:  0.9736842105263158
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        13
     class_1       1.00      0.94      0.97        16
     class_2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

```
k=15
training score:  0.9642857142857143
testing score:  0.9736842105263158
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        13
     class_1       1.00      0.94      0.97        16
     class_2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

```
k=20
training score:  0.9642857142857143
testing score:  0.9736842105263158
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        13
     class_1       1.00      0.94      0.97        16
     class_2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

We also tried different training and testing data splits, and found the best performance is between 70% and 80%. So, we chose default 75% as the split percentage.



Comparison between Naïve Bayes and KNN classifiers

Below is the training and testing accuracy comparison between Naïve Bays and KNN classifier based on both original data (training testing data split at 75%) and normalized data (MinMaxScaler, training testing data split at 75%):

- The KNN classifier has better training accuracy than Naïve Bayes classifier. However, the testing result is worse. In contrast, Naïve Bayes classifiers has better accuracy in testing data.
- For Naïve Bayes classifier, the training and testing data accuracy stays same between original data and normalized data. However, for the KNN classifier the normalized data has better performance on the training dataset.
- The Naïve Bayes classifier has overall better performance even on original data. This is because Naïve Bayes algorithm is a classification technique based on Bayes' Theorem. It has the assumption that each predictor is independent, and the calculation is based only on the occurrence of a particular feature in a class.
- KNN classifier has performance improvement on normalized data. This is because KNN finds the distances between data points and selects (K) amount of the closest points. Therefore, KNN's performance requires scaled and centered data to improve the performance.

Accuracy	**Naive Bayes classifier**	**KNN (neighbor = 5) classifier**
Original Data Accuracy	`Train: 0.9378787878787878` `Test: 1.0`	`Train: 0.9545454545454547` `Test: 0.9736842105263158`
Normalized Data Accuracy	`Train: 0.9378787878787878` `Test: 1.0`	`Train: 0.9643939393939395` `Test: 0.9736842105263158`