Pattern Recognition HW1

我選用 Iris、Wheat-Seeds Dataset、Breast Cancer Wisconsin (Diagnostic) Dataset、Pima Indians Diabetes Database 四種 Dataset，前兩者為三個 class，後兩者皆為兩個 class。

每個 Dataset 分別用 Bayesian classifier、Naïve-Bayes classifier、linear classifier 進行分類。Linear classifier 採用 MSE 為 loss, 以 Gradient Descent 做 optimization(Batch version)。

1. Experiments and Analysis

I. Iris Dataset

- Datasize：150
- Features： sepal length、sepal width、petal length、petal width
- Class: Iris Setosa、Iris Versicolour、Iris Virginica
- Split Dataset to **Training Set：Testing Set = 8 : 2**

Training Set 上錯誤率：

Bayesian classifier: 2.5%

Naïve-Bayes classifier: 3%

Testing Set 上錯誤率：

Bayesian classifier: 3.34%

Naïve-Bayes classifier: 4%

3-fold-Cross-Validation 上平均錯誤率(On Training Set)：

Bayesian classifier: 2.5%

Naïve-Bayes classifier: 4.17%

分析：

Training Set、Testing Set 上 兩者準確率都還不錯，Bayesian classifier 錯誤率都略低於 Naïve-Bayes classifier，推測 Iris 的 features 是具有些微相關性的，也可能因僅有 4 個 features 所以 Naïve-Bayes classifier 的優點並沒有展現出來，因此沒有比 Bayesian classifier 有更好的 performance

II. Wheat-Seeds Dataset

- Datasize：199
- Features：Area、Perimeter、Compactness、Kernel.Length、Kernel.Width、Asymmetry.Coeff、Kernel.Groove (7 features)
- Class: 分類種子種類 Type(1、2、3)
- Split Dataset to **Training Set：Testing Set = 8 : 2**

Training Set 上錯誤率：

Bayesian classifier: 3.78%

Naïve-Bayes classifier: 7.55%

Testing Set 上錯誤率：

Bayesian classifier: 6.5%

Naïve-Bayes classifier: 8.9%

3-fold-Cross-Validation 上平均錯誤率(On Training Set)：

Bayesian classifier: 5.1%

Naïve-Bayes classifier: 8.81%

分析：

大部分情況 Bayesian classifier 仍比 Naïve-Bayes classifier 準確率更高一些，features 間應仍有些許關聯，Training Set、Testing Set 上的錯誤率平均來說並沒有差太多。

III. Breast Cancer Wisconsin (Diagnostic) Dataset

- Datasize：569
- Features：30 features
- Class: 是否有乳癌
- Split Dataset to **Training Set：Testing Set = 8 : 2**

[Note]Linear Classifier 參數設定-> learning rate 0.000001 (測試後設定 2 倍就會 Diverge)；Iteration: 100000, Target label 設定為[-1, 1]

Training Set 上錯誤率：

Bayesian classifier: 2%

Naïve-Bayes classifier: 5.5%

Linear classifier: 29.36%

Testing Set 上錯誤率：

Bayesian classifier: 5.3%

Naïve-Bayes classifier: 6.2%

Linear classifier: 31%

3-fold-Cross-Validation 上平均錯誤率(On Training Set)：

Bayesian classifier: 3.84%

Naïve-Bayes classifier: 6.6%

Linear classifier: 25.82%

分析：

Linear classifier 在此 Dataset 錯誤率相當大，但 Bayesian、Naïve-Bayes 表現還不錯，因此 Dataset 應是 linearly non-separable 且 Linear classifier 的 Bias 太大，且受 learning rate、iteration 次數影響準確率。

Confusion Matrix (On Training Set)

Bayesian classifier

	Predict Negative↓	Predict Positive↓
Actual Negative→	167	8
Actual Positive →	5	275

Naïve-Bayes classifier:

	Predict Negative↓	Predict Positive↓
Actual Negative→	156	19
Actual Positive →	7	273

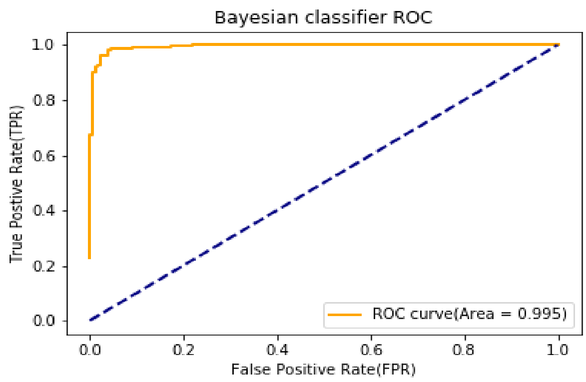
linear classifier:

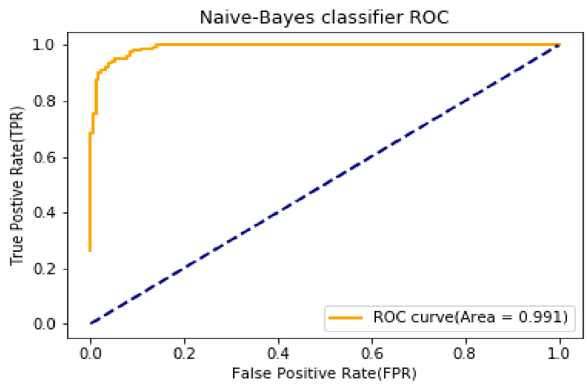
	Predict Negative↓	Predict Positive↓
Actual Negative→	100	65
Actual Positive →	37	253

分析:

此組 Dataset 的 Positive Samples 多於 Negative Samples，且 False Positive 數量比 False Negative 高一些，原因之一可能是 Features 有 30 個但取樣的 Negative 數量太少，造成有些 underfitting。

ROC curve、AUC score(On Training Set)





分析:

Bayesian、Naïve-Baes ROC、AUC score 表現都還不錯，Bayesian 略高一點，與錯誤率的數據相同，因此 Bayesian classifier 在此 Dataset 上是最好的

IV. Pima Indians Diabetes Database

- Datasize：768
- Features：8 features
- Class: 是否有糖尿病
- Split Dataset to **Training Set：Testing Set = 8 : 2**

[Note]Linear Classifier 參數設定-> learning rate 0.000001；Iteration: 100000, Target label 設定為[-1, 1]

Training Set 上錯誤率：

Bayesian classifier: 22.5%

Naïve-Bayes classifier: 23.2%

Linear classifier: 45.8%

Testing Set 上錯誤率：

Bayesian classifier: 34.4%

Naïve-Bayes classifier: 24.1%

Linear classifier: 47.8%

3-fold-Cross-Validation 上平均錯誤率(On Training Set)：

Bayesian classifier: 25.3%

Naïve-Bayes classifier: 24.3%

Linear classifier: 36.6%

分析：

此組 Dataset 3 個 classifier 表現都很差，可能需要其他的 classifier、特徵工程、Data

preprocessing 等技巧才能有好的表現。其中 Linear classifier 還是最差的(線性分類 Bias 太大)。在 Training Set 表現上 Bayesian 、Naïve-Bayes 都差不多，但在 Testing Set 的表現上 Naïve-Bayes 反而是比較好的，推估 Bayesian classifier 考慮了 features 相關性但卻造成比 Naïve-Bayes 更嚴重的 overfitting，反而沒有考慮 features 相關性的 Naïve-Bayes 平均表現更好。

Confusion Matrix (On Training Set)

Bayesian classifier

	Predict Negative↓	Predict Positive↓
Actual Negative→	314	59
Actual Positive →	85	129

Naïve-Bayes classifier:

	Predict Negative↓	Predict Positive↓
Actual Negative→	342	58
Actual Positive →	89	125

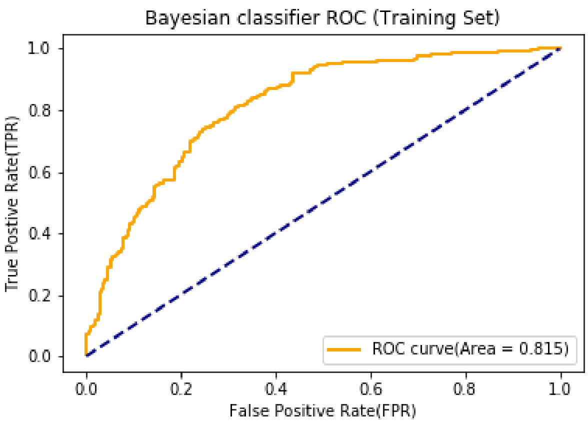
linear classifier:

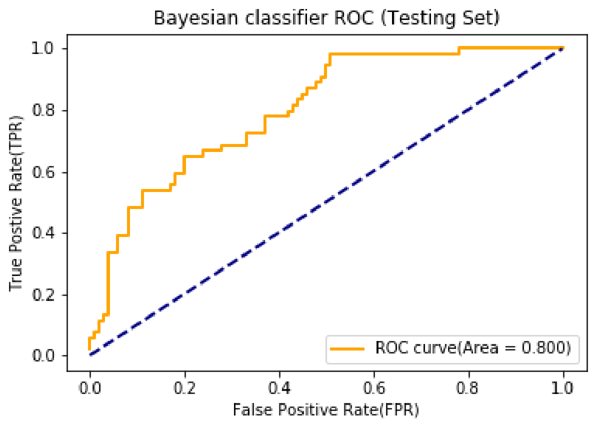
	Predict Negative↓	Predict Positive↓
Actual Negative→	327	73
Actual Positive →	153	61

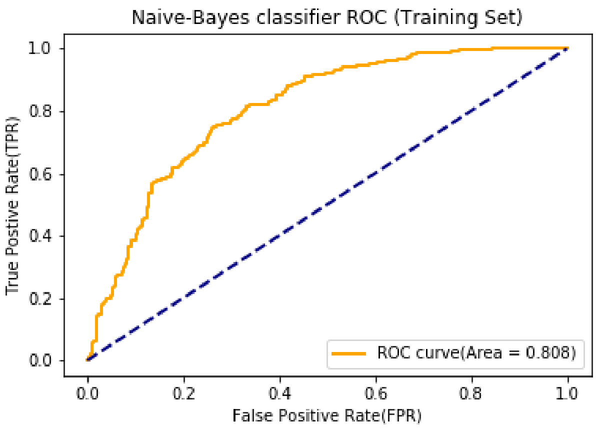
分析:

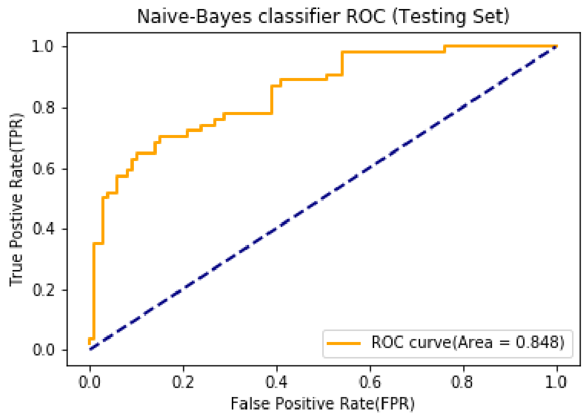
False Positive 數量明顯比 False Negative 高，原因之一應該取樣的 Actual Positive 數量(268 個 Samples)比 Actual Negative(500 個 Samples)少很多，取樣的沒辦法有效涵蓋所有 Positive 分布。

ROC curve、AUC score(On Training Set、Testing Set)









分析:

上排 2 張圖為 Bayesian Classifier 在 Training Data、Testing Data 的表現，可以看到 Bayesian 的 AUC score 在 Testing 比較低，但是下兩張圖 Naïve-Bayes 的 AUC score 在 Testing Set 反而比在 Training Set 更高；此分數與錯誤率數據相同，在 Naïve-Bayes 在 Testing Data 上跟在 Training Data 上表現一樣，但是 Bayesian 卻是有差距的，同樣驗證 Bayesian 考慮 Features 的關聯產生了 overfitting、但 Naïve-Bayes 反而比較因此不受影響，展現了該 Classifier 的特性與優點。

2.Model implementation (API)

分別將 Bayesian classifier、Naïve-Bayes classifier、linear classifier、Evaluation、cross-validation 實作為 python class object，**三個 classifier 皆有相同 Methods (Common interface, 但 linear classifier 的 output 略不同)** 以下列出其使用方法：(參考 python scikit-learn API interface 實作之)

***以下 X 參數皆代表 data, y 代表對應的 label (class)**

Bayesian classifier

Methods:

fit(self, X, y): Fit Gaussian Bayesian classifier according to X, y(Training Data and label)

predict(self, X): Perform classification on an array of test data X.(預測 test data class)

predict_proba(self, X): Return probability estimates for the test data X.(Values of the discriminant functions for each class)

score(self, X, y): Return the accuracy on the given test data and labels.(計算準確率)

Naïve-Bayes classifier

Methods: [Note] Methods 與 Bayesian classifier 完全相同

fit(self, X, y): Fit Gaussian Bayesian classifier according to X, y(Training Data and label)

predict(self, X): Perform classification on an array of test data X.(預測 test data class)

predict_proba(self, X): Return probability estimates for the test data X.(Values of the discriminant functions for each class)

score(self, X, y): Return the accuracy on the given test data and labels.(計算準確率)

linear classifier

Parameters:

eta: learning rate (default 0.01)

n_iterations: The number of iterations of Gradient Descent times. (default 1000)

Attribute:

w_: Weights assigned to the features. Initial by uniform distribution [0, 1)

Methods:

fit(self, X, y): Fit linear model with Stochastic Gradient Descent.

predict(self, X): Predict using the linear model (Regression->數值, 非 class label)

predict_label(self, X): Predict using the linear model (Predict class label)

score(self, X, y): Return the accuracy on the given test data and labels.(計算準確率)[Note: 只可用在 binary classification, label 為{1, -1}, 以 step function 為 activation function 做 class 預測]

Evaluation:

計算 confusion matrix, roc curve, auc score。.

Methods:

confusion_matrix(self, y_true, y_pred): 計算 confusion matrix

- y_true: Ground truth (correct) target values.
- y_pred: Estimated targets as returned by a classifier.

roc_curve (self, y_true, y_score): 計算 ROC curve (binary classification 限定)

- y_score: Probability estimates of the positive class (Values of the discriminant functions for positive class)

roc_auc_score(self, y_true, y_score): 計算 AUC 值 (binary classification 限定)

Cross Validation:

score(self, estimator, X, y, cv): 使用 Cross Validation 計算 n 次準確率

- estimator: estimator(classifier) object implementing `fit`
- cv: How many subset is used for validation (n-fold, 決定 n 值)

3. Appendix

Code in My Github link:

https://github.com/Yunyung/BayesianClassifier-NaiveBayesClassifier-linearClassifier-ImplementationFromScratch

Code

- <u>Gaussian Bayesian Classifier Module</u>

```
import numpy as np

class GaussianBC:
    def __init__(self):
        self.numOfClasses = None
        self.numOfFeatures = None
        self.means = None
        self.cov = None
        self.numOfDataOfeachClass = None
        self.numOftrainingData = np.zeros(1)

    def __getInfoFromDataset(self, separated):
        """
            The function get some information from the separated dataset

            args:
                separated: dictionary object where each key is target label(class value) a
nd then add a list of all the records as the value in the dictionary
        """
        self.numOfClasses = len(separated.keys())  # Get the number of class
        for i in range(self.numOfClasses):
            if len(separated[i]) != 0:
                self.numOfFeatures = len(separated[i][0])
                break

        self.numOfDataOfeachClass = np.zeros(self.numOfClasses)
        for i in range(self.numOfClasses):
            numOfDataOfClass = len(separated[i])
            self.numOfDataOfeachClass[i] = numOfDataOfClass
            self.numOftrainingData += numOfDataOfClass

    def __separate_by_class(self, X, Y):
        """
            This function split the dataset by class values, return dictionary
```

```
        args:
            X: training data
            Y: target label(class value)
        return:
            separated: dictionary object where each key is target label(class value) a
nd then add a list of all the records as the value in the dictionary
        """

        separated = dict()
        for i in range(len(X)):
            vector = X[i]
            class_value = Y[i]
            if (class_value not in separated):
                separated[class_value] = list()
            separated[class_value].append(vector)

        self.__getInfoFromDataset(separated)

        return separated

    def __summarize_dataset(self, separated):
        """
            Calculate the mean, cov and count for each column(feature) in separated datase
t


            args:
                separated: dictionary object where each key is target label(class value) a
nd then add a list of all the records as the value in the dictionary


            returns:
                means    : The means of each class
                cov: The covariance maxtirx of each class
        """

        means = np.zeros(shape=(self.numOfClasses, self.numOfFeatures))
        cov = np.zeros(shape=(self.numOfClasses, self.numOfFeatures, self.numOfFeatures))
        for class_value, rows in separated.items():
            # The mean for each input feature
            means[class_value] = np.mean(rows, axis=0)
            cov[class_value] = np.cov(np.array(rows).T)
```

```
        return means, cov

    def __multivariate_gaussian_pdf(self, X, mean, cov):
        """
            Returns the pdf of a multivariate gaussian distribution
        """

        cov_inv = np.linalg.inv(cov)
        denominator = np.sqrt(((2 * np.pi)**self.numOfFeatures) * np.linalg.det(cov))
        exponent = -(1/2) * ((X - mean) @ cov_inv @ (X - mean).T)

        return (1 / denominator) * np.exp(exponent)

    def fit(self, X, y):
        """
            The fitting function

            args:
                X : array-like, shape = [n_samples, n_features]
                Training samples
                y : array-like, shape = [n_samples,]
                    Target values
            return:
        """

        X = np.array(X)
        y = np.array(y)
        separated = self.__separate_by_class(X, y)
        self.means, self.cov = self.__summarize_dataset(separated)

    def predict(self, X):

        numOfTest = X.shape[0]
        best_labels = np.full(numOfTest, -1)
        # print("The Number of test data : ", numOfTest)
        for i in range(numOfTest):
            best_label, best_prob = None, -np.inf
            for j in range(self.numOfClasses):
                probability = (self.numOfDataOfeachClass[j] / self.numOftrainingData)
                probability *= self.__multivariate_gaussian_pdf(X[i], self.means[j], self.
cov[j])
```

```python
                if (best_prob < probability):
                    best_prob = probability
                    best_label = j
            best_labels[i] = best_label
        return best_labels

    def predict_proba(self, X):

        X = np.atleast_2d(X) # numpy array and make sure at least two dimenison

        numOfTest = X.shape[0]

        probs = np.full((numOfTest, self.numOfClasses), np.inf)

        for i in range(numOfTest):
            for j in range(self.numOfClasses):
                probability = (self.numOfDataOfeachClass[j] / self.numOftrainingData)
                probability *= self.__multivariate_gaussian_pdf(X[i], self.means[j], self.
cov[j])

                probs[i, j] = probability

        # Normalization, each element divide by sum of all elements
        for i in range(numOfTest):
            row = probs[i]
            total_prob = np.sum(row)
            # print(total_prob)
            probs[i] = row / total_prob

        return probs

    def score(self, X, Y):
        """
            Return the  accuracy on the given test data and labels.

            args:
                X: test data
                Y: ground-truth labels

            return:
                accuracy
        """
```

```
        X = np.array(X)  # translate testing data to numpy array

        count_correct = (self.predict(X) == Y).sum()
        return count_correct / X.shape[0]
```

- ## Gaussian Naïve-Bayes Classifier Module

```
import numpy as np
from sklearn.cross_validation import train_test_split

class GaussianNaiveBayes:
    def __init__(self):
        self.numOfClasses = None
        self.numOfFeatures = None
        self.means = None
        self.cov = None
        self.numOfDataOfeachClass = None
        self.numOftrainingData = np.zeros(1)

    def __getInfoFromDataset(self, separated):
        """
            The function get some information from the separated dataset

            args:
                separated: dictionary object where each key is target label(class value) a
nd then add a list of all the records as the value in the dictionary
        """

        self.numOfClasses = len(separated.keys())  # Get the number of class
        for i in range(self.numOfClasses):
            if len(separated[i]) != 0:
                self.numOfFeatures = len(separated[i][0])
                break

        self.numOfDataOfeachClass = np.zeros(self.numOfClasses)
        for i in range(self.numOfClasses):
            numOfDataOfClass = len(separated[i])
            self.numOfDataOfeachClass[i] = numOfDataOfClass
            self.numOftrainingData += numOfDataOfClass

    def __separate_by_class(self, X, Y):
        """
            This function split the dataset by class values, return dictionary
```

```
            args:
                X: training data
                Y: target label(class value)
            return:
                separated: dictionary object where each key is target label(class value) a
nd then add a list of all the records as the value in the dictionary
        """

        separated = dict()
        for i in range(len(X)):
            vector = X[i]
            class_value = Y[i]
            if (class_value not in separated):
                separated[class_value] = list()
            separated[class_value].append(vector)

        self.__getInfoFromDataset(separated)
        # print("Separated Data:", separated)

        return separated

    def __summarize_dataset(self, separated):
        """
            Calculate the mean, cov and count for each column(feature) in separated datase
t

            args:
                separated: dictionary object where each key is target label(class value) a
nd then add a list of all the records as the value in the dictionary

            returns:
                means    : The means of each class
                cov: The covariance maxtirx of each class
        """
        means = np.zeros(shape=(self.numOfClasses, self.numOfFeatures))
        cov = np.zeros(
            shape=(self.numOfClasses, self.numOfFeatures, self.numOfFeatures))
        for class_value, rows in separated.items():
            # The mean for each input feature
            means[class_value] = np.mean(rows, axis=0)
            cov[class_value] = np.cov(np.array(rows).T, ddof=0)
```

```
            # For a Naive Bayes classifier, we can estimate the variance of each feature i
ndependently.
            # However It has the same effect of making all the non-
diagonal elements of the covariance matrix zero.
            for i in range(self.numOfFeatures):
                for j in range(self.numOfFeatures):
                    if (i != j):
                        cov[class_value, i, j] = 0

        # print(means)
        # print(cov)

        return means, cov

    def __multivariate_gaussian_pdf(self, X, mean, cov):
        """
            Returns the pdf of a multivariate gaussian distribution
        """

        cov_inv = np.linalg.inv(cov)
        denominator = np.sqrt(
            ((2 * np.pi)**self.numOfFeatures) * np.linalg.det(cov))
        exponent = -(1/2) * ((X - mean) @ cov_inv @ (X - mean).T)

        return (1 / denominator) * np.exp(exponent)

    def fit(self, X, y):
        """
            The fitting function

            args:
                X : array-like, shape = [n_samples, n_features]
                Training samples
                y : array-like, shape = [n_samples,]
                    Target values
            return:
        """
        X = np.array(X)
        y = np.array(y)
        separated = self.__separate_by_class(X, y)
        self.means, self.cov = self.__summarize_dataset(separated)
```

```
    def predict(self, X):

        # translate testing data to numpy array and make sure at least two dimenison
        X = np.atleast_2d(X)

        # print(X)
        numOfTest = X.shape[0]
        best_labels = np.full(numOfTest, -1)
        # print("The Number of test data : ", numOfTest)
        for i in range(numOfTest):
            best_label, best_prob = None, -np.inf
            for j in range(self.numOfClasses):
                probability = (self.numOfDataOfeachClass[j] / self.numOftrainingData)
                probability *= self.__multivariate_gaussian_pdf(X[i], self.means[j], self.
cov[j])


                if (best_prob < probability):
                    best_prob = probability
                    best_label = j
            best_labels[i] = best_label
        return best_labels

    def predict_proba(self, X):
        # numpy array and make sure at least two dimenison
        X = np.atleast_2d(X)

        numOfTest = X.shape[0]
        # print("The Number of test data : ", numOfTest)

        probs = np.full((numOfTest, self.numOfClasses), np.inf)

        for i in range(numOfTest):
            for j in range(self.numOfClasses):
                probability = (
                    self.numOfDataOfeachClass[j] / self.numOftrainingData)
                probability *= self.__multivariate_gaussian_pdf(
                    X[i], self.means[j], self.cov[j])

                probs[i, j] = probability

        # Normalization, each element divide by sum of all elements
```

```python
        for i in range(numOfTest):
            row = probs[i]
            total_prob = np.sum(row)
            # print(total_prob)
            probs[i] = row / total_prob

        return probs

    def score(self, X, Y):
        """
            Return the  accuracy on the given test data and labels.

            args:
                X: test data
                Y: ground-truth labels

            return:
                accuracy
        """
        X = np.array(X)  # translate testing data to numpy array

        count_correct = (self.predict(X) == Y).sum()
        return count_correct / X.shape[0]
```

- <u>Linear classifier Module</u>

[Note]採用 MSE 為 loss 以 Gradient Descent 做 optimization(Batch version)。

```python
import numpy as np

class linearClassifierGD:
    """
        Learning Regression Using Gradient Desent (Batch Version)

    Parameter
    ------------
    eta: float
        constant learning rate

    n_iterations: int
        # of passes over the training set
```

```
    Attributes
    ------------
    w_ :
        weights of fitting the model

    cost_: total error of model after each iteration
    """

    def __init__(self,  eta=0.01, n_iterations=1000):
        self.eta = eta
        self.n_iterations = n_iterations

    def fit(self, X, y):
        """
            The fitting function

            args:
                X : array-like, shape = [n_samples, n_features]
                Training samples
                y : array-like, shape = [n_samples,]
                    Target values
            return:
        """
        # add bias term to X
        X = np.hstack((np.ones((X.shape[0], 1)), X))

        self.cost_ = []
        self.w_ = np.random.rand(X.shape[1])

        m = X.shape[0]
        # gradient desent
        for _ in range(self.n_iterations):
            y_pred = np.dot(X, self.w_)
            residuals = y_pred - y
            gradient_vector = np.dot(X.T, residuals)
            self.w_ = self.w_ - (self.eta / m) * gradient_vector
            # record cost in each iteration
            cost = np.sum(residuals ** 2) / (2 * m)

            # Set threshold to stop
            if cost <= 1e-8:  # converge
                print("Linear Classifier has been converge!")
```

```
                break
            if cost >= 1e+100: # diverge
                print("*Linear Classifier has been Diverge!")
                break
            self.cost_.append(cost)

    def predict(self, X):
        """
        Predicts the value after the model has been trained.
        args:
            x : array-like, shape = [n_samples, n_features]
                Test samples

        Returns:
            Predicted value
        """

        # add bias term to X
        X = np.hstack((np.ones((X.shape[0], 1)), X))

        return self.__step_function(np.dot(X, self.w_))

    def predict_regressionValue(self, X):
        """
        Predicts the value after the model has been trained.
        Predict Regression value, not class label
        args:
            x : array-like, shape = [n_samples, n_features]
                Test samples

        Returns:
            Predicted value
        """

        # add bias term to X
        X = np.hstack((np.ones((X.shape[0], 1)), X))

        return np.dot(X, self.w_)

    def __step_function(self, X):
        """
```

```
            Step activiation function used for two class(label:1 and -
1) classification with threshold->0
            args:
                X: Predicted value using the linear model
        """
        labels = np.zeros(X.shape[0])

        labels[X >= 0] = 1
        labels[X < 0] = -1

        return labels

    def score(self, X, Y):
        """
            Calculate accuracy(score) used for two class(label:1 and -
1) classification with step function
            Note: this function only apply to two class classfication
            args:
                X : array-like, shape = [n_samples, n_features]
                Training samples
                y : array-like, shape = [n_samples,]
                    Target values
            return:
                Accuracy value
        """
        # add bias term to X
        count_correct = (self.__step_function(self.predict(X)) == Y).sum()
        return count_correct / X.shape[0]
```

- Cross-Validation Module

```
import numpy as np
from sklearn.model_selection import KFold

class cross_validate:
    """
        Evaluate fit/score by cross-validataion

        Parameter
        ------------
        estimator: estimator object implementing 'fit'
            The object to use fit the data
```

```
        X : array-like, shape = [n_samples, n_features]
                Dataset, Training samples

        y : array-like
            The target variable to try to predict in the case of supervied learning

        cv: int
            How many fold(split). Default 5-fold cross validation
    """


    def score(self, estimator, X, y, cv = 5):
        score_list = list()
        kf = KFold(n_splits = cv, shuffle=True)
        # Calculate each fold score
        for train_index, test_index in kf.split(X):
            #print("TRAIN:", train_index, "TEST:", test_index)
            #print(y[train_index])
            estimator.fit(X[train_index], y[train_index])
            score = estimator.score(X[test_index], y[test_index])
            score_list.append(score)
        return score_list
```

- Evaluation Module (Confusion Matrix, ROC curve, AUC score)

```
import numpy as np
import matplotlib.pyplot as plt


class Evaluation:
    """
        Evaluation Module
    """

    def __init__(self):
        pass
```

```python
    def confusion_matrix(self, y_true, y_pred):
        """
            Calculate  class confusion Matrix to evaluation classifier. Can apply to multi
 classes dataset
            Note: This function assume that label from 0 to N(The Maximim of label value)
            args:
                y_true: Ground truth (correct) target values.
                y_pred: Estimated targets as returned by a classifier

            returns:
                Confusion matrix: narray of shape:(n_classs, n_classes)
        """
        # find the maximun of label value
        max_label_value = np.max(y_true)

        print("max_label_value:", max_label_value)
        confusionMatrix = np.zeros((max_label_value + 1, max_label_value + 1))
        print(confusionMatrix)
        # buidl confusion Matrix
        for i in range(len(y_true)):
            confusionMatrix[y_true[i], y_pred[i]] += 1

        return confusionMatrix

    def roc_curve(self, y_true, y_score):
        """
            Plot roc_curve to evaluation classifier
            Note: this implementation is restricted to the binary classification task.

            args:
                y_true: True binary labels. If labels are not either {-
1, 1} or {0, 1}, then pos_label should be explicitly given.
                y_score: Target scores, can either be probability estimates of the positiv
e class

            return:
                fpr : Increasing false positive rates such that element i is the false pos
itive rate of predictions with score >= thresholds[i].
                tpr : Increasing true positive rates such that element i is the true posit
ive rate of predictions with score >= thresholds[i].
                threshold : Decreasing thresholds on the decision function used to compute
 fpr and tpr.
```

```
        """

        tpr_list = []
        fpt_list = []
        thresholds = np.linspace(1.1, 0, 10)

        for t in thresholds:
            y_pred = np.zeros(y_true.shape[0])
            # print(y_score)
            y_pred[y_score >= t] = 1
            TP = y_pred[(y_pred == y_true) & (y_true == 1)].shape[0]
            TN = y_pred[(y_pred == y_true) & (y_true == 0)].shape[0]
            FN = y_pred[(y_pred != y_true) & (y_true == 1)].shape[0]
            FP = y_pred[(y_pred != y_true) & (y_true == 0)].shape[0]
            TPR = TP / (TP + FN)
            FPR = FP / (FP + TN)
            tpr_list.append(TPR)
            fpt_list.append(FPR)

        return tpr_list, fpt_list, thresholds

    def plot_roc_curve(self, y_true, y_score):
        """
            plot roc curve
        """
        tpr_list, fpt_list, thresholds = self.roc_curve(y_true, y_score)

        plt.plot(fpt_list, tpr_list, 'b')
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.show()

    def roc_auc_score(self, y_true, y_score):
        """
            Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from
prediction scores.
            Note: this implementation can be used with binary,
        """
        tpr_list, fpt_list, thresholds = self.roc_curve(y_true, y_score)

        print(tpr_list)
        print(fpt_list)
```

```
        score = np.zeros(1)
        for i in range(len(tpr_list) - 1):
            score += (fpt_list[i + 1] - fpt_list[i]) * (tpr_list[i + 1])
        return score
```

- <u>Main Program</u> (For Experiment and Analysis)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cross_validation import train_test_split
from sklearn import datasets
from sklearn import metrics

# my implement modules
from evaluation import Evaluation
from cross_validate import cross_validate
from linearClassifierGD import linearClassifierGD
from GaussianBayesianClassifier import GaussianBC
from GaussianNaiveBayes import GaussianNaiveBayes

def binary_Classification_analysis(X_train, y_train, X_test, y_test):
    BCmodel = GaussianBC()
    NBmodel = GaussianNaiveBayes()
    BCmodel.fit(X_train, y_train)
    NBmodel.fit(X_train, y_train)
    print("Training Data Score-
> BC:", BCmodel.score(X_train, y_train), ", NB:", NBmodel.score(X_train, y_train))
    print("Testing Data Score-
> BC:", BCmodel.score(X_test, y_test), ", NB:", NBmodel.score(X_test, y_test))
    print("Average Score of Cross-
Validation : BC:", np.sum(cv.score(BCmodel, X_train, y_train, 3)) / 3, " NB:", np.sum(cv.s
core(NBmodel, X_train, y_train, 3)) / 3)


    # (Training Set) Calculate Bayesian Classification confusion Matrix、
ROC curve and AUC score, and plot it
    print("Bayesian Classifier confusion Matrix (Training Set):\n", metrics.confusion_matr
ix(y_train, BCmodel.predict(X_train)))
    fpr, tpr, thresholds = metrics.roc_curve(y_train, BCmodel.predict_proba(X_train)[:, 1]
, pos_label=1)
```

```python
    score = metrics.roc_auc_score(y_train, BCmodel.predict_proba(X_train)[:, 1])
    fig = plt.figure(1)
    plt.plot(fpr, tpr, color='orange', lw=2, label="ROC curve(Area = %0.3f)" % score)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.title("Bayesian classifier ROC (Training Set)")
    plt.legend(loc = 'lower right')
    plt.xlabel('False Positive Rate(FPR)')
    plt.ylabel('True Postive Rate(TPR)')
    plt.savefig("Bayesian classifier ROC(Training Set).png")

    # (Testing Set) Calculate Bayesian Classification confusion Matrix、ROC curve and AUC score, and plot it
    print("Bayesian Classifier confusion Matrix (Testing Set) :\n", metrics.confusion_matrix(y_test, BCmodel.predict(X_test)))
    fpr, tpr, thresholds = metrics.roc_curve(y_test, BCmodel.predict_proba(X_test)[:, 1], pos_label=1)
    score = metrics.roc_auc_score(y_test, BCmodel.predict_proba(X_test)[:, 1])
    fig2 = plt.figure(2)
    plt.plot(fpr, tpr, color='orange', lw=2, label="ROC curve(Area = %0.3f)" % score)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.title("Bayesian classifier ROC (Testing Set)")
    plt.legend(loc = 'lower right')
    plt.xlabel('False Positive Rate(FPR)')
    plt.ylabel('True Postive Rate(TPR)')
    plt.savefig("Bayesian classifier ROC(Testing Set).png")

    # (Training Set) Calculate Naive-Bayes classification confusion Matrix、ROC curve and AUC score, and plot it
    print("Naive-Bayes Classifier confusion Matrix (Training Set) :\n", metrics.confusion_matrix(y_train, NBmodel.predict(X_train)))
    fpr, tpr, thresholds = metrics.roc_curve(y_train, NBmodel.predict_proba(X_train)[:, 1], pos_label=1)
    score = metrics.roc_auc_score(y_train, NBmodel.predict_proba(X_train)[:, 1])
    fig3 = plt.figure(3)
    plt.plot(fpr, tpr, color='orange', lw=2, label="ROC curve(Area = %0.3f)" % score)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.title("Naive-Bayes classifier ROC (Training Set)")
    plt.legend(loc = 'lower right')
    plt.xlabel('False Positive Rate(FPR)')
    plt.ylabel('True Postive Rate(TPR)')
    plt.savefig("Naive-Bayes classifier ROC(Training Set).png")
```

```python
    # (Testing Set) Calculate Naive-Bayes classification confusion Matrix、ROC curve and AUC score, and plot it
    print("Naive-Bayes Classifier confusion Matrix (Testing Set) :\n", metrics.confusion_matrix(y_test, NBmodel.predict(X_test)))
    fpr, tpr, thresholds = metrics.roc_curve(y_test, NBmodel.predict_proba(X_test)[:, 1], pos_label=1)
    score = metrics.roc_auc_score(y_test, NBmodel.predict_proba(X_test)[:, 1])
    fig4 = plt.figure(4)
    plt.plot(fpr, tpr, color='orange', lw=2, label="ROC curve(Area = %0.3f)" % score)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.title("Naive-Bayes classifier ROC (Testing Set)")
    plt.legend(loc = 'lower right')
    plt.xlabel('False Positive Rate(FPR)')
    plt.ylabel('True Postive Rate(TPR)')
    plt.savefig("Naive-Bayes classifier ROC(Testing Set).png")


    plt.figure(figsize=(20, 20))
    plt.show()

    # linear classifier
    LCmodel = linearClassifierGD(eta=0.000001, n_iterations = 100000)
    LCmodel.fit(X_train, y_train)
    # label data to [-1 1] for linear classficaition
    y_train[y_train == 0] = -1
    y_test[y_test == 0] = -1
    print("Training Data Score-> Linear Classifier:", LCmodel.score(X_train, y_train))
    print("Testing Data Score-> Linear Classifier:", LCmodel.score(X_test, y_test))
    print("Average Score of Cross-Validation : Linear Classifier:", np.sum(cv.score(LCmodel, X_train, y_train, 3)) / 3)
    # Calculate linear classifier confusion Matrix
    print("linear Classifier confusion Matrix (Training Set) :\n", metrics.confusion_matrix(y_train, LCmodel.predict(X_train)))
    print("linear Classifier confusion Matrix (Testing Set) :\n", metrics.confusion_matrix(y_test, LCmodel.predict(X_test)))

if __name__ == '__main__':
    # Dataset1 - iris dataset
    print("----------1. Iris Dataset-----------")
    iris = datasets.load_iris()  # load iris dataset
```

```
    X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size
= 0.2, random_state=42)  # Split dataset into trianing and testing randomly
    BCmodel = GaussianBC()
    NBmodel = GaussianNaiveBayes()
    BCmodel.fit(X_train, y_train)
    NBmodel.fit(X_train, y_train)
    cv = cross_validate()
    print("Training Data Score-
> BC:", BCmodel.score(X_train, y_train), ", NB:", NBmodel.score(X_train, y_train))
    print("Testing Data Score-
> BC:", BCmodel.score(X_test, y_test), ", NB:", NBmodel.score(X_test, y_test))
    print("Average Score of Cross-
Validation : BC:", np.sum(cv.score(BCmodel, X_train, y_train, 3)) / 3, " NB:", np.sum(cv.s
core(NBmodel, X_train, y_train, 3)) / 3)

    # Dataset2 - Wheat-Seeds Dataset
    print("----------2. Wheat-Seeds Dataset-----------")
    seeds=pd.read_csv('seeds.csv')
    train, test = train_test_split(seeds, test_size=0.2) # Split dataset into trianing and
 testing randomly

    # Data spliting(split features and ground-true label) and processing
    X_train = train.iloc[:, :7]
    y_train = train.iloc[:, 7] - 1  # training data label (All value minus one, Because or
iginally 1 ~ 3)
    X_test = test.iloc[:, :7]
    y_test = test.iloc[:, 7] - 1   # testing data label (All value minus one, Because orig
inally 1 ~ 3)
    X_train, y_train, X_test, y_test = np.array(X_train), np.array(y_train), np.array(X_te
st), np.array(y_test)

    BCmodel = GaussianBC()
    NBmodel = GaussianNaiveBayes()
    BCmodel.fit(X_train, y_train)
    NBmodel.fit(X_train, y_train)
    cv = cross_validate()
    print("Training Data Score-
> BC:", BCmodel.score(X_train, y_train), ", NB:", NBmodel.score(X_train, y_train))
    print("Testing Data Score-
> BC:", BCmodel.score(X_test, y_test), ", NB:", NBmodel.score(X_test, y_test))
```

```
    print("Average Score of Cross-
Validation : BC:", np.sum(cv.score(BCmodel, X_train, y_train, 3)) / 3, " NB:", np.sum(cv.s
core(NBmodel, X_train, y_train, 3)) / 3)


    # Dataset3 - Breast Cancer Wisconsin (Diagnostic) DataSet
    print("----------3. Breast Cancer Wisconsin (Diagnostic) DataSet------------")
    breast_cancer = datasets.load_breast_cancer()
    X_train, X_test, y_train, y_test = train_test_split(breast_cancer.data, breast_cancer.
target, test_size=0.2, stratify=breast_cancer.target)
    binary_Classification_analysis(X_train, y_train, X_test, y_test)

    # Dataset4 - Pima Indians Diabetes Database
    print("----------4. Pima Indians Diabetes Database------------")
    diab = pd.read_csv('diabetes.csv')
    print(diab.isnull().sum()) # checking the data
    outcome = diab['Outcome']
    data = diab[diab.columns[:8]]
    train, test = train_test_split(diab, test_size=0.2, stratify=diab['Outcome']) #stratif
y the outcome
    X_train = train[train.columns[:8]]
    y_train = train['Outcome']
    X_test = test[test.columns[:8]]
    y_test = test['Outcome']
    X_train, y_train, X_test, y_test = np.array(X_train), np.array(y_train), np.array(X_te
st), np.array(y_test)
    binary_Classification_analysis(X_train, y_train, X_test, y_test)
```