Спочатку я просто зчитую дані, дивлюсь чи вони збалансовані та малюю їх:

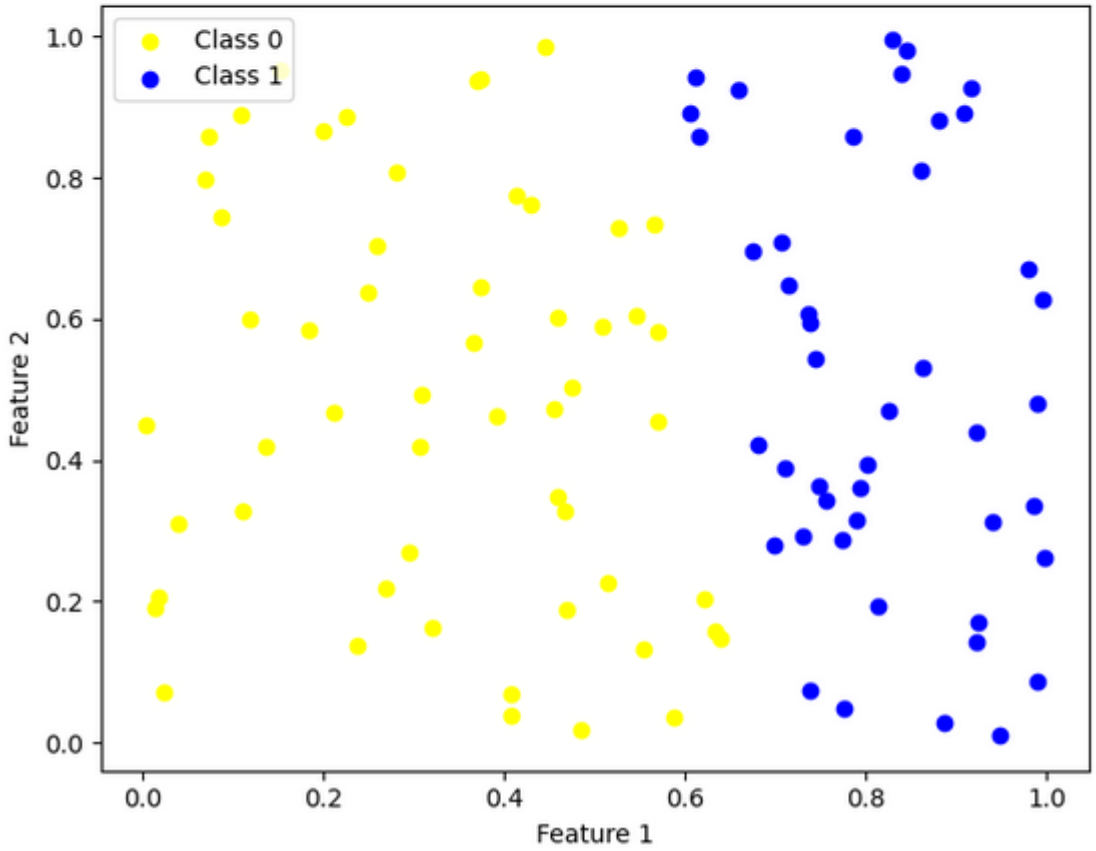
```
file_path = './data/data09.csv'
df = pd.read_csv(file_path, header=None, delimiter=';')
df.head()
```

	0	1	2
0	0.774	0.288	1
1	0.989	0.480	1
2	0.135	0.419	0
3	0.802	0.395	1
4	0.924	0.172	1

```
df[2].value_counts()
```

```
2
0    55
1    45
Name: count, dtype: int64
```

```
plot_classes(df)
```



Після цього я розбиваю дані на трейнові та тестові, треную на них SLP та знову малюю ті ж дані, але тепер з прямою, натренованою за допомогою SLP:

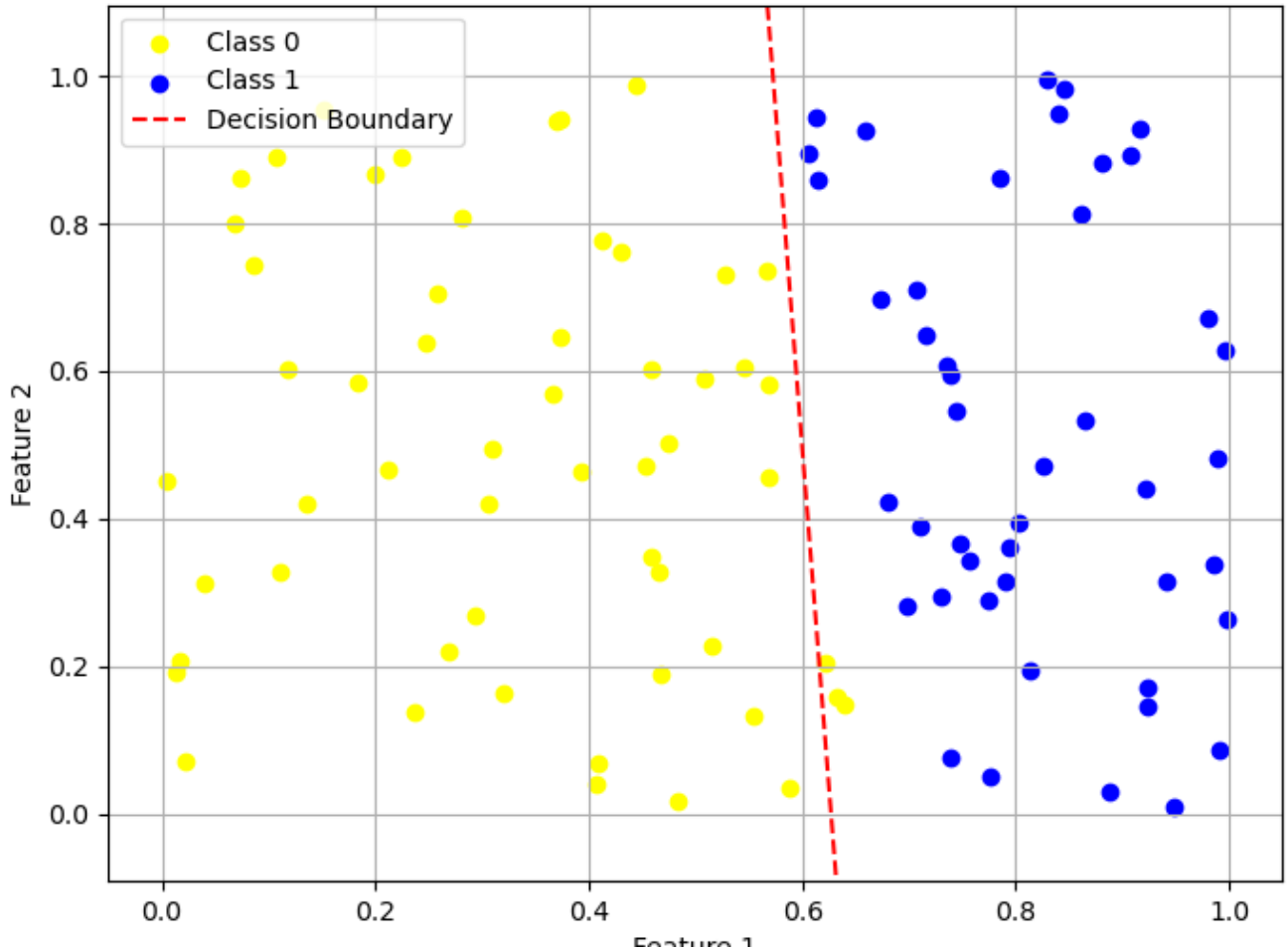
```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

```
model = SLP()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"Accuracy on test data: {accuracy(y_pred, y_test)}")
```

```
Cost after epoch 0 = 0.835381114179155
Weights after epoch w1 = -1.0647209377247837 w2 = 0.9708305967132712
Bias after epoch 0 = 3.644524895527918e-05

Cost after epoch 10000 = 0.30654057898729414
Weights after epoch w1 = 4.989926249555559 w2 = -0.1621374735106393
Bias after epoch 10000 = -2.746237739808501

Cost after epoch 20000 = 0.2215995035817268
Weights after epoch w1 = 7.442948595776269 w2 = -0.11639393076440087
Bias after epoch 20000 = -4.260001793376221

Cost after epoch 30000 = 0.1828603402447771
Weights after epoch w1 = 9.072120271621852 w2 = 0.0355925411841861
Bias after epoch 30000 = -5.340769745621257

Cost after epoch 40000 = 0.15942337794042263
Weights after epoch w1 = 10.329861384674553 w2 = 0.18728628006282969
Bias after epoch 40000 = -6.195198585430621

Cost after epoch 50000 = 0.14328109040314568
Weights after epoch w1 = 11.370772370527135 w2 = 0.3237268646991769
Bias after epoch 50000 = -6.908274255128667

Cost after epoch 60000 = 0.13128791920853938
Weights after epoch w1 = 12.267220223720622 w2 = 0.4446905511353472
Bias after epoch 60000 = -7.524106832413903

Cost after epoch 70000 = 0.1219201428614388
Weights after epoch w1 = 13.059417701072176 w2 = 0.5524755915125042
Bias after epoch 70000 = -8.0686840773982

Cost after epoch 80000 = 0.11433779325192457
Weights after epoch w1 = 13.772262219473298 w2 = 0.6494493196239318
Bias after epoch 80000 = -8.558646227010499
```

```
Cost after epoch 90000 = 0.10803477536925747
Weights after epoch w1 = 14.422358480068345 w2 = 0.7375823478065727
Bias after epoch 90000 = -9.005307912410736

Accuracy on test data: 0.9
```

```
plot_decision_bound(df, model)
```



Після цього я порівнюю свій результат з вбудованим у sklearn SLP

```
perceptron = Perceptron(max_iter=100000, eta0=0.01, random_state=42)
perceptron.fit(X_train, y_train)
y_pred = perceptron.predict(X_test)
acc = accuracy(y_pred, y_test)
print(f"Accuracy: {acc}")
```

```
Accuracy: 0.8666666666666667
```

Як видно точність сильно відрізняється, я передивився документацію, та вимкнув усі значення по замовчуванню, але це не змінило точність, тому мені здається, що такі розбіжності в точності можуть бути через те що я не так генерую ваги, як в sklearn.

Реалізація кастомного SLP:

```python
class SLP:
    def __init__(self, learning_rate=0.01, num_epochs=100000):
        self.learning_rate = learning_rate
        self.num_epochs = num_epochs

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)
        n_in = len(X[0])
        n_out = 1

        # Xavier Initialization
        limit = np.sqrt(6 / (n_in + n_out))
        self._w = np.random.uniform(-limit, limit, size=(n_in, n_out))
        self._b = 0.0

        # propagate
        for epoch in range(self.num_epochs):
            m = X.shape[0]
            A = 1 / (1 + np.exp(-(np.dot(X, self._w) + self._b)))
            cost = (-1 / m) * np.sum(np.dot(y, np.log(A)) + np.dot(1 - y, np.log(1 - A)))
            dw = (1 / m) * np.dot(X.T, (A - y.reshape(-1, 1)))
            db = (1 / m) * np.sum(A - y.reshape(-1, 1))

            # update rule
            self._w = self._w - self.learning_rate * dw
            self._b = self._b - self.learning_rate * db

            if epoch % 10000 == 0:
                print(f"Cost after epoch {epoch} = {cost}")
                print(f"Weights after epoch {epoch}: w1 = {self._w[0, 0]} w2 = {self._w[1, 0]}")
                print(f"Bias after epoch {epoch} = {self._b}")
                print()

        print(f"Cost after epoch {self.num_epochs} = {cost}")
        print(f"Weights after epoch {self.num_epochs}: w1 = {self._w[0, 0]} w2 = {self._w[1, 0]}")
        print(f"Bias after epoch {self.num_epochs} = {self._b}")
        print()

    def predict(self, X):
        X = np.array(X)
        m = X.shape[0]
        y_prediction = np.zeros((m, 1))
        A = 1 / (1 + np.exp(-(np.dot(X, self._w) + self._b)))

        for i in range(A.shape[0]):
            if A[i, 0] > 0.5 :
                y_prediction[i, 0] = 1
            else:
                y_prediction[i, 0] = 0

        return y_prediction
```

Спочатку ми ініціалізуємо модель льорнінг рейтом та кількістю епох. Далі в методі фіт ми ініціалізуємо ваги та біас за методом Xavier, після цього вже у циклі, який працює стільки скільки у нас епох я обраховую результат роботи нейрона та cost, а також похідні cost за w та b та апдейчу ваги та біас згідно з алгоритмом градієнтного спуску. Ну і в методі predict я просто предікчузначення використовуючи натреновані ваги та біас.