

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ  
Кафедра математичних методів системного аналізу

## **ЗВІТ**

про виконання лабораторної роботи № 2  
з дисципліни «Інтелектуальний аналіз даних»

Виконала:

студентка 3 курсу  
групи КА-91  
Оніщенко Д.С.

Перевірила:

Недашківська Н. І.

Київ – 2021

## **Умова задачі**

### **1 Хід виконання роботи:**

1. Побудувати моделі класифікації або регресії згідно з варіантом.
2. Виконати прогнози на основі побудованих моделей.
3. Для кожної з моделей оцінити, чи має місце перенавчання.
4. Розрахувати додаткові результати моделей, наприклад, апостеріорні імовірності, опорні вектори або інші (згідно з варіантом).
5. В задачах класифікації побудувати границі рішень графічно дляожної з моделей.
6. В задачах класифікації розрахувати значення наступних критеріїв якості, дляожної моделі окремо на навчальній та перевірочній множинах:
  - матрицю неточностей (confusion matrix),
  - точність (precision),
  - повноту (recall),
  - міру F1 (F1 score),
  - побудувати криву точності-повноти (precision-recall (PR) curve), ROC-криву, показник AUC.
7. В задачах регресії розрахувати критерії якості дляожної моделі окремо на навчальній та перевірочній множинах:
  - коефіцієнт детермінації  $R^2$ ,
  - помилки RMSE, MAE та MAPE.
8. Виконати решітчастий пошук (grid search) для підбору гіперпараметрів моделей.
9. Зробити висновки про якість роботи моделей на досліджених даних. На основі критеріїв якості вибрати найкращу модель.
10. Навчити моделі на підмножинах навчальних даних. Оцінити, наскільки розмір навчальної множини впливає на якість моделі.
11. Кожен варіант містить два набори даних. Дослідити обидва набори за наведеними вище етапами.

## Варіант 6:

6. Побудувати моделі логістичної регресії:

- Просту логістичну регресію, використовуючи `sklearn.linear_model.LogisticRegression`.
- Поліноміальну логістичну регресію (multinomial logistic regression), встановивши гіперпараметри `multi_class = "multinomial"` та `solver = "lbfgs"`.
- Для наведених моделей побудувати варіанти з і без регуляризації.
- Для кожної моделі розрахувати апостеріорні імовірності для тестового прикладу, використовуючи метод `predict_proba`.

Початкові дані:

- (a) `sklearn.datasets.make_moons`  
(б) `sklearn.datasets.load_digits`

## Хід роботи

1) Графічно представити дані

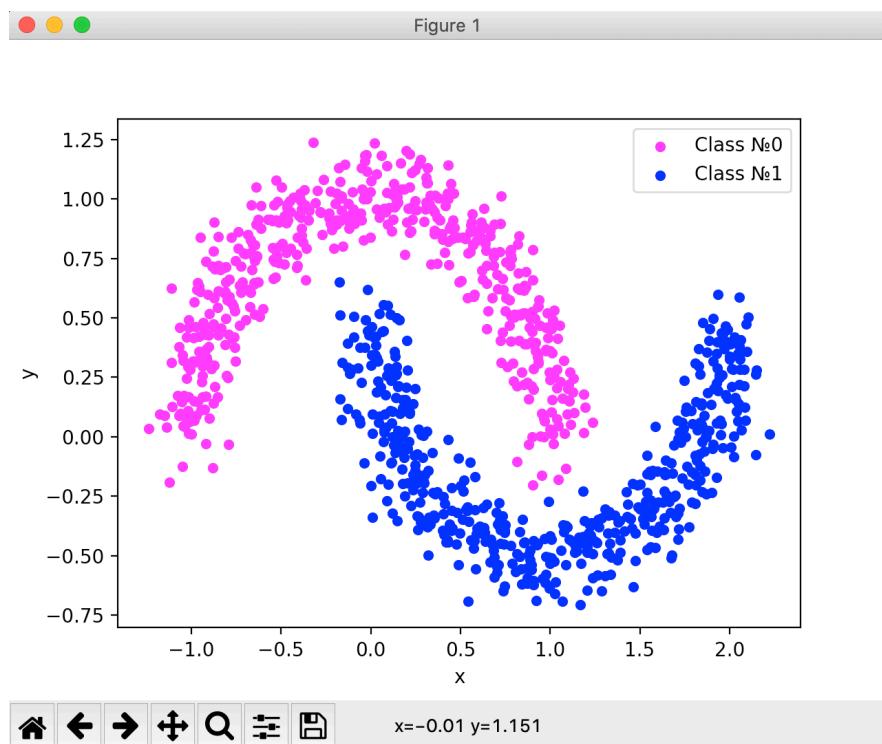
```
from sklearn.datasets import load_digits
from sklearn.datasets import make_moons
from matplotlib import pyplot as plt
from pandas import DataFrame
X_m, y_m = make_moons(n_samples=1000, noise=0.1)
digits_ = load_digits()
def show_moons(X, y):
    df = DataFrame(dict(x=X[:, 0], y=X[:, 1], label=y))
    colors = {0: 'magenta', 1: 'blue'}
    fig, ax = plt.subplots()
    grouped = df.groupby('label')
    for key, group in grouped:
        group.plot(ax=ax, kind='scatter', x='x', y='y', label='Class №' +
str(key), color=colors[key])
    plt.show()
show_moons(X_m, y_m)
def show_digits(digits):
    fig, axes = plt.subplots(8, 8, figsize=(8, 8), subplot_kw={'xticks': [],
'yticks': []},
                           gridspec_kw=dict(hspace=0.1, wspace=0.1))
    for i, ax in enumerate(axes.flat):
```

```

        ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
color=magenta)
ax.text(0.05, 0.05, str(digits.target[i]), transform=ax.transAxes,
plt.show()
show_digits(digits_)

```

Перший датасет:



Другий датасет:

2) розбити на навчальний та перевірочний набори

```

y_digits = digits_.target
X_digits = digits_.data
X_moons_train, X_moons_test, y_moons_train, y_moons_test =
train_test_split(X_moons, y_moons, test_size=0.1)
X_digits_train, X_digits_test, y_digits_train, y_digits_test =
train_test_split(X_digits, y_digits, test_size=0.2)

```

3) побудувати моделі класифікації (логістичної регресії)

```

def create_models(X_train, y_train):
    simple_lr = LogisticRegression(penalty='none').fit(X_moons_train, y_moons_train) #
without regularization
    simple_lr_regularized = LogisticRegression().fit(X_moons_train, y_moons_train) #
with regularization
    multinomial_lr = LogisticRegression(penalty='none', multi_class='multinomial',
solver='lbfgs', max_iter=1000).fit(

```



Figure 1



```

        X_train, y_train)                      # without regularization
    multinomial_lr_regularized = LogisticRegression(multi_class='multinomial',
solver='lbfgs', max_iter=1500).fit(X_moons_train, y_moons_train)      # with regularization
    return simple_lr, multinomial_lr, simple_lr_regularized, multinomial_lr_regularized
moons_simple, moons_multinomial, moons_simple_reg, moons_multinomial_reg =
create_models(X_moons_train, y_moons_train)
digits_simple, digits_multinomial, digits_simple_reg, digits_multinomial_reg =
create_models(X_digits_train, y_digits_train)

```

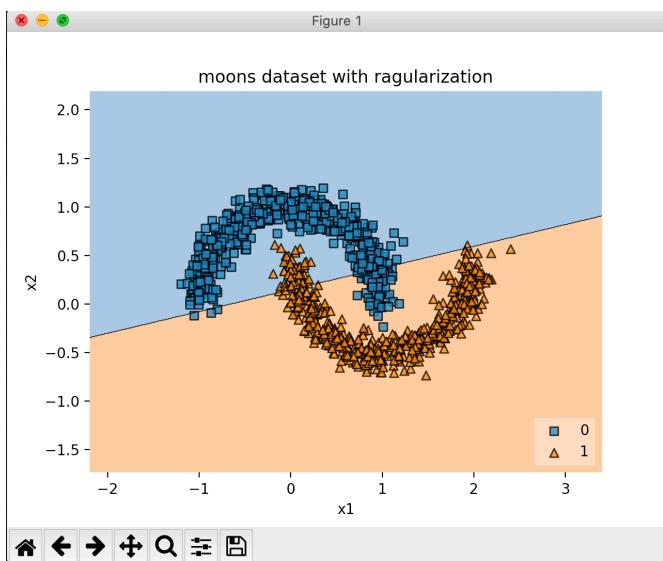
4) представити моделі графічно (представлено модель лише для первого датасету, оскільки розмірність другого датасету занадто велика для коректоного відображення моделей)

5) побудувати границі рішень графічно

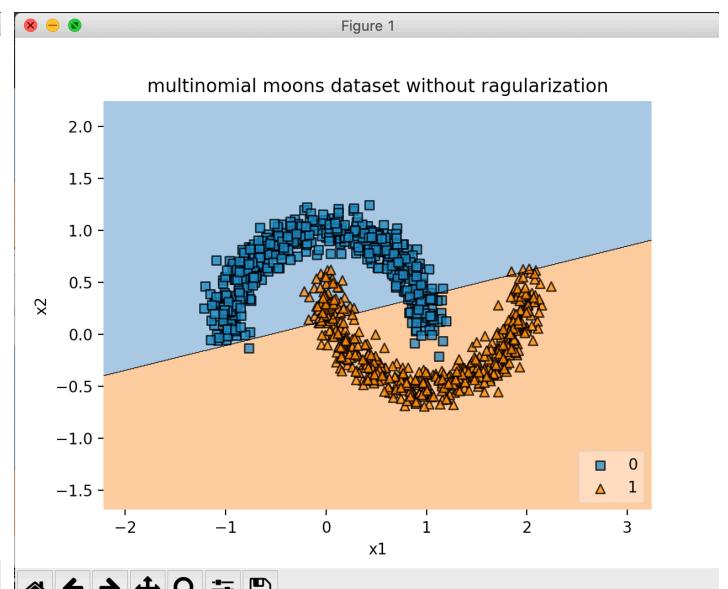
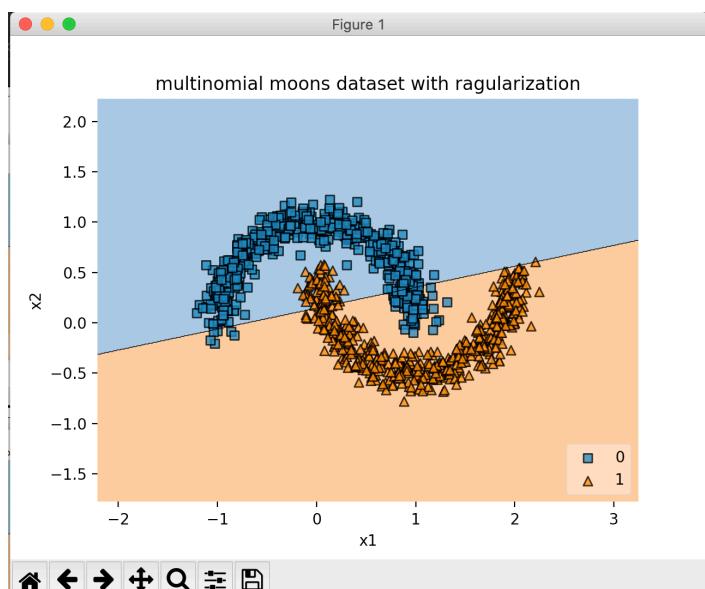
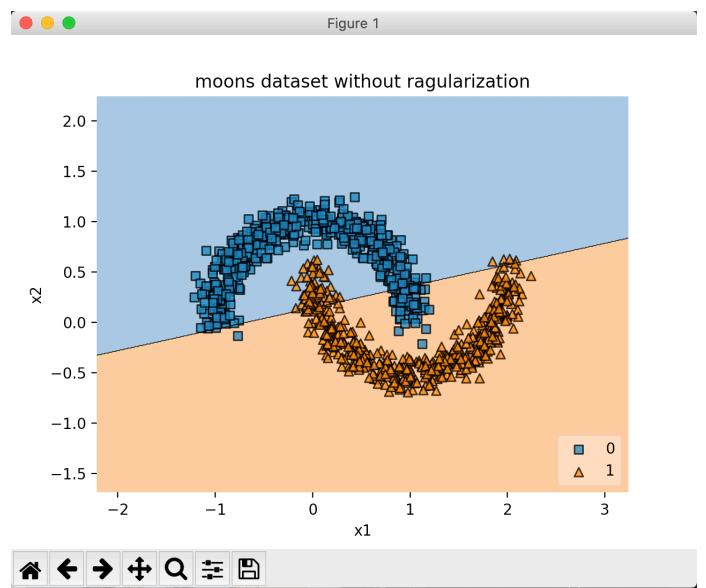
```

plot_decision_regions(X_moons, y_moons, clf=moons_simple, legend=4)
plt.xlabel('x1')
plt.ylabel('x2')

```



```
plt.title('moons dataset')
plt.show()
```



6) виконати прогнози на основі моделей

+ визначити апостеріорні ймовірності для тестового прикладу

+ 7) розрахувати критерій якості окремо для кожної моделі на навчальні та тестовій підмножинах

матрицю неточностей (confusion matrix)

```

def create_matrix(curr_model, x, y_test, y_predict):
    cm = mt.confusion_matrix(y_test, y_predict)
    score = curr_model.score(x, y_test)
    plt.figure(figsize=(9, 9))
    sns.heatmap(cm, annot=True, fmt=".3f", square=True)
    plt.ylabel("Actual values")
    plt.xlabel("Predicted values")
    plt.title(f"Accuracy : {score}\n\nconfusion matrix for {curr_model}", size=15)
    plt.show()

```

точність (precision),

- повноту (recall),
- міру F1 (F1 score),
- побудувати криву точності-повноти (precision-recall (PR) curve), ROC-криву, показник AUC.

## -----METRICS FOR MOONS-----

\* Metrics for train data \*

confusion matrix

```

#-----METRICS FOR MOONS-----
for model in models_moon[:2]:
    y_predict = model.predict(X_moons_train)
    create_matrix(model, X_moons_train, y_moons_train, y_predict)
-----quality criteria,-----
def estimate(curr_model, x, y_true, num_classes):
    y_predict = curr_model.predict(x)
    create_matrix(curr_model, x, y_true, y_predict)
    print({curr_model})
    print('Precision = ', mt.precision_score(y_true, y_predict, average='micro'))
    print('Recall = ', mt.recall_score(y_true, y_predict, average='micro'))
    print('F1-score = ', mt.f1_score(y_true, y_predict, average='micro'))
    if num_classes == 2:
        pr_curve = PrecisionRecallDisplay.from_predictions(y_true, y_predict)
        pr_curve.ax_.set_title('Precision-Recall curve')
        plt.show()

        roc_curve_moons = RocCurveDisplay.from_predictions(y_true, y_predict)
        roc_curve_moons.ax_.set_title('ROC curve')
        plt.show()
    y_probs = curr_model.predict_proba(x)

    if num_classes > 2:
        skplt.plot_precision_recall(y_true, y_probs)
        plt.show()

        skplt.plot_roc(y_true, y_probs)
        plt.show()

```

## -----METRICS FOR MOONS-----

```

for model in models_moon[:1]:
    estimate(model, X_moons_train, y_moons_train, 2)

```

```

print('-----METRICS FOR DIGITS-----')

```

```

for model in models_digits[1:]:
    print('\n* Metrics for ', model, ' on train data *\n')
    estimate(model, X_digits_train, y_digits_train, 10)

```

## Moons probability

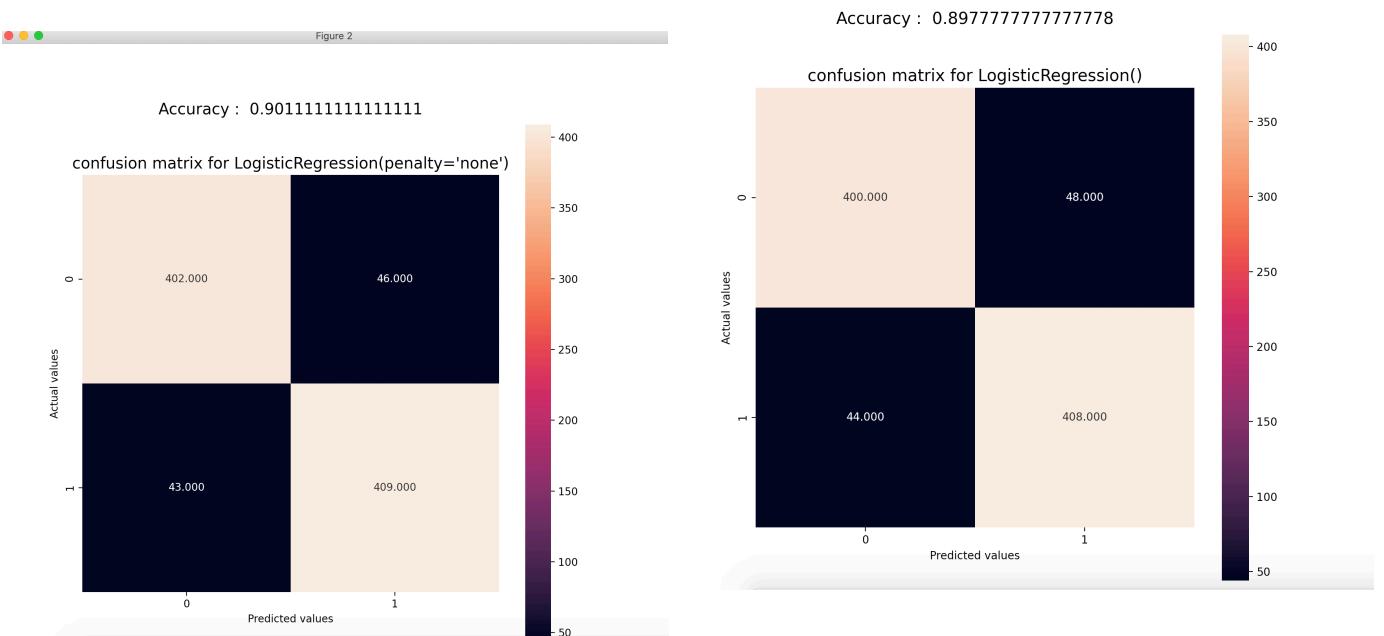
LogisticRegression(penalty='none') [[0.74202085 0.25797915]]

LogisticRegression() [[0.70414148 0.29585852]]

LogisticRegression(max\_iter=1000, multi\_class='multinomial', penalty='none')  
[[0.74202041 0.25797959]]

LogisticRegression(max\_iter=1500, multi\_class='multinomial') [[0.72046067  
0.27953933]]

Figure 1

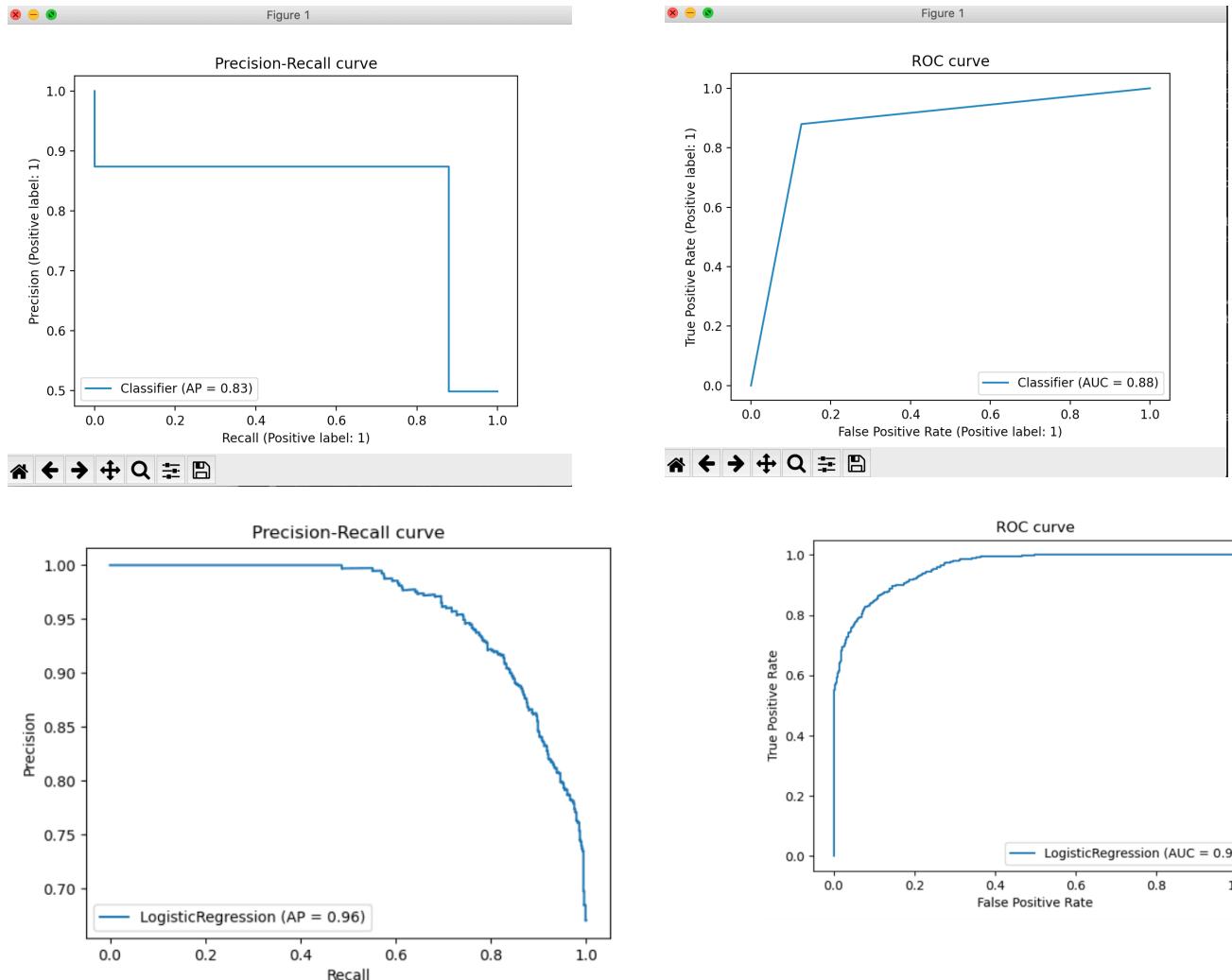


```

-----METRICS FOR MOONS-----
for model in models_moon[:1]:
    estimate(model, X_moons_train, y_moons_train, 2)

-----METRICS FOR DIGITS-----
for model in models_digits[1:]:
    print('\n* Metrics for ', model, ' on train data *\n')
    estimate(model, X_digits_train, y_digits_train, 10)

```



```
{LogisticRegression(penalty='none')}
```

```
Precision = 0.8788888888888889
```

```
Recall = 0.8788888888888889
```

```
F1-score = 0.8788888888888889
```

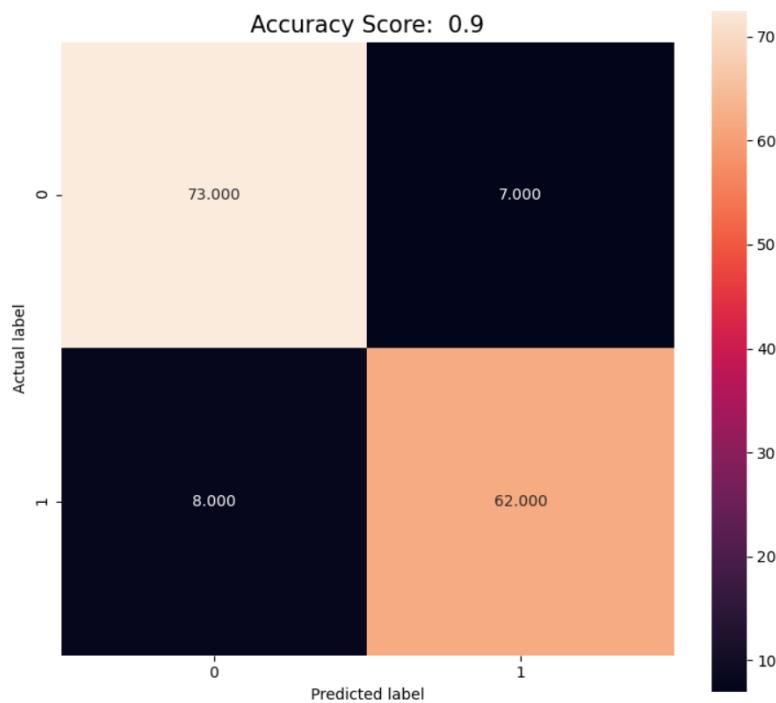
```
{LogisticRegression()}
```

```
Precision = 0.8755555555555555
```

```
Recall = 0.8755555555555555
```

```
F1-score = 0.8755555555555555
```

\* Metrics for LogisticRegression(penalty='none') on test data \*

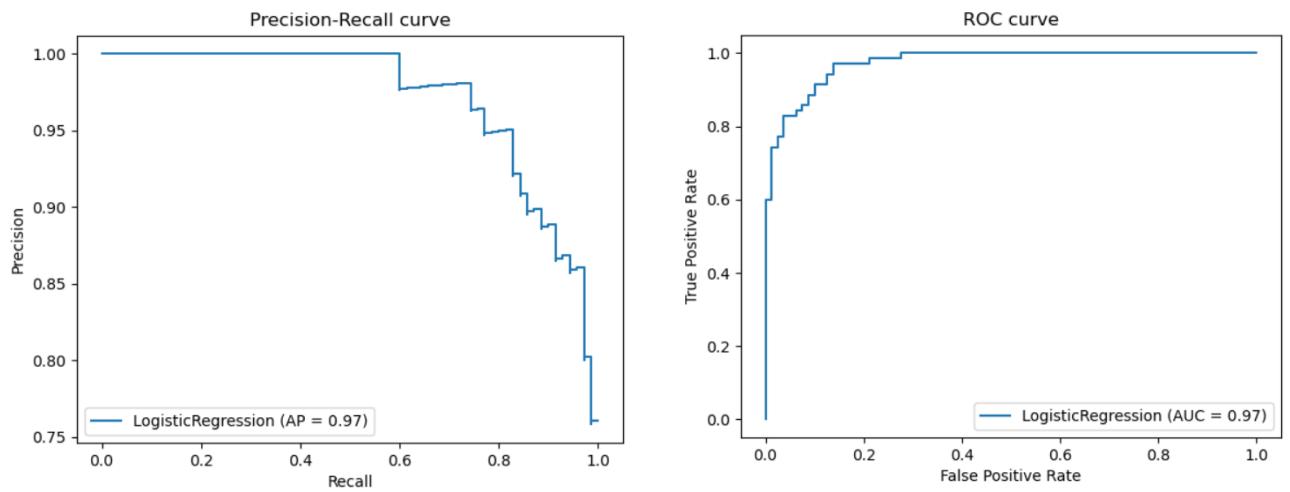


{LogisticRegression(penalty='none')}

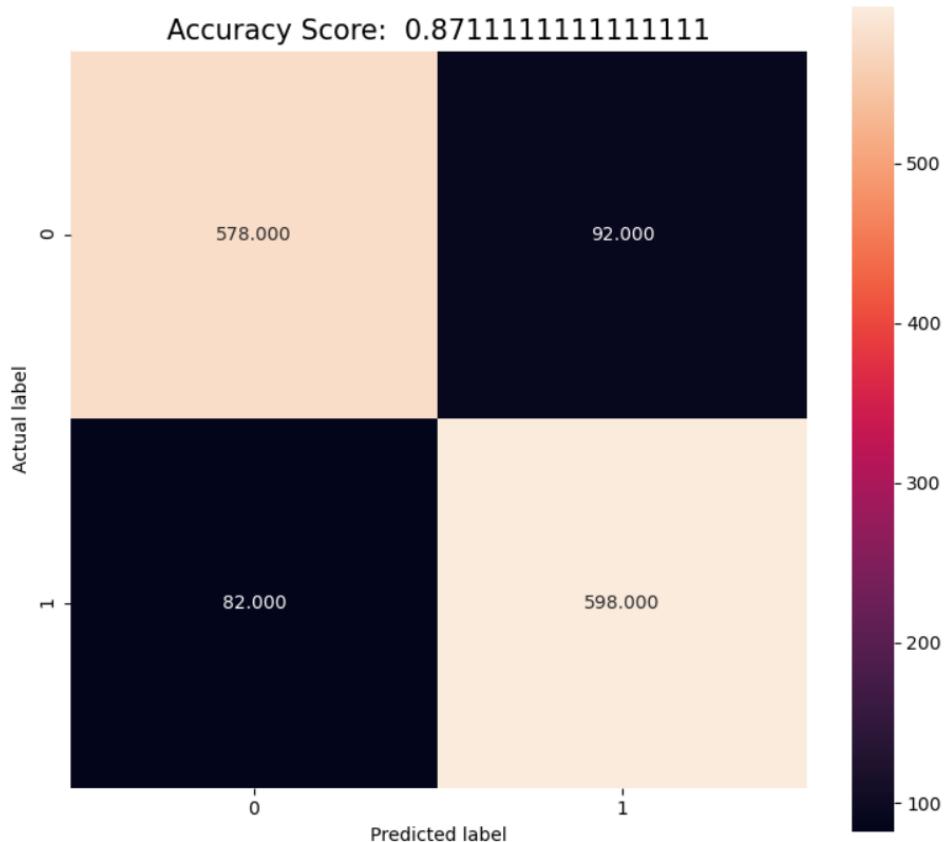
Precision-score: 0.9

Recall-score: 0.9

F1-score: 0.9



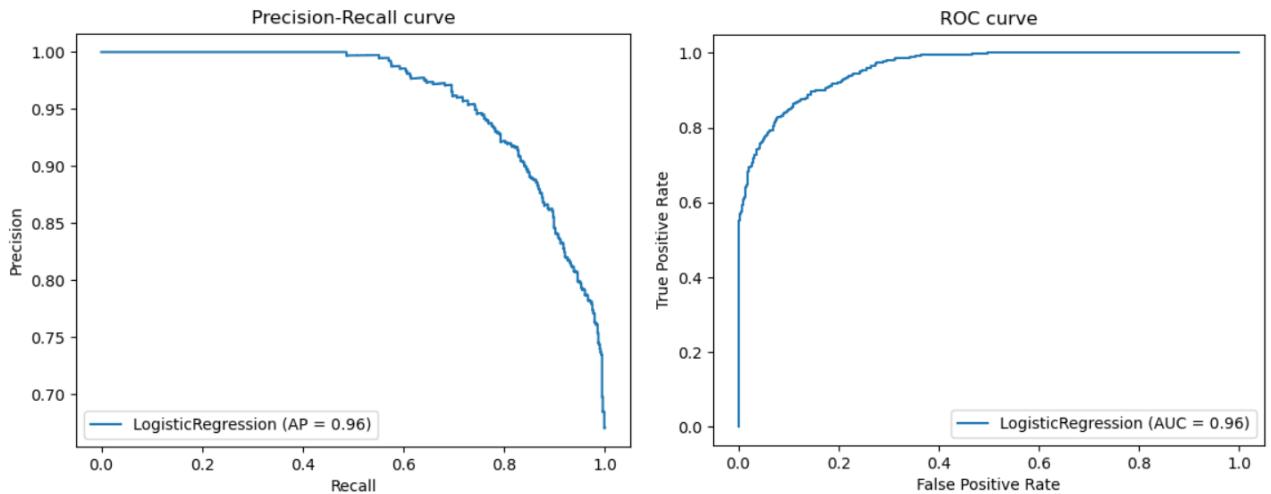
\* Metrics for LogisticRegression() on train data \*



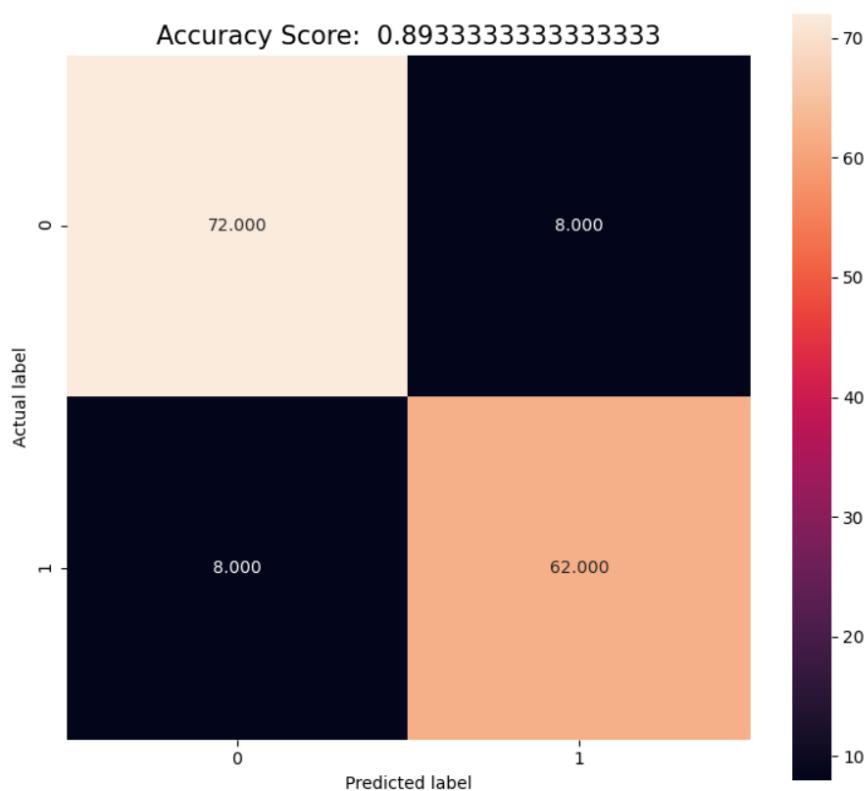
Precision-score: 0.8711111111111111

Recall-score: 0.8711111111111111

F1-score: 0.8711111111111111



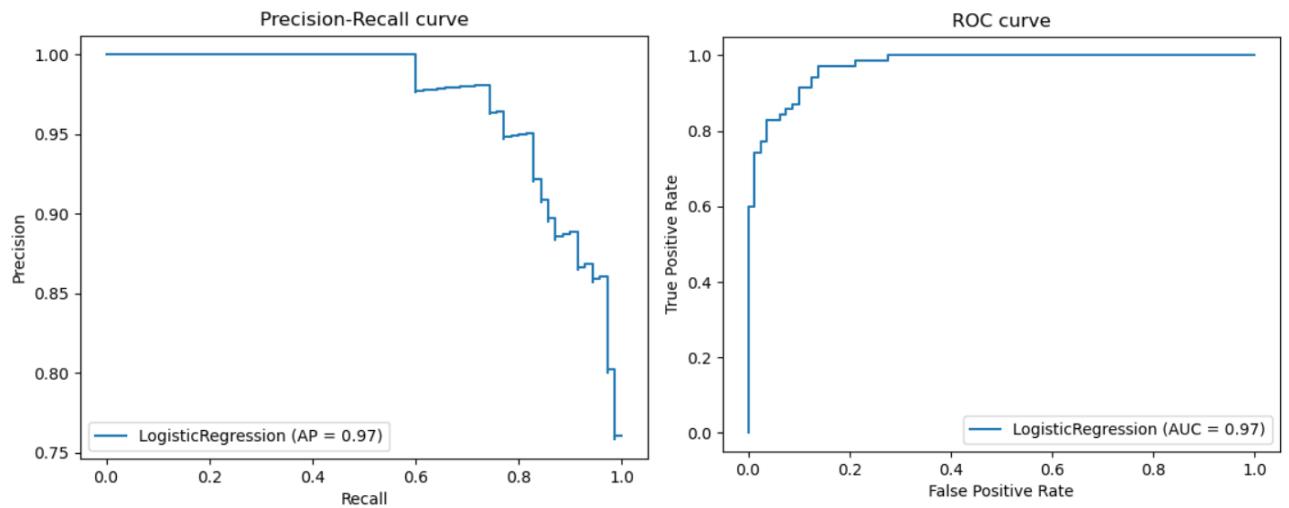
\* Metrics for LogisticRegression() on test data \*



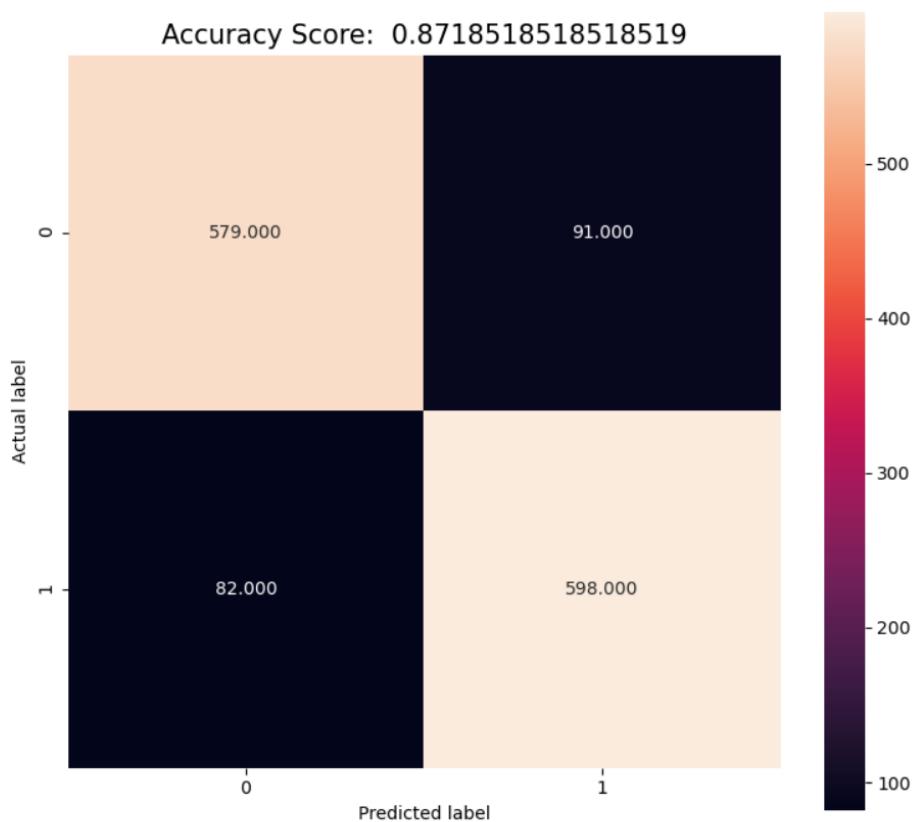
Precision-score: 0.8933333333333333

Recall-score: 0.8933333333333333

F1-score: 0.8933333333333333



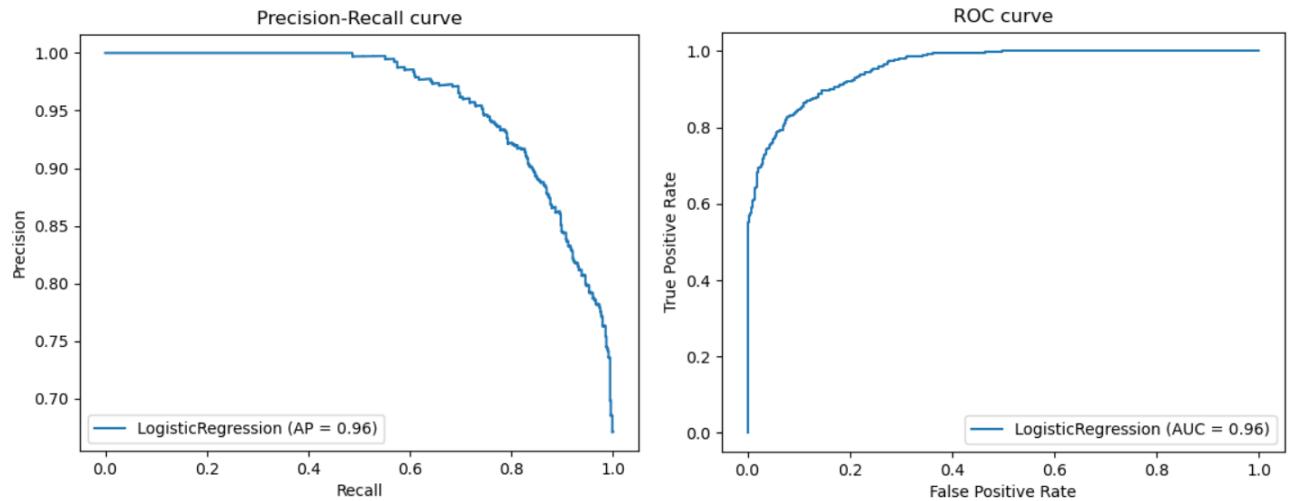
\* Metrics for LogisticRegression(max\_iter=1000, multi\_class='multinomial', penalty='none') on train data \*



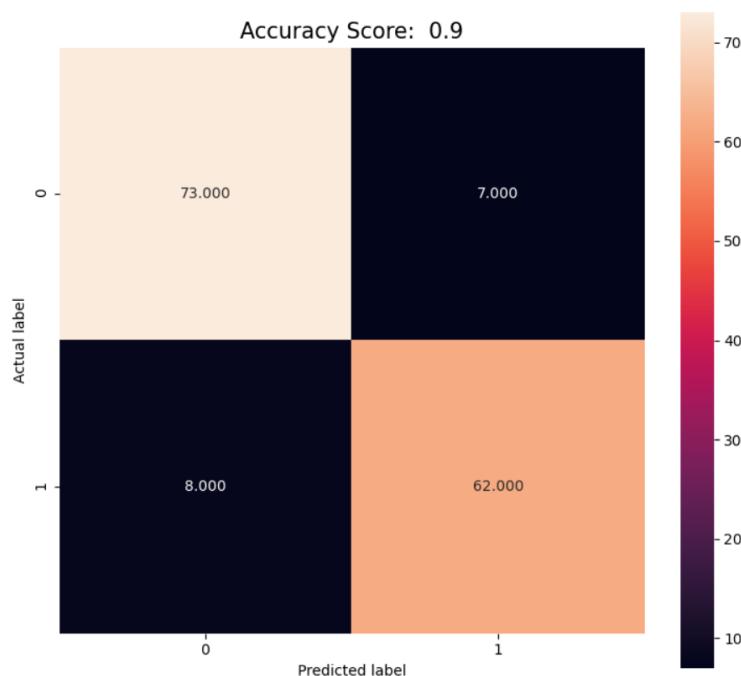
Precision-score: 0.8718518518518519

Recall-score: 0.8718518518518519

F1-score: 0.8718518518518519



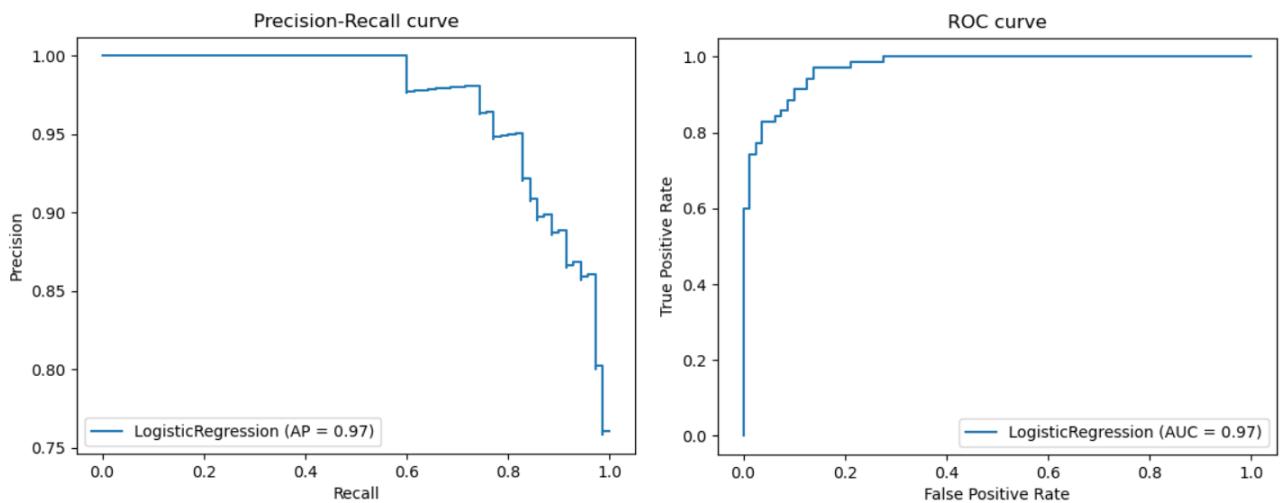
\* Metrics for LogisticRegression(max\_iter=1000, multi\_class='multinomial', penalty='none') on test data \*



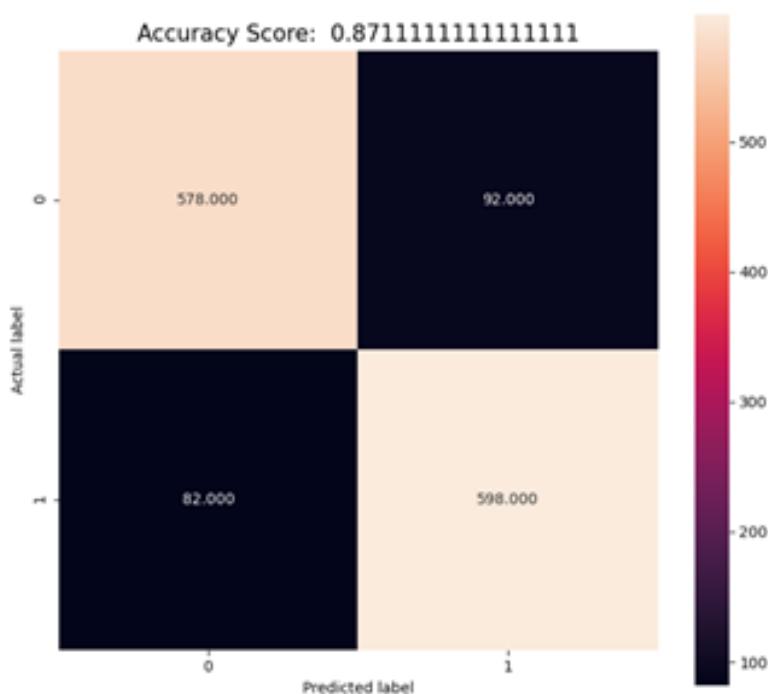
Precision-score: 0.9

Recall-score: 0.9

F1-score: 0.9



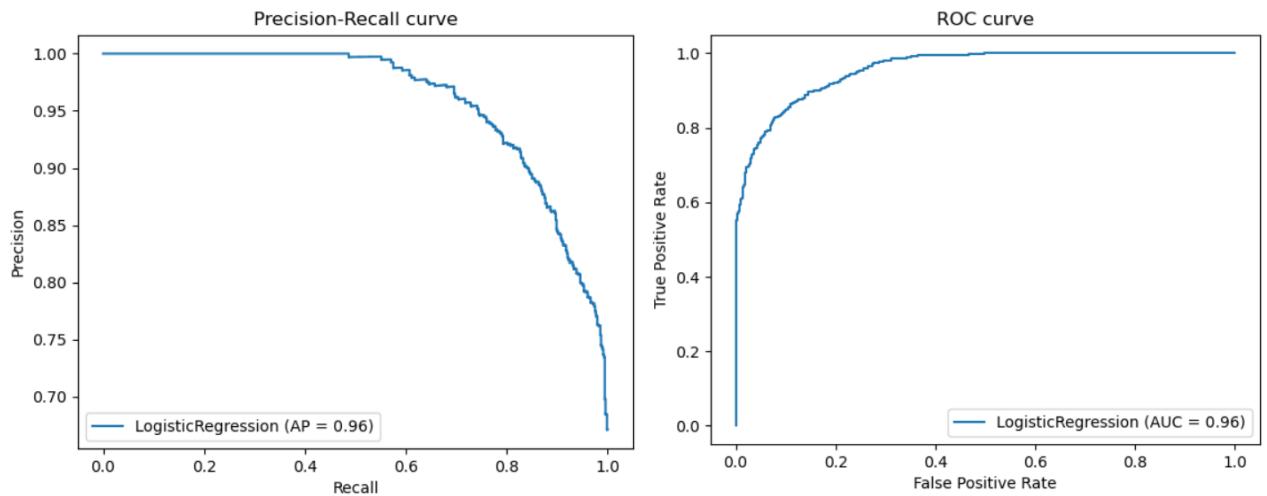
\* Metrics for LogisticRegression(max\_iter=1000, multi\_class='multinomial') on train data \*



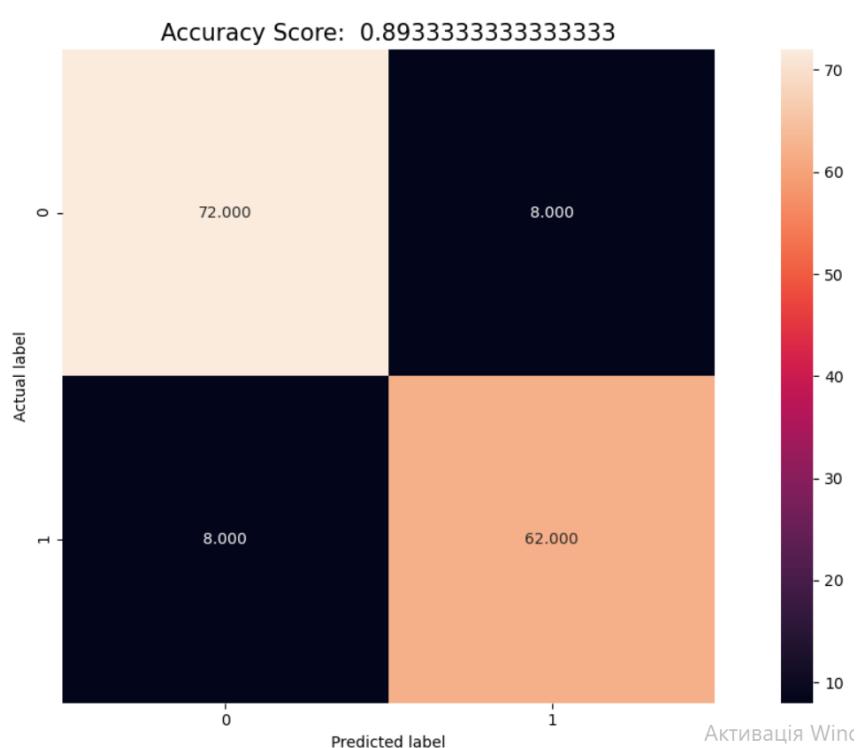
Precision-score: 0.8711111111111111

Recall-score: 0.8711111111111111

F1-score: 0.8711111111111111



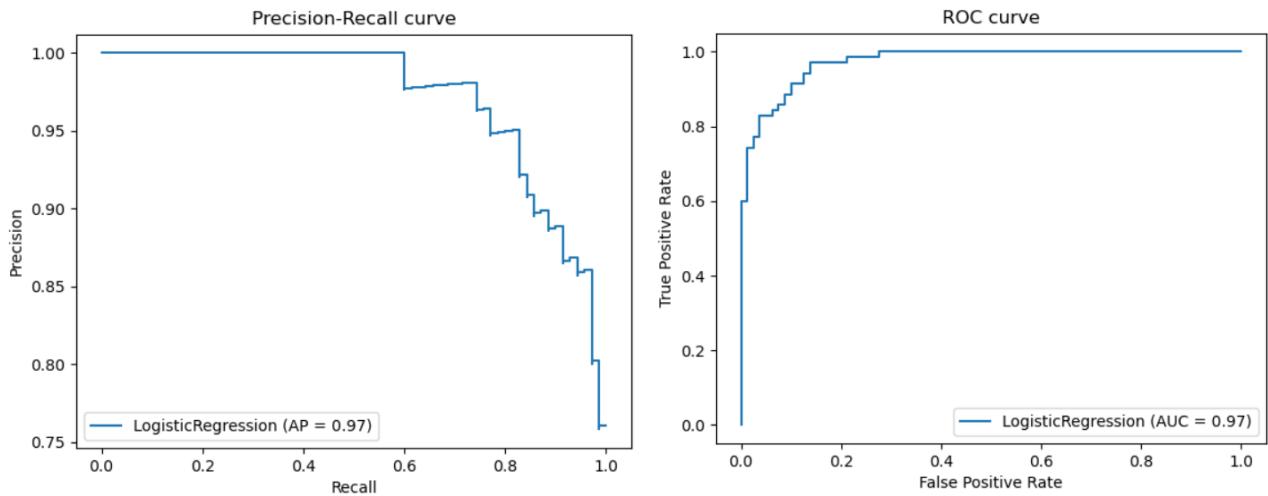
\* Metrics for LogisticRegression(max\_iter=1000, multi\_class='multinomial') on test data \*



Precision-score: 0.8933333333333333

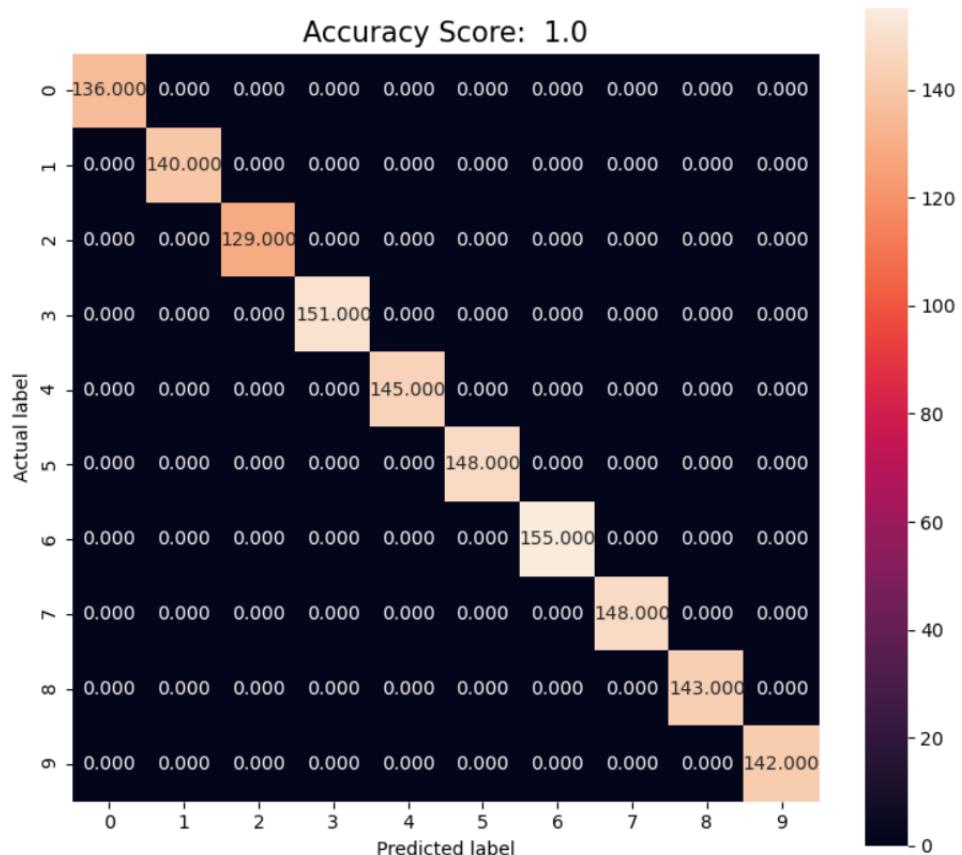
Recall-score: 0.8933333333333333

F1-score: 0.8933333333333333



## METRICS FOR DIGITS

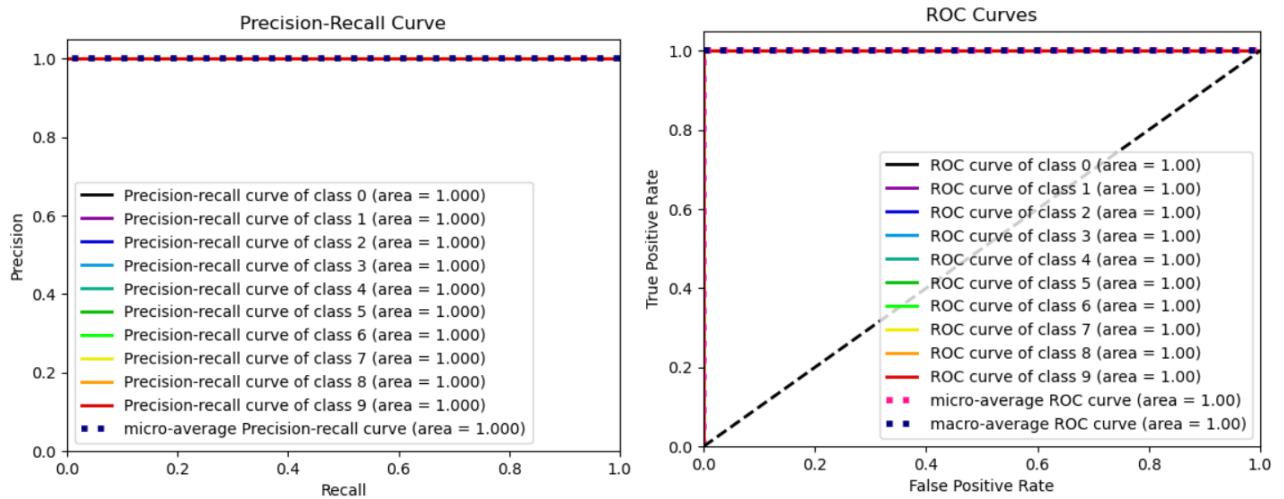
\* Metrics for LogisticRegression(penalty='none') on train data \*



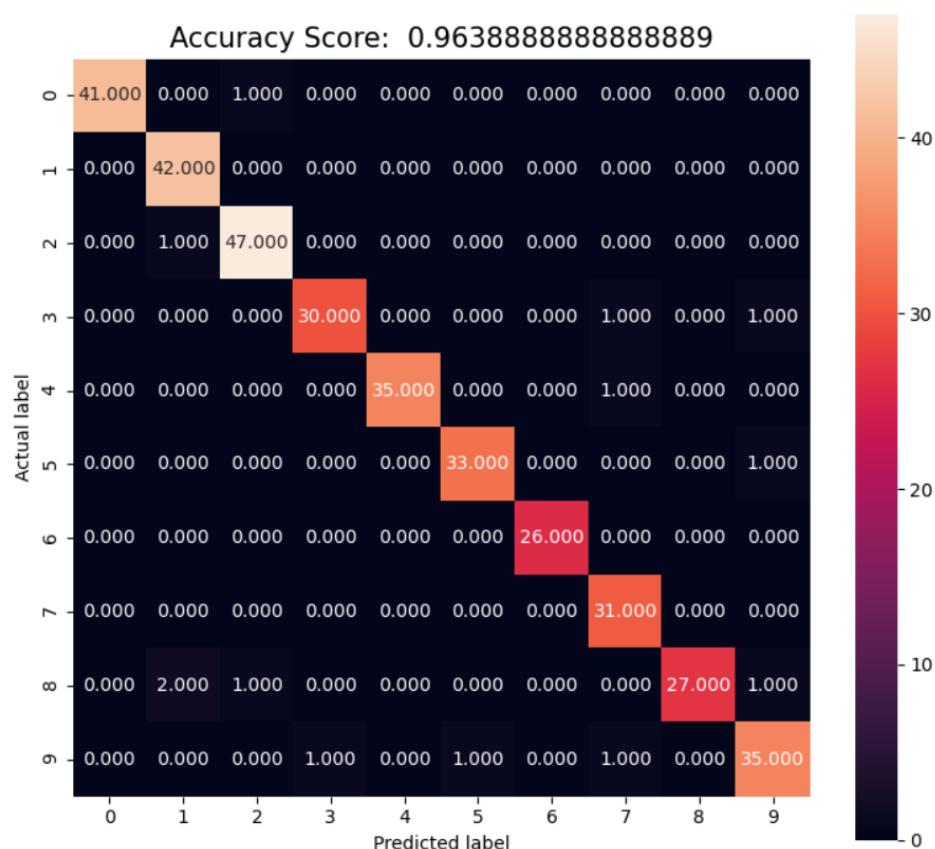
Precision-score: 1.0

Recall-score: 1.0

F1-score: 1.0



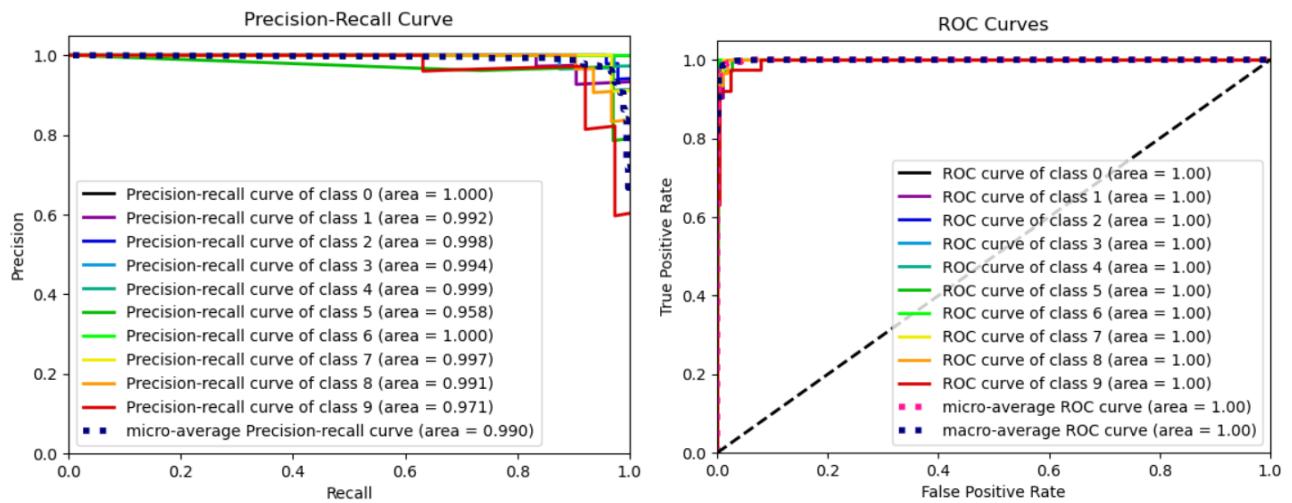
\* Metrics for LogisticRegression(penalty='none') on test data \*



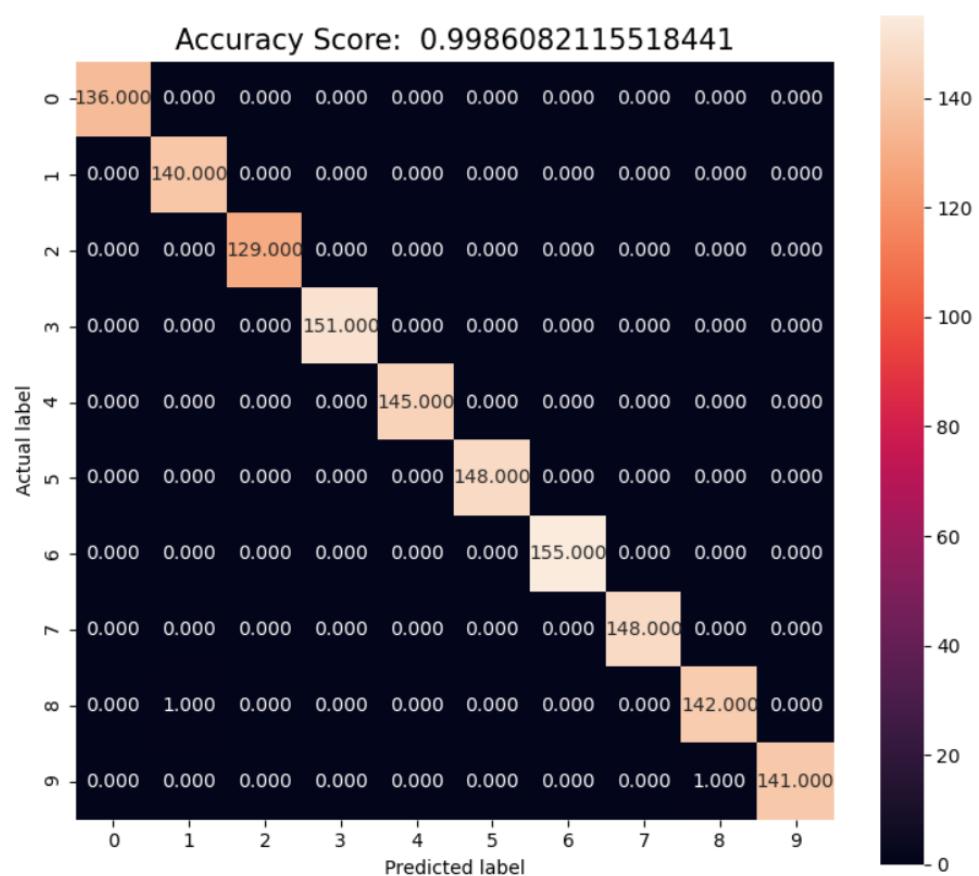
Precision-score: 0.9638888888888889

Recall-score: 0.9638888888888889

F1-score: 0.9638888888888889



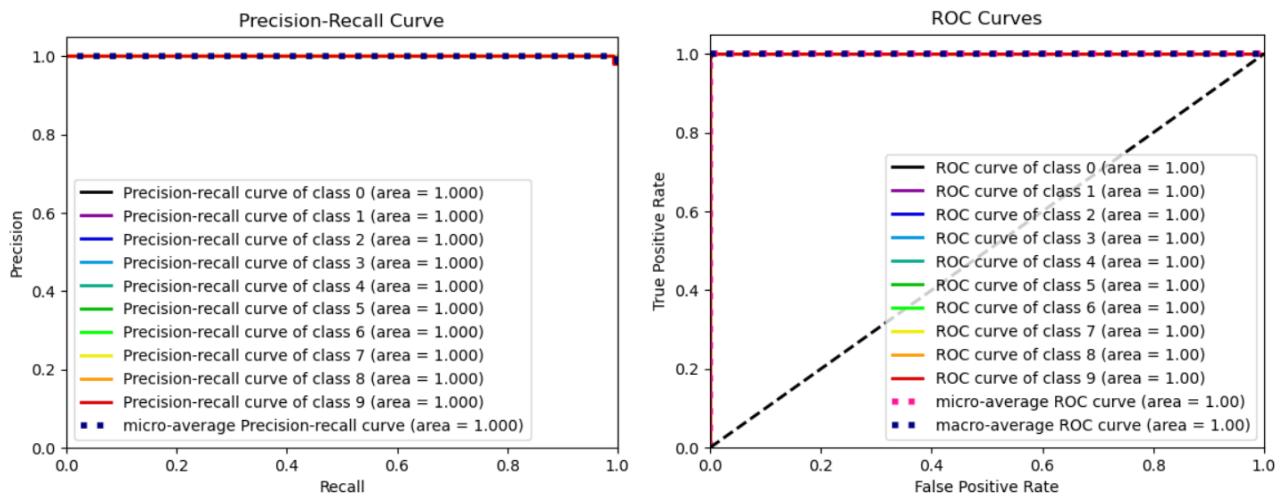
\* Metrics for LogisticRegression() on train data \*



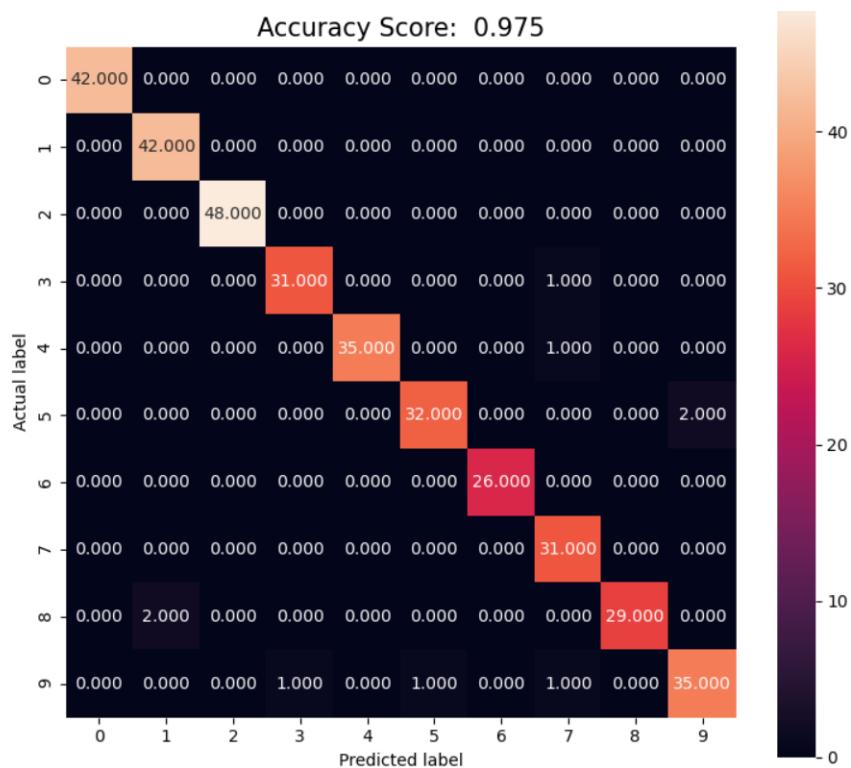
Precision-score: 0.9986082115518441

Recall-score: 0.9986082115518441

F1-score: 0.9986082115518441



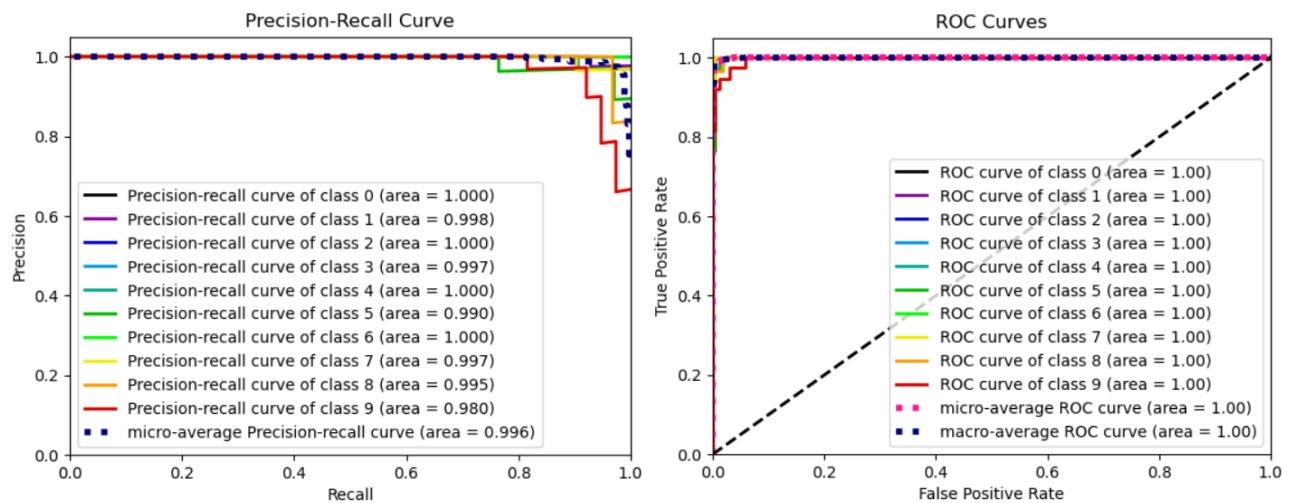
\* Metrics for LogisticRegression() on test data \*



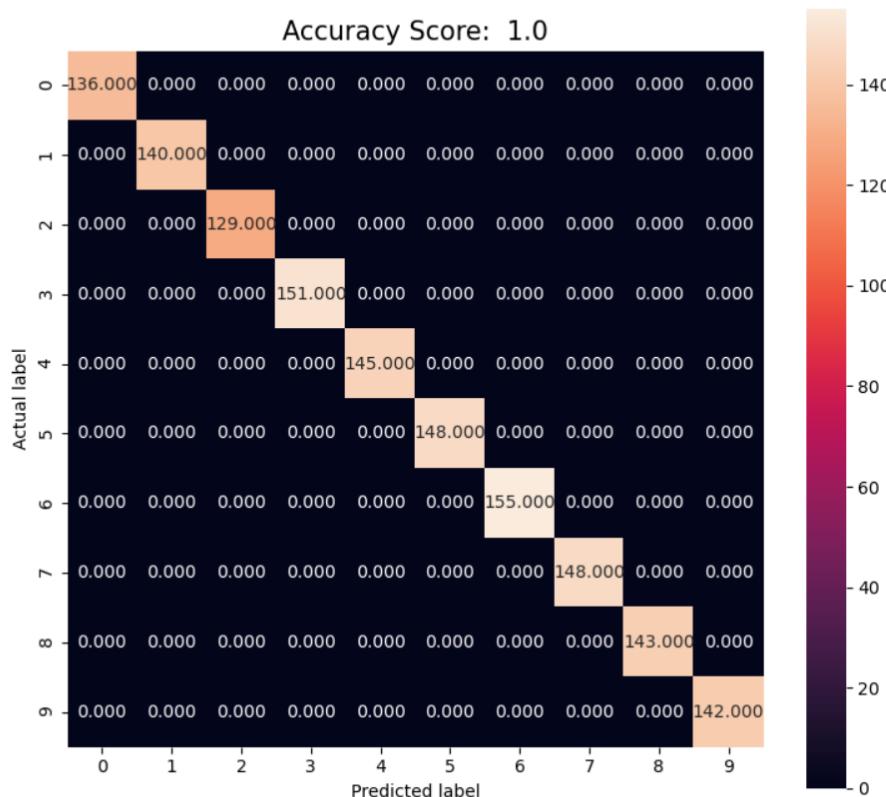
Precision-score: 0.975

Recall-score: 0.975

F1-score: 0.975



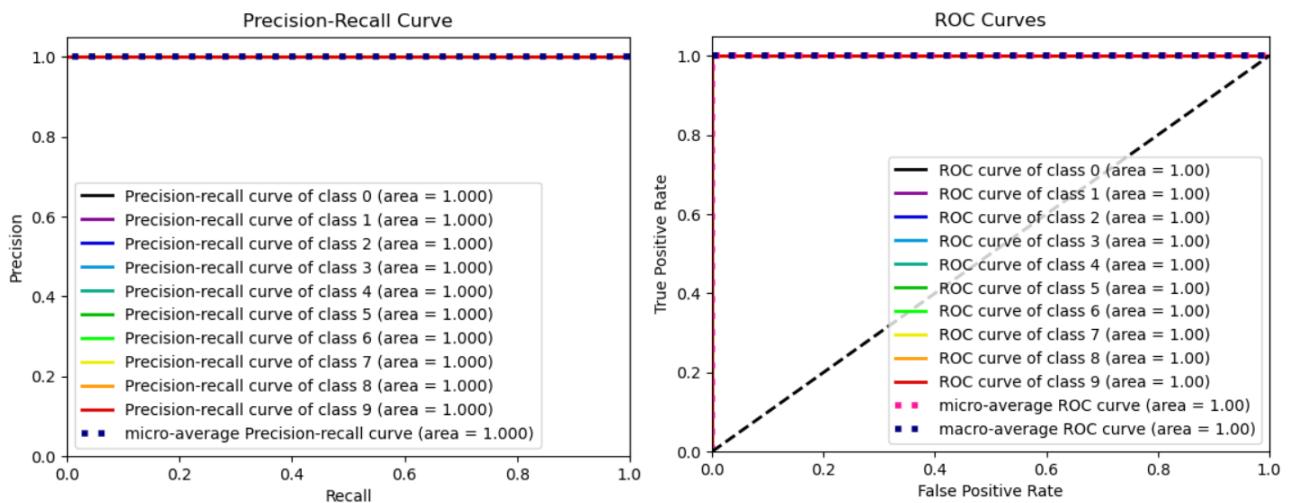
\* Metrics for LogisticRegression(max\_iter=1000, multi\_class='multinomial', penalty='none') on train data \*



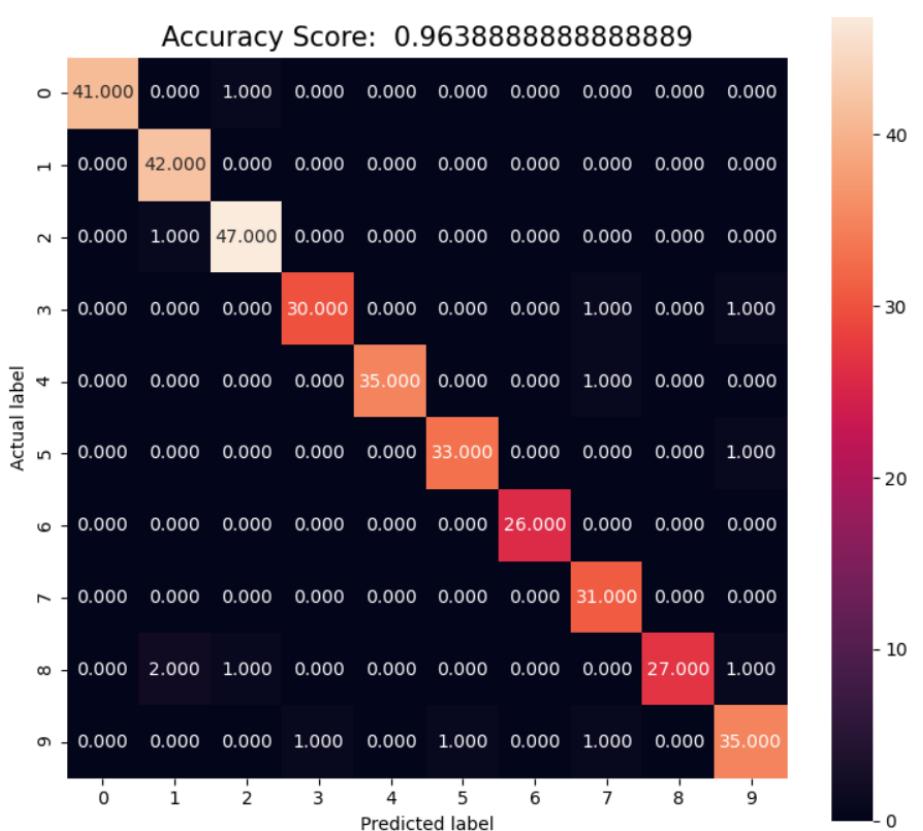
Precision-score: 1.0

Recall-score: 1.0

F1-score: 1.0



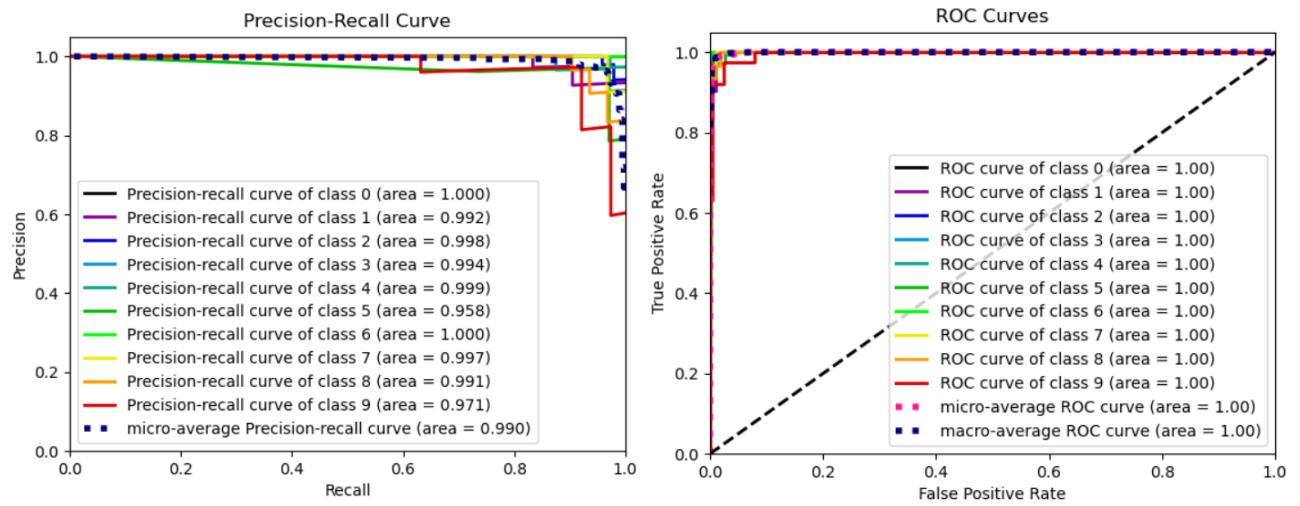
\* Metrics for LogisticRegression(max\_iter=1000, multi\_class='multinomial', penalty='none') on test data \*



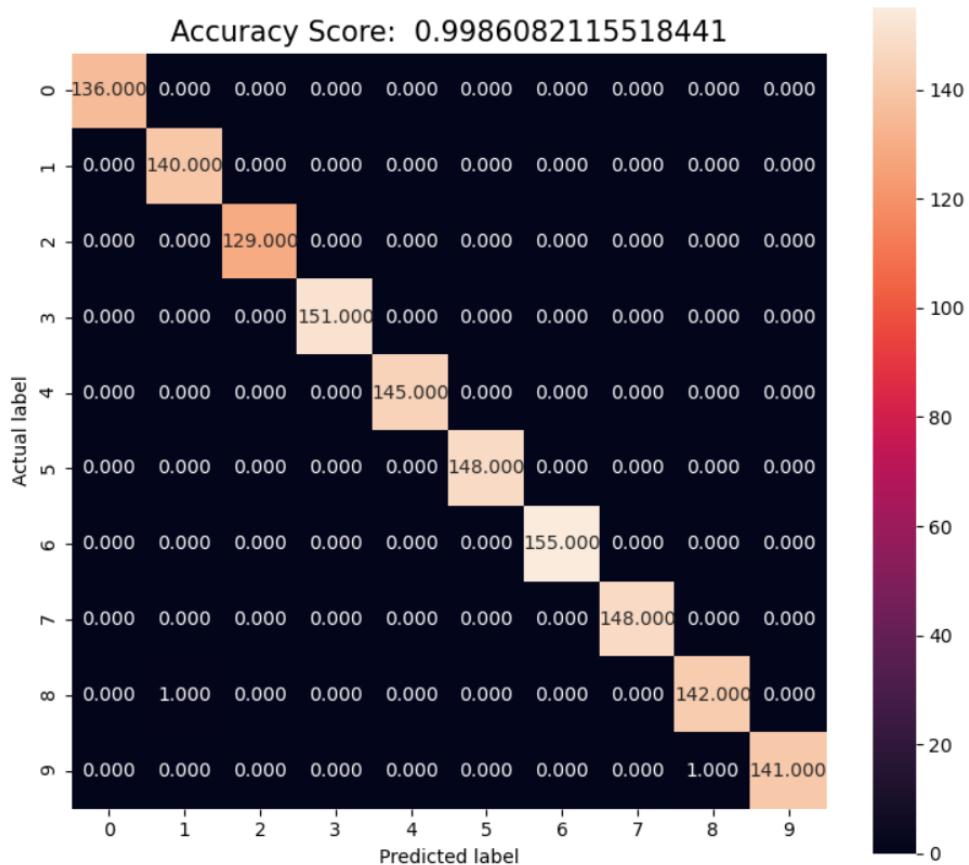
Precision-score: 0.9638888888888889

Recall-score: 0.9638888888888889

F1-score: 0.9638888888888889



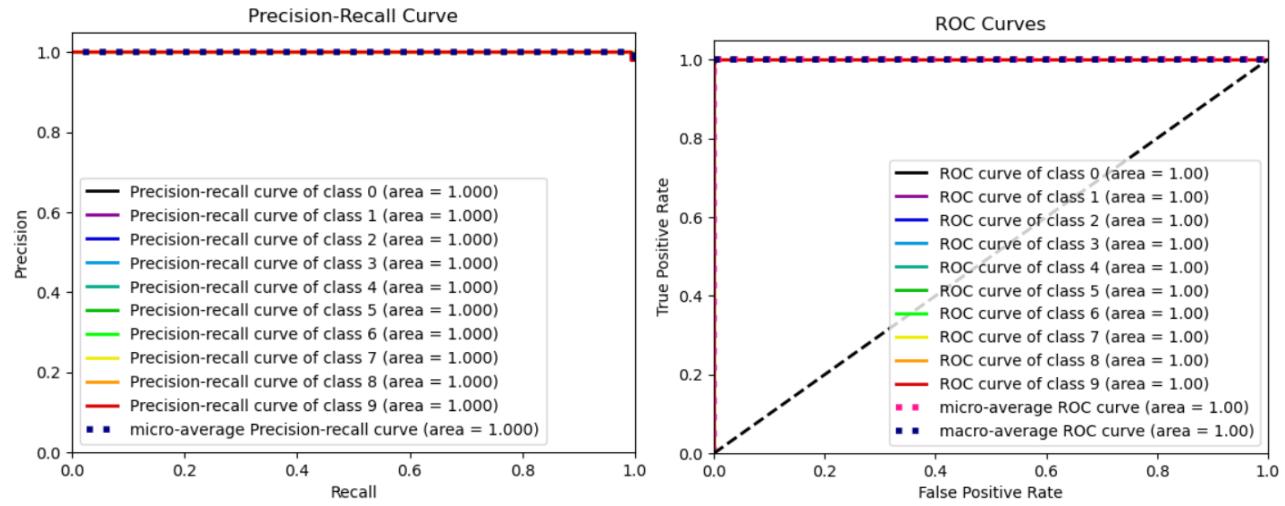
\* Metrics for LogisticRegression(max\_iter=1000, multi\_class='multinomial') on train data \*



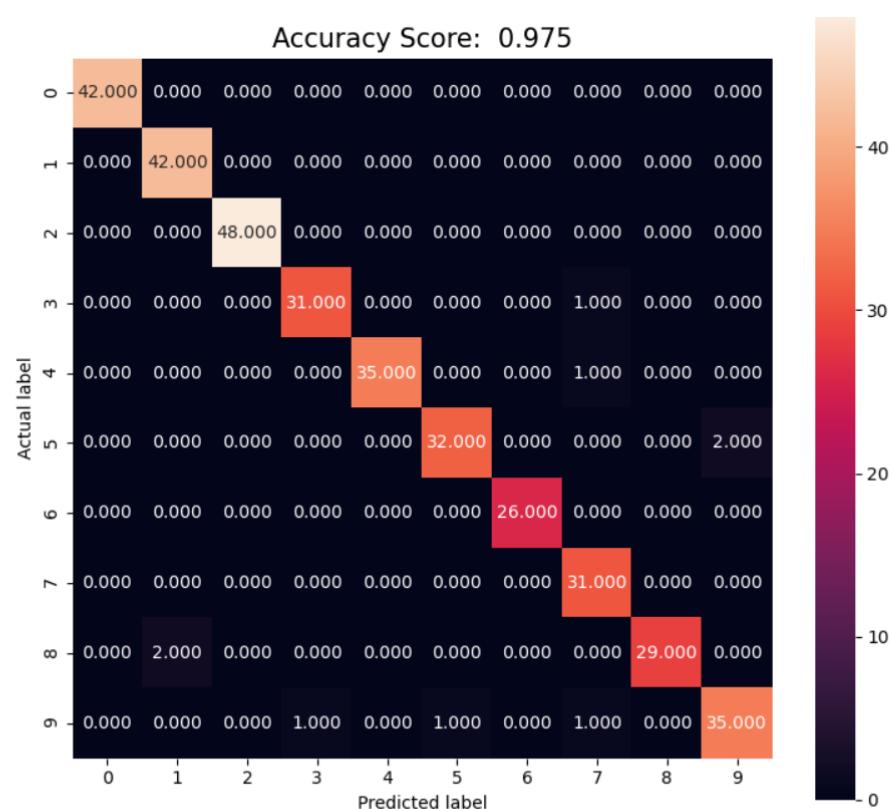
Precision-score: 0.9986082115518441

Recall-score: 0.9986082115518441

F1-score: 0.9986082115518441



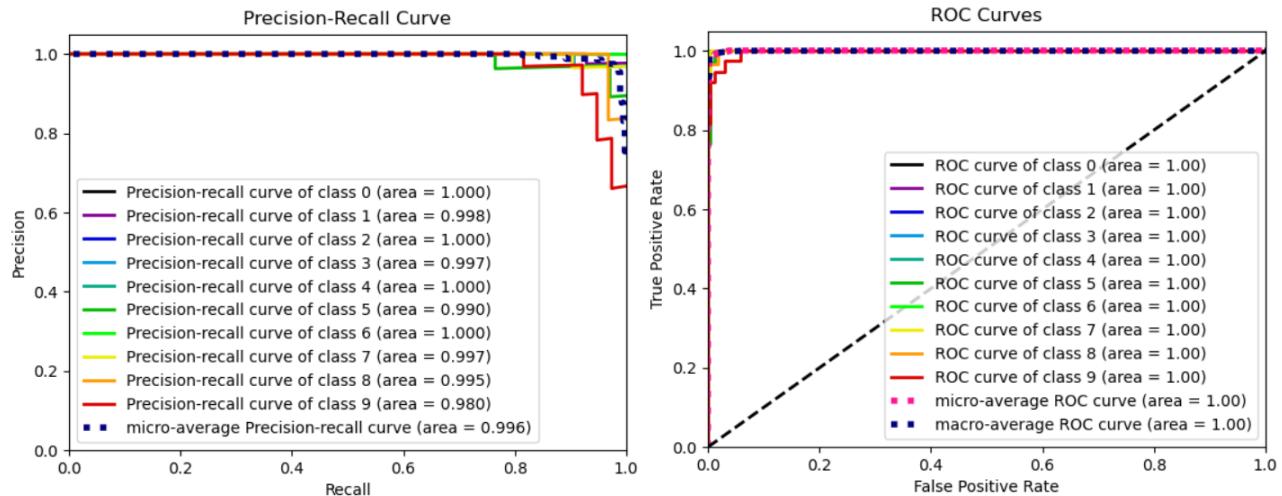
\* Metrics for LogisticRegression(max\_iter=1000, multi\_class='multinomial') on test data \*



Precision-score: 0.975

Recall-score: 0.975

F1-score: 0.975



8) для кожної моделі оцінити, чи має місце перенавчання

9) В задачах регресії розрахувати критерій якості для кожної моделі окремо на навчальній та перевірочній множинах:

```
def issue(x_true,y_true,x_test, test):
    for model in models_moon:
        print(model)
        y_predict=model.predict(x_true)
        print('train')
        print('MSE = ', mean_squared_error(y_true, y_predict))
        print('RMSE = ', mean_squared_error(y_true, y_predict, squared=False))
        print('R2 = ', r2_score(y_true, y_predict))
        print('MAE = ', mean_absolute_error(y_true, y_predict))
        print('test')
        y_predict = model.predict(x_test)
        print('MSE = ', mean_squared_error(test, y_predict))
        print('RMSE = ', mean_squared_error(test, y_predict, squared=False))
        print('R2 = ', r2_score(test, y_predict))
        print('MAE = ', mean_absolute_error(test, y_predict))
    issue(X_moons_train, y_moons_train,X_moons_test, y_moons_test)
```

```
LogisticRegression(penalty='none')
train
MSE = 0.1155555555555555
RMSE = 0.339934634239519
R2 = 0.5377777777777777
MAE = 0.1155555555555555
test
MSE = 0.07
RMSE = 0.2645751311064591
R2 = 0.72
MAE = 0.07
LogisticRegression()
train
MSE = 0.1166666666666667
RMSE = 0.3415650255319866
R2 = 0.5333333333333333
MAE = 0.1166666666666667
test
MSE = 0.08
RMSE = 0.282842712474619
R2 = 0.6799999999999999
```

```

MAE = 0.08
LogisticRegression(max_iter=1000, multi_class='multinomial', penalty='none')
train
MSE = 0.115555555555555555
RMSE = 0.339934634239519
R2 = 0.5377777777777778
MAE = 0.115555555555555555
test
MSE = 0.07
RMSE = 0.2645751311064591
R2 = 0.72
MAE = 0.07
LogisticRegression(max_iter=1500, multi_class='multinomial')
train
MSE = 0.11666666666666667
RMSE = 0.3415650255319866
R2 = 0.5333333333333333
MAE = 0.11666666666666667
test
MSE = 0.08
RMSE = 0.282842712474619
R2 = 0.6799999999999999
MAE = 0.08
train
MSE = 0.1111111111111111
RMSE = 0.3333333333333333
R2 = 0.5555533607573371
MAE = 0.1111111111111111
test
MSE = 0.15
RMSE = 0.3872983346207417
R2 = 0.3997599039615847
MAE = 0.15

```

Для першого датасету усі моделі показують на тестових наборах кращі результати, ніж на навчальних, отже, моделі не перенавчаються, а високі показники оцінок свідчать про ефективність логістичної регресії на цих даних.

Для другого датасету всі моделі на начальних наборах показують ідеальний або майже ідеальний результат, однак на тестових наборах показують незначно, але нижчі. Отже, в цьому випадку має місце перенавчання.

## 10) виконати решітчастий пошук гіперпараметрів

```

# grid search
C = np.logspace(0, 4, 10)
grid_values = [{ 'penalty': ['l1', 'l2'], 'C': C,
                 'solver': ['liblinear', 'saga']},
                { 'penalty': ['l2'], 'C': C,
                 'solver': ['newton-cg', 'lbfgs', 'sag']}]

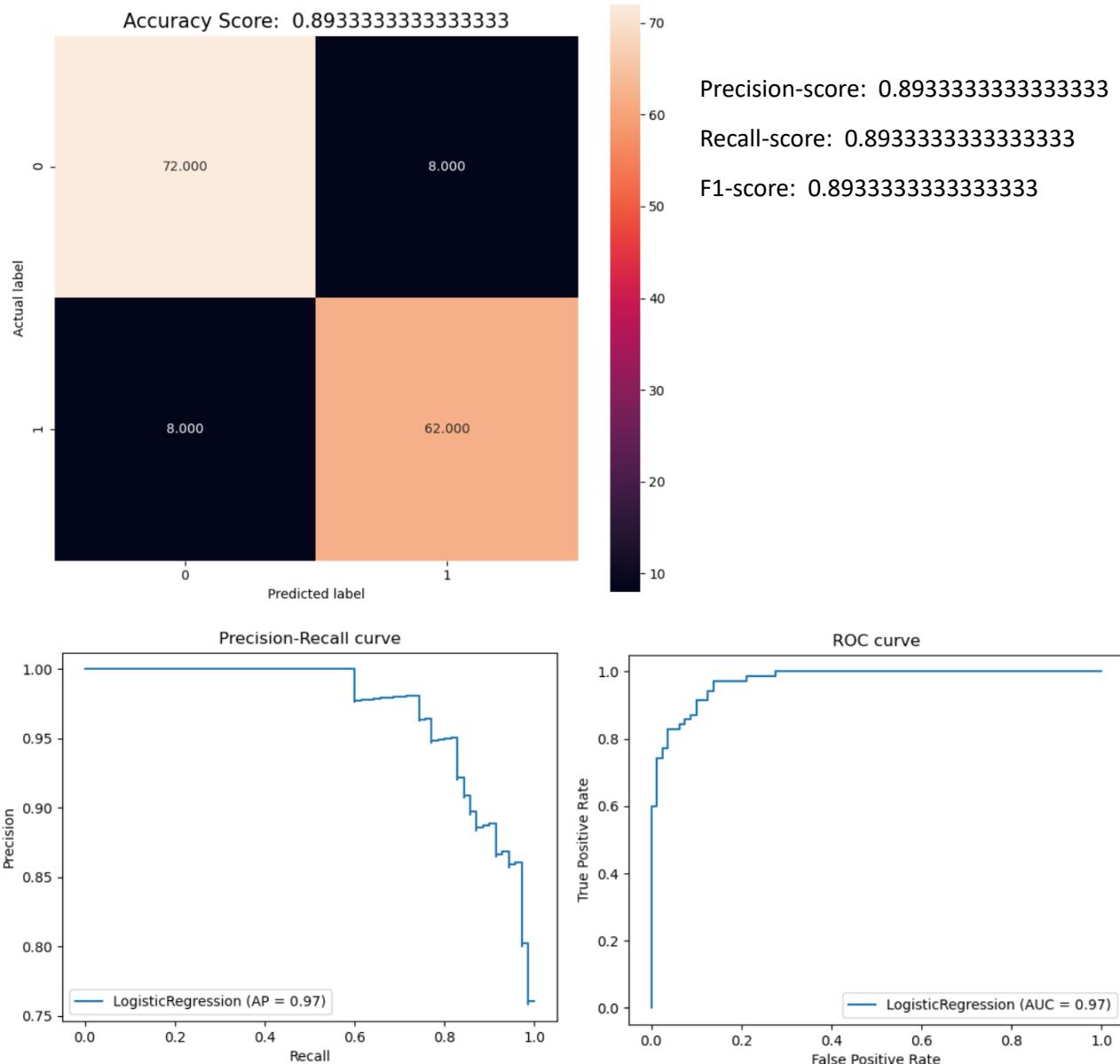
grid_search_moons = GridSearchCV(LogisticRegression(max_iter=1000), grid_values,
                                 cv=5, n_jobs=-1)
grid_search_moons.fit(X_moons_train, y_moons_train)
print('\nOptimal model for first dataset: ', grid_search_moons.best_estimator_)

```

```
print('Optimal parameters for first dataset: ', grid_search_moons.best_params_)
estimate(grid_search_moons.best_estimator_, X_moons_test, y_moons_test, 2)
```

Optimal model for first dataset: LogisticRegression(max\_iter=1000, solver='liblinear')

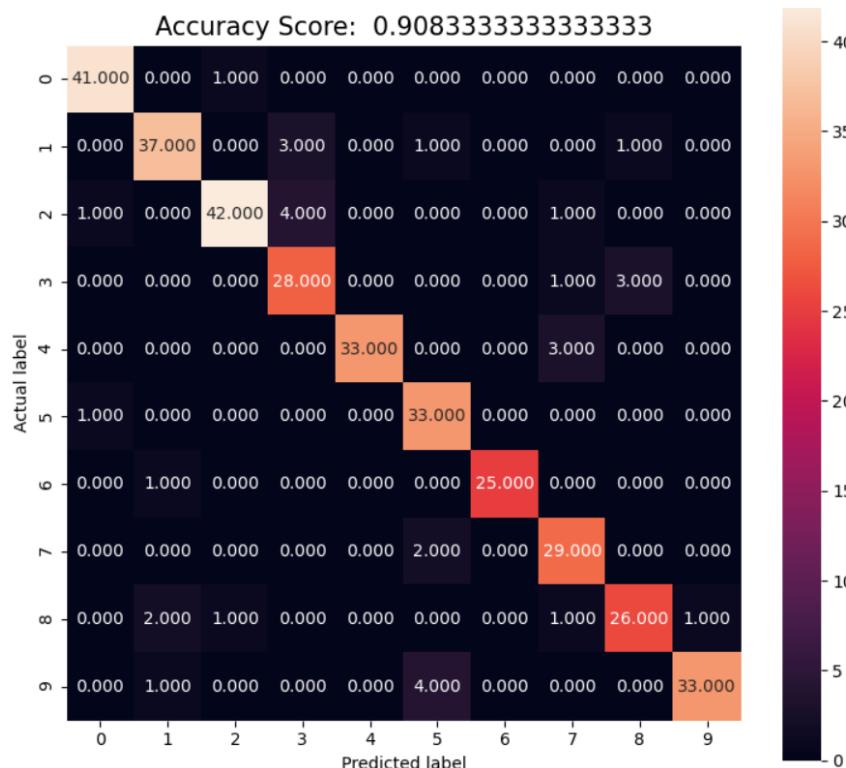
Optimal parameters for first dataset: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}



```
grid_search_digits = GridSearchCV(LogisticRegression(max_iter=20000),
grid_values, cv=5, n_jobs=-1)
grid_search_digits.fit(X_digits_train[:200], y_digits_train[:200])
print('\nOptimal model for second dataset: ',
grid_search_digits.best_estimator_)
print('Optimal parameters for second dataset: ',
grid_search_digits.best_params_)
estimate(grid_search_digits.best_estimator_, X_digits_test, y_digits_test, 10)
```

Optimal model for second dataset: LogisticRegression(C=59.94842503189409, max\_iter=20000, penalty='l1', solver='saga')

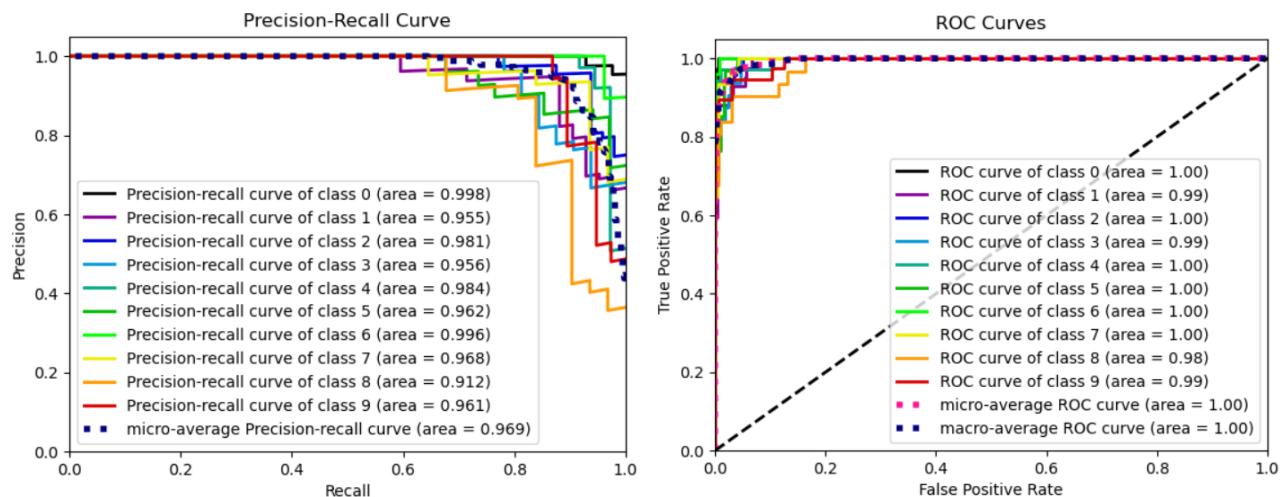
Optimal parameters for second dataset: {'C': 59.94842503189409, 'penalty': 'l1', 'solver': 'saga'}



Precision-score: 0.9083333333333333

Recall-score: 0.9083333333333333

F1-score: 0.9083333333333333



11) навчити моделі на підмножинах навчальних даних. Оцінити, наскільки розмір навчальної множини впливає на якість моделі

```
def estimate_size_quality(model, x, y):
    train_test_ratio = [i/10 for i in range(1, 10, 1)]
    f1_scores = []

    for i in train_test_ratio:
        x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=i)
        model.fit(x_train, y_train)
        f1_scores.append(f1_score(y_test, model.predict(x_test)))
```

```

f1_scores.append(mt.f1_score(y_test, model.predict(x_test),
average='micro'))

plt.plot(train_test_ratio, f1_scores)
plt.ylabel("Model's f1 score")
plt.show()

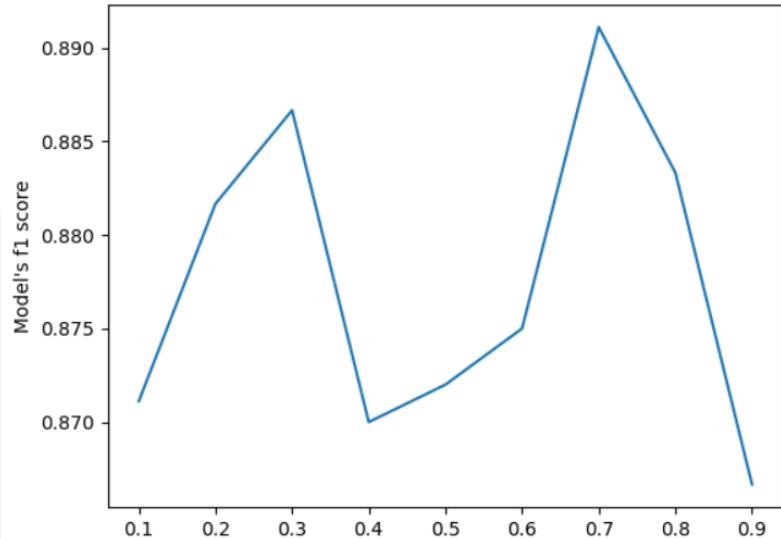
```

```

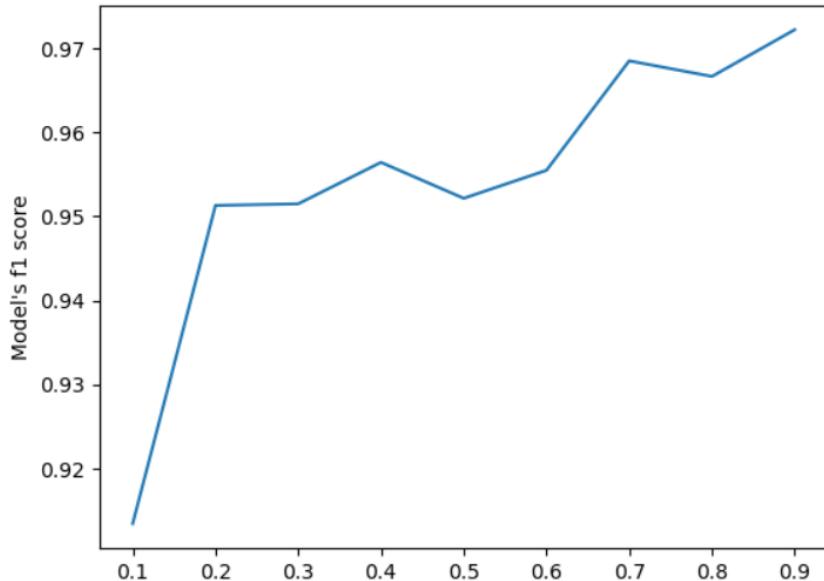
# estimate models depending on train data
estimate_size_quality(grid_search_moons.best_estimator_, X_moons, y_moons)
estimate_size_quality(grid_search_digits.best_estimator_, X_digits, y_digits)

```

Перший датасет:



Другий датасет:



\*по осі x – частка навчального набору даних від загального набору

Як видно з графіків, для першого набору якість моделі різко спадає та зростає, залежно від розміру вибірки, а для другого набору якість стає вищою зі зростанням навчальних даних.

**Висновки (12):** для обох дасатетів всі моделі працюють приблизно однаково з незначними відхиленнями, проте моделі без регуляризації працюють трохи краще. Для другого датасету моделі показують, в цілому, вищі результати, однак для них є характерним незначне перенавчання, що не спостерігається на першому датасеті.

## Код программы

```
from sklearn.datasets import load_digits
from sklearn.datasets import make_moons
from matplotlib import pyplot as plt
from pandas import DataFrame
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import RocCurveDisplay
from mlxtend.plotting import plot_decision_regions
from sklearn.metrics import plot_confusion_matrix
import scikitplot.metrics as skplt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn import metrics as mt
from sklearn.metrics import r2_score
import seaborn as sns
X_moons, y_moons = make_moons(n_samples=1000, noise=0.1)      ##### data set 1
digits_ = load_digits()                                         ##### data set 2

----- grafic performance of data sets -----
def show_moons(X, y):
    df = DataFrame(dict(x=X[:, 0], y=X[:, 1], label=y))
    colors = {0: 'magenta', 1: 'blue'}
    fig, ax = plt.subplots()
    grouped = df.groupby('label')
    for key, group in grouped:
        group.plot(ax=ax, kind='scatter', x='x', y='y', label='Class #' + str(key),
color=colors[key])
    plt.show()

#show_moons(X_moons, y_moons)
def show_digits(digits):
    fig, axes = plt.subplots(8, 8, figsize=(7, 7), subplot_kw={'xticks': [], 'yticks': []},
                           gridspec_kw=dict(hspace=0.1, wspace=0.1))
    for i, ax in enumerate(axes.flat):
        ax.imshow(digits.images[i], cmap='RdPu', interpolation='nearest')
        ax.text(0.05, 0.05, str(digits.target[i]), transform=ax.transAxes, color='fuchsia')
    plt.show()
#show_digits(digits_)

----- divide into train and test samples-----
y_digits = digits_.target
X_digits = digits_.data
X_moons_train, X_moons_test, y_moons_train, y_moons_test = train_test_split(X_moons, y_moons,
test_size=0.1)
X_digits_train, X_digits_test, y_digits_train, y_digits_test = train_test_split(X_digits, y_digits,
test_size=0.2)
X_moons = StandardScaler().fit_transform(X_moons)
X_digits = StandardScaler().fit_transform(X_digits)

----- creation of models-----
def create_models(X_train, y_train):
    simple_lr = LogisticRegression(penalty='none').fit(X_moons_train, y_moons_train) # without regularization
    simple_lr_regularized = LogisticRegression().fit(X_moons_train, y_moons_train) # with regularization
    multinomial_lr = LogisticRegression(penalty='none', multi_class='multinomial', solver='lbfgs',
max_iter=1000).fit(
        X_train, y_train) # without regularization
    multinomial_lr_regularized = LogisticRegression(multi_class='multinomial', solver='lbfgs',
max_iter=1500).fit(
        X_moons_train, y_moons_train) # with regularization
    return simple_lr, multinomial_lr, simple_lr_regularized, multinomial_lr_regularized
```

```

moons_simple, moons_multinomial, moons_simple_reg, moons_multinomial_reg =
create_models(x_moons_train, y_moons_train)
digits_simple, digits_multinomial, digits_simple_reg, digits_multinomial_reg =
create_models(x_digits_train, y_digits_train)

----- show simple model without regularization for first dataset-----
plot_decision_regions(X_moons, y_moons, clf=moons_simple, legend=4)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('moons dataset without regularization')
# plt.show()

plot_decision_regions(X_moons, y_moons, clf=moons_simple_reg, legend=4)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('moons dataset with regularization')
# plt.show()

plot_decision_regions(X_moons, y_moons, clf=moons_simple_reg, legend=4)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('multinomial moons dataset without regularization')
# plt.show()

plot_decision_regions(X_moons, y_moons, clf=moons_multinomial, legend=4)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('multinomial moons dataset with regularization')
# plt.show()

#####-----confusion matrix-----
models_moon = [moons_simple, moons_simple_reg, moons_multinomial, moons_multinomial_reg]
models_digits = [digits_simple, digits_simple_reg, digits_multinomial, digits_multinomial_reg]

def create_matrix(curr_model, x, y_test, y_predict):
    cm = mt.confusion_matrix(y_test, y_predict)
    score = curr_model.score(x, y_test)
    plt.figure(figsize=(9, 9))
    sns.heatmap(cm, annot=True, fmt=".3f", square=True)
    plt.ylabel("True values")
    plt.xlabel("Predicted values")
    plt.title(f"Accuracy : {score}\n\nconfusion matrix for {curr_model}", size=15)
    plt.show()

def create_matrix2(curr_model, x, y_test, y_predict):
    t=y_predict
    plot_confusion_matrix(curr_model, x, y_test)
    plt.show()

```

```

-----quality criteria,-----
def estimate(curr_model, x, y_true, num_classes):
    y_predict = curr_model.predict(x)
    create_matrix(curr_model, x, y_true, y_predict)
    print({curr_model})
    print('Precision = ', mt.precision_score(y_true, y_predict, average='micro'))
    print('Recall = ', mt.recall_score(y_true, y_predict, average='micro'))
    print('F1-score = ', mt.f1_score(y_true, y_predict, average='micro'))

    if num_classes == 2:
        pr_curve = PrecisionRecallDisplay.from_predictions(y_true, y_predict)
        pr_curve.ax_.set_title('Precision-Recall curve')
        plt.show()

```

```

roc_curve_moons = RocCurveDisplay.from_predictions(y_true, y_predict)
roc_curve_moons.ax_.set_title('ROC curve')
plt.show()
y_probs = curr_model.predict_proba(x)

if num_classes > 2:
    skplt.plot_precision_recall(y_true, y_probs)
    plt.show()

    skplt.plot_roc(y_true, y_probs)
    plt.show()

```

```

#-----METRICS FOR MOONS-----
def ississue(x_true,y_true,x_test, test):
    for model in models_digits:
        print(model)
        y_predict = model.predict(x_true)
        print('train')
        print('MSE = ', mean_squared_error(y_true, y_predict))
        print('RMSE = ', mean_squared_error(y_true, y_predict, squared=False))
        print('R2 = ', r2_score(y_true, y_predict))
        print('MAE = ', mean_absolute_error(y_true, y_predict))
        print('test')
        y_predict = model.predict(x_test)
        print('MSE = ', mean_squared_error(test, y_predict))
        print('RMSE = ', mean_squared_error(test, y_predict, squared=False))
        print('R2 = ', r2_score(test, y_predict))
        print('MAE = ', mean_absolute_error(test, y_predict))
ississue(X_digits_train, y_digits_train,X_digits_test, y_digits_test)

# create models
moons_simple, moons_multinomial, moons_simple_reg, moons_multinomial_reg = \
    create_models(X_moons_train, y_moons_train)
digits_simple, digits_multinomial, digits_simple_reg, digits_multinomial_reg \
= \
    create_models(X_digits_train, y_digits_train)

models = [moons_simple, moons_simple_reg, moons_multinomial,
moons_multinomial_reg, digits_simple,
           digits_simple_reg, digits_multinomial, digits_multinomial_reg]

# show simple model without regularization for first dataset
plot_decision_regions(X_moons, y_moons, clf=moons_simple)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('first dataset')
plt.show()

# make predicts
print('-----METRICS FOR MOONS-----')
for model in models[:4]:

    print('\n* Metrics for ', model, ' on train data *\n')
    estimate(model, X_moons_train, y_moons_train, 2)
    print('\n* Metrics for ', model, ' on test data *\n')
    estimate(model, X_moons_test, y_moons_test, 2)

print('-----METRICS FOR DIGITS-----')
for model in models[4:]:
    print('\n* Metrics for ', model, ' on train data *\n')
    estimate(model, X_digits_train, y_digits_train, 10)
    print('\n* Metrics for ', model, ' on test data *\n')
    estimate(model, X_digits_test, y_digits_test, 10)

```

```
# grid search
C = np.logspace(0, 4, 10)
grid_values = [{ 'penalty': ['l1', 'l2'], 'C': C,
                 'solver': ['liblinear', 'saga'] },
                { 'penalty': ['l2'], 'C': C,
                 'solver': ['newton-cg', 'lbfgs', 'sag']}]

grid_search_moons = GridSearchCV(LogisticRegression(max_iter=1000),
grid_values, cv=5, n_jobs=-1)
grid_search_moons.fit(X_moons_train, y_moons_train)
print('\nOptimal model for first dataset: ',
grid_search_moons.best_estimator_)
print('Optimal parameters for first dataset: ',
grid_search_moons.best_params_)
estimate(grid_search_moons.best_estimator_, X_moons_test, y_moons_test, 2)

grid_search_digits = GridSearchCV(LogisticRegression(max_iter=20000),
grid_values, cv=5, n_jobs=-1)
grid_search_digits.fit(X_digits_train[:200], y_digits_train[:200])
print('\nOptimal model for second dataset: ',
grid_search_digits.best_estimator_)
print('Optimal model for second dataset: ', grid_search_digits.best_params_)
estimate(grid_search_digits.best_estimator_, X_digits_test, y_digits_test,
10)

# estimate models depending on train data
estimate_size_quality(grid_search_moons.best_estimator_, X_moons, y_moons)
estimate_size_quality(grid_search_digits.best_estimator_, X_digits, y_digits)
```