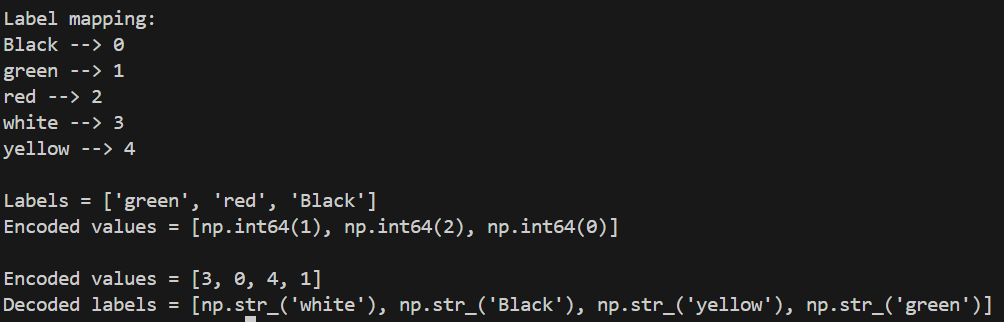
ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити попередню обробку та класифікацію даних.

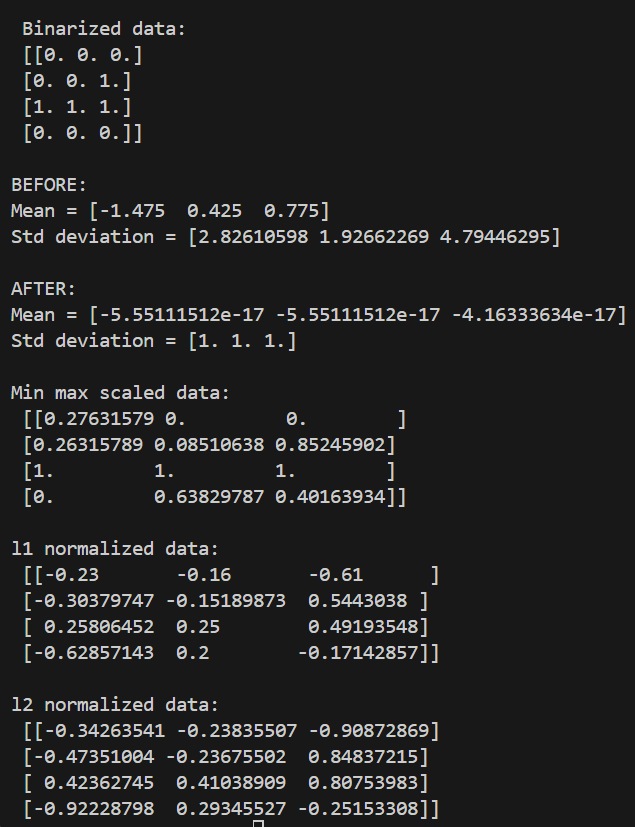
LR\_1\_task\_1.py

|  |
| --- |
| from sklearn import preprocessing  # Надання позначок вхідних даних  input\_labels = ["red", "Black", "red", "green", "Black", "yellow", "white"]  # Створення кодувальника та встановлення відповідності  # між мітками та числами  encoder = preprocessing.LabelEncoder()  encoder.fit(input\_labels)  # Виведення відображення  print("\nLabel mapping:")  for i, item in enumerate(encoder.classes\_):      print(item, "-->", i)  # перетворення міток за допомогою кодувальника  test\_labels = ["green", "red", "Black"]  encoded\_values = encoder.transform(test\_labels)  print("\nLabels =", test\_labels)  print("Encoded values =", list(encoded\_values))  # Декодування набору чисел за допомогою декодера  encoded\_values = [3, 0, 4, 1]  decoded\_list = encoder.inverse\_transform(encoded\_values)  print("\nEncoded values =", encoded\_values)  print("Decoded labels =", list(decoded\_list)) |



LR\_1\_task\_2.py

|  |
| --- |
| import numpy as np  from sklearn import preprocessing  input\_data = np.array(      [          [-2.3, -1.6, -6.1],          [-2.4, -1.2, 4.3],          [3.2, 3.1, 6.1],          [-4.4, 1.4, -1.2]       ]  )  # Бінаризація даних  data\_binarized = preprocessing.Binarizer(threshold=2.1).transform(input\_data)  print("\n Binarized data:\n", data\_binarized)  # Виведення середнього значення та стандартного відхилення  print("\nBEFORE: ")  print("Mean =", input\_data.mean(axis=0))  print("Std deviation =", input\_data.std(axis=0))  # Виключення середнього  data\_scaled = preprocessing.scale(input\_data)  print("\nAFTER: ")  print("Mean =", data\_scaled.mean(axis=0))  print("Std deviation =", data\_scaled.std(axis=0))  # Масштабування MinМax  data\_scaler\_minmax = preprocessing.MinMaxScaler(feature\_range=(0, 1))  data\_scaled\_minmax = data\_scaler\_minmax.fit\_transform(input\_data)  print("\nМin max scaled data:\n", data\_scaled\_minmax)  # Нормалізація даних  data\_normalized\_l1 = preprocessing.normalize(input\_data, norm="l1")  data\_normalized\_l2 = preprocessing.normalize(input\_data, norm="l2")  print("\nl1 normalized data:\n", data\_normalized\_l1)  print("\nl2 normalized data:\n", data\_normalized\_l2) |



LR\_1\_task\_3.py

|  |
| --- |
| import numpy as np  from sklearn import linear\_model  import matplotlib.pyplot as plt  from utilities import visualize\_classifier  # Визначення зразка вхідних даних  X = np.array(      [          [3.1, 7.2],          [4, 6.7],          [2.9, 8],          [5.1, 4.5],          [6, 5],          [5.6, 5],          [3.3, 0.4],          [3.9, 0.9],          [2.8, 1],          [0.5, 3.4],          [1, 4],          [0.6, 4.9],      ]  )  y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])  # Створення логістичного класифікатора  classifier = linear\_model.LogisticRegression(solver='liblinear',C=1)  # Тренування класифікатора  classifier.fit(X, y)  visualize\_classifier(classifier, X, y) |

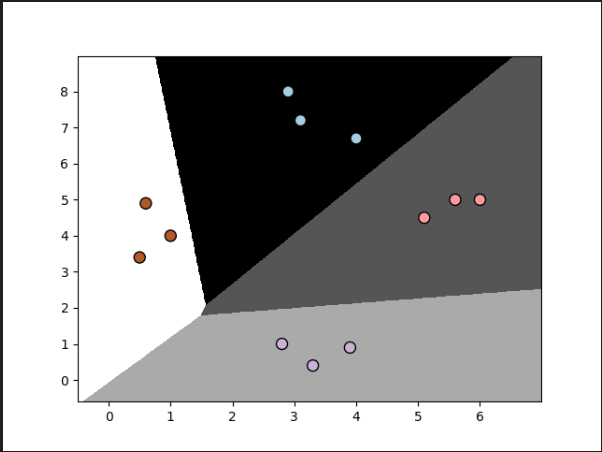
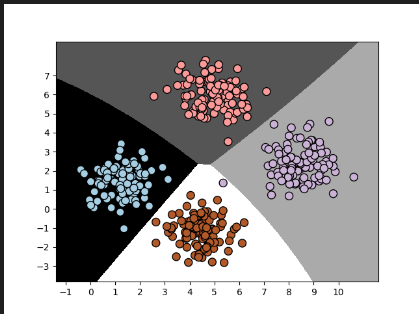


Рис. 4 результат роботи класифікатора

LR\_1\_task\_4.py

|  |
| --- |
| import numpy as np  from sklearn.model\_selection import cross\_val\_score, train\_test\_split  from sklearn.naive\_bayes import GaussianNB  from utilities import visualize\_classifier  # Вхідний файл, який містить дані  input\_file = "data\_multivar\_nb.txt"  # Завантаження даних із вхідного файлу  data = np.loadtxt(input\_file, delimiter=",")  X, y = data[:, :-1], data[:, -1]  # Створення наївного байєсовського класифікатора  classifier = GaussianNB()  # Тренування класифікатора  classifier.fit(X, y)  # Прогнозування значень для тренувальних даних  y\_pred = classifier.predict(X)  # Обчислення якості класифікатора  accuracy = 100.0 \* (y == y\_pred).sum() / X.shape[0]  print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")  # Візуалізація результатів роботи класифікатора  visualize\_classifier(classifier, X, y, "task4.1")  # Розбивка даних на навчальний та тестовий набори  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=3)  classifier\_new = GaussianNB()  classifier\_new.fit(X\_train, y\_train)  y\_test\_pred = classifier\_new.predict(X\_test)  # Обчислення якості класифікатора  accuracy = 100.0 \* (y\_test == y\_test\_pred).sum() / X\_test.shape[0]  print("Accuracy of the new classifier =", round(accuracy, 2), "%")  # Візуалізація роботи класифікатора  visualize\_classifier(classifier\_new, X\_test, y\_test, "task4.2")  num\_folds = 3  accuracy\_values = cross\_val\_score(classifier, X, y, scoring="accuracy", cv=num\_folds)  print("Accuracy: " + str(round(100 \* accuracy\_values.mean(), 2)) + "%")  precision\_values = cross\_val\_score(      classifier, X, y, scoring="precision\_weighted", cv=num\_folds  )  print("Precision: " + str(round(100 \* precision\_values.mean(), 2)) + "%")  recall\_values = cross\_val\_score(      classifier, X, y, scoring="recall\_weighted", cv=num\_folds  )  print("Recall: " + str(round(100 \* recall\_values.mean(), 2)) + "%")  f1\_values = cross\_val\_score(classifier, X, y, scoring="f1\_weighted", cv=num\_folds)  print("F1: " + str(round(100 \* f1\_values.mean(), 2)) + "%") |
| Accuracy of Naive Bayes classifier = 99.75 %  Accuracy of the new classifier = 100.0 %  Accuracy: 99.75%  Precision: 99.76%  Recall: 99.75%  F1: 99.75% |

  
Рис. 5 – Без перехресної перевірки

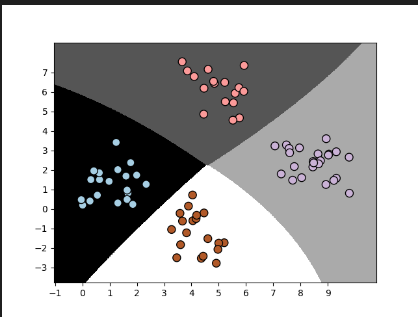
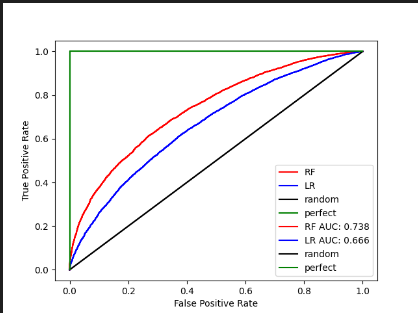


Рис. 6 – З перехресною перевіркою

Завдяки перехресній перевірці результати вийшли більш точними.

LR\_1\_task\_5.py

|  |
| --- |
| import numpy as np  import pandas as pd  from sklearn.metrics import (      accuracy\_score,      confusion\_matrix,      f1\_score,      precision\_score,      recall\_score,  )  df = pd.read\_csv("data\_metrics.csv")  df.head()  thresh = 0.5  df["predicted\_RF"] = (df.model\_RF >= 0.5).astype("int")  df["predicted\_LR"] = (df.model\_LR >= 0.5).astype("int")  df.head()  confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  def find\_TP(y\_true, y\_pred):      # counts the number of true positives (y\_true = 1, y\_pred =1)      return sum((y\_true == 1) & (y\_pred == 1))  def find\_FN(y\_true, y\_pred):      # counts the number of false negatives (y\_true = 1, y\_pred =0)      return sum((y\_true == 1) & (y\_pred == 0))  def find\_FP(y\_true, y\_pred):      # counts the number of false positives (y\_true = 0, y\_pred =1)      return sum((y\_true == 0) & (y\_pred == 1))  def find\_TN(y\_true, y\_pred):      # counts the number of true negatives (y\_true = 0, y\_pred =0)      return sum((y\_true == 0) & (y\_pred == 0))  print("TP:", find\_TP(df.actual\_label.values, df.predicted\_RF.values))  print("FN:", find\_FN(df.actual\_label.values, df.predicted\_RF.values))  print("FP:", find\_FP(df.actual\_label.values, df.predicted\_RF.values))  print("TN:", find\_TN(df.actual\_label.values, df.predicted\_RF.values))  def find\_conf\_matrix\_values(y\_true, y\_pred):      # calculate TP, FN, FP, TN      TP = find\_TP(y\_true, y\_pred)      FN = find\_FN(y\_true, y\_pred)      FP = find\_FP(y\_true, y\_pred)      TN = find\_TN(y\_true, y\_pred)      return TP, FN, FP, TN  def ozornin\_confusion\_matrix(y\_true, y\_pred):      TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)      return np.array([[TN, FP], [FN, TP]])  ozornin\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values)  assert np.array\_equal(      ozornin\_confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values),      confusion\_matrix(df.actual\_label.values, df.predicted\_RF.values),  ), "ozornin\_confusion\_matrix() is not correct for RF"  assert np.array\_equal(      ozornin\_confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values),      confusion\_matrix(df.actual\_label.values, df.predicted\_LR.values),  ), "ozornin\_confusion\_matrix() is not correct for LR"  accuracy\_score(df.actual\_label.values, df.predicted\_RF.values)  def ozornin\_accuracy\_score(y\_true, y\_pred):      # calculates the fraction of samples      TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)      return (TP + TN) / (TP + FN + FP + TN)  assert ozornin\_accuracy\_score(      df.actual\_label.values, df.predicted\_RF.values  ) == accuracy\_score(      df.actual\_label.values, df.predicted\_RF.values  ), "my\_accuracy\_score failed on RF"  assert ozornin\_accuracy\_score(      df.actual\_label.values, df.predicted\_LR.values  ) == accuracy\_score(      df.actual\_label.values, df.predicted\_LR.values  ), "my\_accuracy\_score failed on LR"  print(      "Accuracy RF:%.3f"      % (ozornin\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values))  )  print(      "Accuracy LR: %.3f"      % (ozornin\_accuracy\_score(df.actual\_label.values, df.predicted\_LR.values))  )  recall\_score(df.actual\_label.values, df.predicted\_RF.values)  def ozornin\_recall\_score(y\_true, y\_pred):      # calculates the fraction of positive samples predicted correctly      TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)      return TP / (TP + FN)  assert ozornin\_recall\_score(      df.actual\_label.values, df.predicted\_RF.values  ) == recall\_score(      df.actual\_label.values, df.predicted\_RF.values  ), "my\_accuracy\_score failed on RF"  assert ozornin\_recall\_score(      df.actual\_label.values, df.predicted\_LR.values  ) == recall\_score(      df.actual\_label.values, df.predicted\_LR.values  ), "my\_accuracy\_score failed on LR"  print(      "Recall RF: %.3f"      % (ozornin\_recall\_score(df.actual\_label.values, df.predicted\_RF.values))  )  print(      "Recall LR: %.3f"      % (ozornin\_recall\_score(df.actual\_label.values, df.predicted\_LR.values))  )  def ozornin\_precision\_score(y\_true, y\_pred):      # calculates the fraction of predicted positives samples that are actually positive      TP, FN, FP, TN = find\_conf\_matrix\_values(y\_true, y\_pred)      return TP / (TP + FP)  assert ozornin\_precision\_score(      df.actual\_label.values, df.predicted\_RF.values  ) == precision\_score(      df.actual\_label.values, df.predicted\_RF.values  ), "my\_accuracy\_score failed on RF"  assert ozornin\_precision\_score(      df.actual\_label.values, df.predicted\_LR.values  ) == precision\_score(      df.actual\_label.values, df.predicted\_LR.values  ), "my\_accuracy\_score failed on LR"  print(      "Precision RF: %.3f"      % (ozornin\_precision\_score(df.actual\_label.values, df.predicted\_RF.values))  )  print(      "Precision LR: %.3f"      % (ozornin\_precision\_score(df.actual\_label.values, df.predicted\_LR.values))  )  def ozornin\_f1\_score(y\_true, y\_pred):      # calculates the F1 score      recall = ozornin\_recall\_score(y\_true, y\_pred)      precision = ozornin\_precision\_score(y\_true, y\_pred)      return (2 \* precision \* recall) / (precision + recall)  assert ozornin\_f1\_score(df.actual\_label.values, df.predicted\_RF.values) == f1\_score(      df.actual\_label.values, df.predicted\_RF.values  ), "my\_accuracy\_score failed on RF"  print(      "F1 RF: %.3f" % (ozornin\_f1\_score(df.actual\_label.values, df.predicted\_RF.values))  )  print(      "Ozornin F1 LR: %.3f"      % (ozornin\_f1\_score(df.actual\_label.values, df.predicted\_LR.values))  )  print("Lib F1 LR: %.3f" % (f1\_score(df.actual\_label.values, df.predicted\_LR.values)))  # assert ozornin\_f1\_score(df.actual\_label.values, df.predicted\_LR.values) == f1\_score(  #     df.actual\_label.values, df.predicted\_LR.values  # ), "my\_accuracy\_score failed on LR"  print("scores with threshold = 0.5")  print(      "Accuracy RF: %.3f"      % (ozornin\_accuracy\_score(df.actual\_label.values, df.predicted\_RF.values))  )  print(      "Recall RF: %.3f"      % (ozornin\_recall\_score(df.actual\_label.values, df.predicted\_RF.values))  )  print(      "Precision RF: %.3f"      % (ozornin\_precision\_score(df.actual\_label.values, df.predicted\_RF.values))  )  print(      "F1 RF: %.3f" % (ozornin\_f1\_score(df.actual\_label.values, df.predicted\_RF.values))  )  print("")  print("scores with threshold = 0.25")  print(      "Accuracy RF: %.3f"      % (          ozornin\_accuracy\_score(              df.actual\_label.values, (df.model\_RF >= 0.25).astype("int").values          )      )  )  print(      "Recall RF: %.3f"      % (          ozornin\_recall\_score(              df.actual\_label.values, (df.model\_RF >= 0.25).astype("int").values          )      )  )  print(      "Precision RF: %.3f"      % (          ozornin\_precision\_score(              df.actual\_label.values, (df.model\_RF >= 0.25).astype("int").values          )      )  )  print(      "F1 RF: %.3f"      % (          ozornin\_f1\_score(              df.actual\_label.values, (df.model\_RF >= 0.25).astype("int").values          )      )  )  from sklearn.metrics import roc\_curve  fpr\_RF, tpr\_RF, thresholds\_RF = roc\_curve(df.actual\_label.values, df.model\_RF.values)  fpr\_LR, tpr\_LR, thresholds\_LR = roc\_curve(df.actual\_label.values, df.model\_LR.values)  import matplotlib.pyplot as plt  plt.plot(fpr\_RF, tpr\_RF, "r-", label="RF")  plt.plot(fpr\_LR, tpr\_LR, "b-", label="LR")  plt.plot([0, 1], [0, 1], "k-", label="random")  plt.plot([0, 0, 1, 1], [0, 1, 1, 1], "g-", label="perfect")  plt.legend()  plt.xlabel("False Positive Rate")  plt.ylabel("True Positive Rate")  plt.savefig("curve1.png")  from sklearn.metrics import roc\_auc\_score  auc\_RF = roc\_auc\_score(df.actual\_label.values, df.model\_RF.values)  auc\_LR = roc\_auc\_score(df.actual\_label.values, df.model\_LR.values)  print("AUC RF:%.3f" % auc\_RF)  print("AUC LR:%.3f" % auc\_LR)  plt.plot(fpr\_RF, tpr\_RF, "r-", label="RF AUC: %.3f" % auc\_RF)  plt.plot(fpr\_LR, tpr\_LR, "b-", label="LR AUC: %.3f" % auc\_LR)  plt.plot([0, 1], [0, 1], "k-", label="random")  plt.plot([0, 0, 1, 1], [0, 1, 1, 1], "g-", label="perfect")  plt.legend()  plt.xlabel("False Positive Rate")  plt.ylabel("True Positive Rate")  plt.savefig("curve2.png") |
| TP: 5047  FN: 2832  FP: 2360  TN: 5519  Accuracy RF:0.671  Accuracy LR: 0.616  Recall RF: 0.641  Recall LR: 0.543  Precision RF: 0.681  Precision LR: 0.636  F1 RF: 0.660  Ozornin F1 LR: 0.586  Lib F1 LR: 0.586  scores with threshold = 0.5  Accuracy RF: 0.671  Recall RF: 0.641  Precision RF: 0.681  F1 RF: 0.660  scores with threshold = 0.25  Accuracy RF: 0.502  Recall RF: 1.000  Precision RF: 0.501  F1 RF: 0.668  AUC RF:0.738  AUC LR:0.666 |

  
Рис. 7 – ROC(AUC) кожної моделі

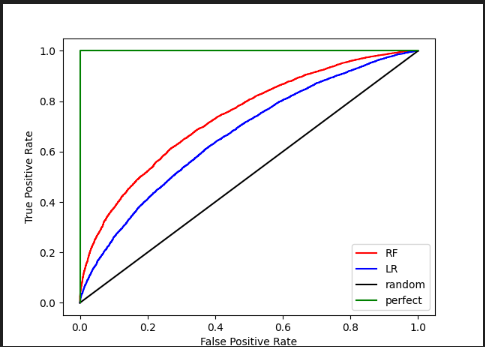


Рис. 8 – ROC кожної моделі

Обидві моделі показують однакові результати, тому вони однаково ефективні

LR\_1\_task\_6.py

|  |
| --- |
| import pandas as pd  from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix  from sklearn.model\_selection import train\_test\_split  from sklearn.naive\_bayes import GaussianNB  from sklearn.svm import SVC  # Завантажимо дані  data = pd.read\_csv(      "data\_multivar\_nb.txt", sep=","  )  # Замість ',' вкажіть коректний роздільник  X = data.iloc[:, :-1].values  # Ознаки  y = data.iloc[:, -1].values  # Цільові значення  # Розділимо на тренувальні та тестові дані  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=3)  # Класифікація за допомогою SVM  svm\_classifier = SVC(kernel="linear", random\_state=42)  svm\_classifier.fit(X\_train, y\_train)  # Прогнозуємо на тестових даних  y\_pred\_svm = svm\_classifier.predict(X\_test)  # Розраховуємо метрики якості  print("SVM Classifier Metrics:")  print("Accuracy:", accuracy\_score(y\_test, y\_pred\_svm))  print("Classification Report:\n", classification\_report(y\_test, y\_pred\_svm))  print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_svm))  # Класифікація за допомогою наївного Байєсівського класифікатора  nb\_classifier = GaussianNB()  nb\_classifier.fit(X\_train, y\_train)  # Прогнозуємо на тестових даних  y\_pred\_nb = nb\_classifier.predict(X\_test)  # Розраховуємо метрики якості  print("\nNaive Bayes Classifier Metrics:")  print("Accuracy:", accuracy\_score(y\_test, y\_pred\_nb))  print("Classification Report:\n", classification\_report(y\_test, y\_pred\_nb))  print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_nb)) |
| SVM Classifier Metrics:  Accuracy: 1.0  Classification Report:  precision recall f1-score support  0 1.00 1.00 1.00 21  1 1.00 1.00 1.00 17  2 1.00 1.00 1.00 14  3 1.00 1.00 1.00 28  accuracy 1.00 80  macro avg 1.00 1.00 1.00 80  weighted avg 1.00 1.00 1.00 80  Confusion Matrix:  [[21 0 0 0]  [ 0 17 0 0]  [ 0 0 14 0]  [ 0 0 0 28]]  Naive Bayes Classifier Metrics:  Accuracy: 1.0  Classification Report:  precision recall f1-score support  0 1.00 1.00 1.00 21  1 1.00 1.00 1.00 17  2 1.00 1.00 1.00 14  3 1.00 1.00 1.00 28  accuracy 1.00 80  macro avg 1.00 1.00 1.00 80  weighted avg 1.00 1.00 1.00 80  Confusion Matrix:  [[21 0 0 0]  [ 0 17 0 0]  [ 0 0 14 0]  [ 0 0 0 28]] |

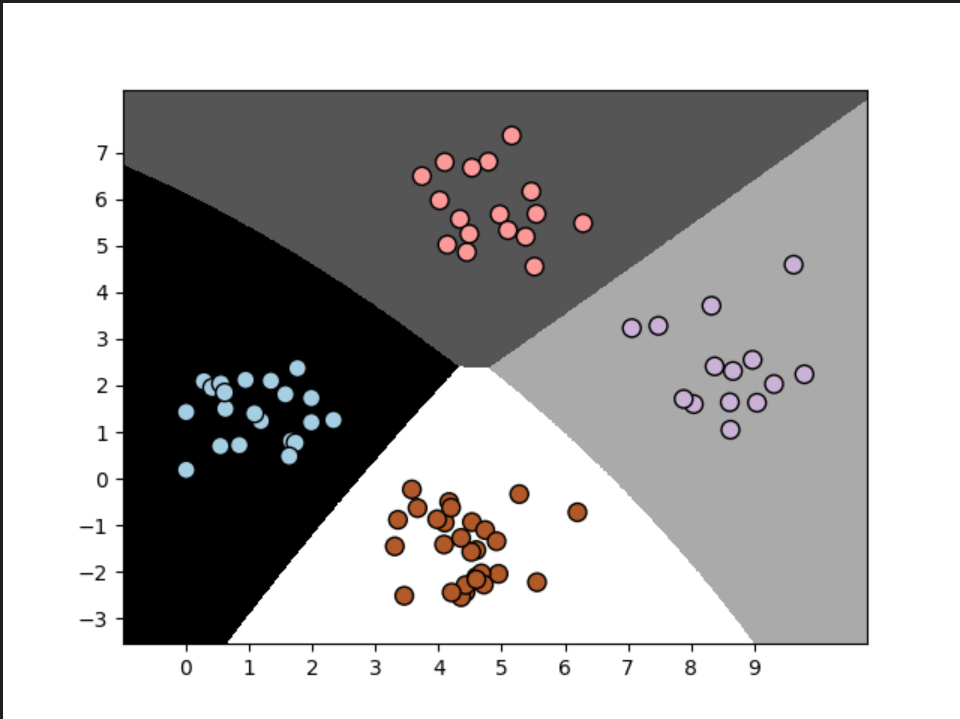


Рис. 9 – Наївний Байєс

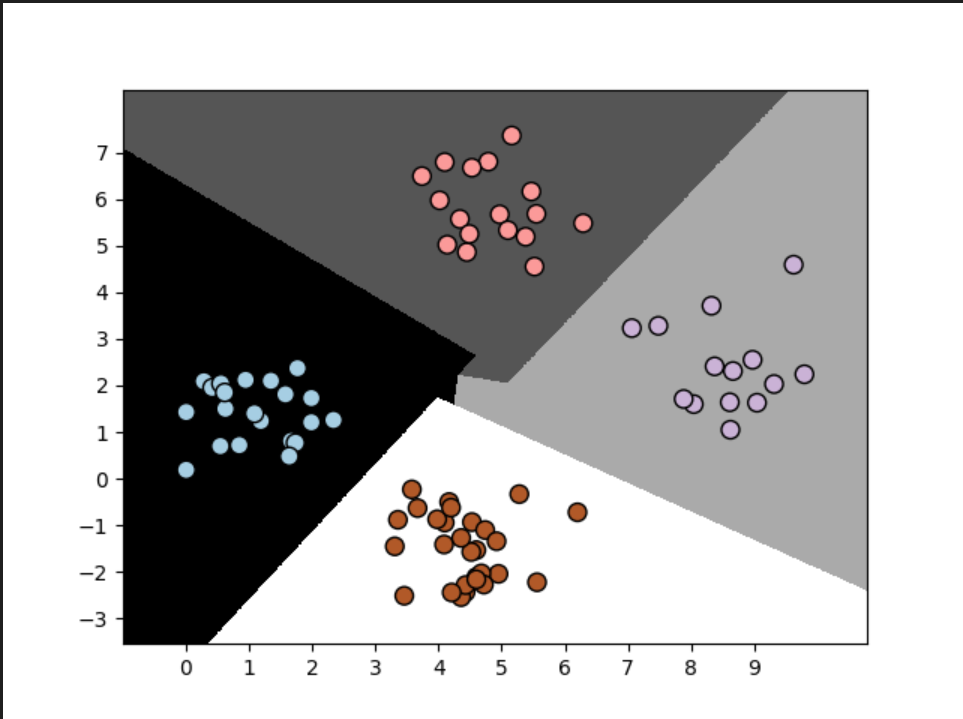


Рис. 10 – SVM

Показники якості класифікацій є ідентичними. Однак і зміни у графіках є.

Змінились області значень для кожної класифікації. Тому в нашому випадку

Модель SVM є кращою.

GitHub = https://github.com/Ozzornin/AIS/tree/master