Iris

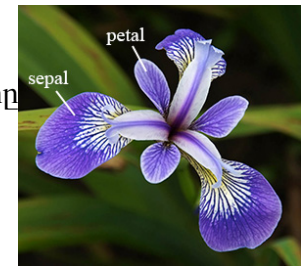
Էրիկ Մկրտչյան

November 2024

Իրիս ծաղիկների դասակարգում

Իրիս ծաղիկների դասակարգման խնդիրը մեքենայական ուսուցման ամենահայտնի օրինակներից մեկն է, որը լայնորեն օգտագործվում է տվյալների վերլուծության և դասակարգման մեջ հիմնական հասկացությունների ներդրման համար:
Տվյալների հավաքածուն, որը ներկայացվել է Ռոնալդ Ա. Ֆիշերի կողմից 1936թ.-ին, բաղկացած է 150 նմուշներից, որոնք բաժանված են երեք տեսակի Iris ծաղիկների՝

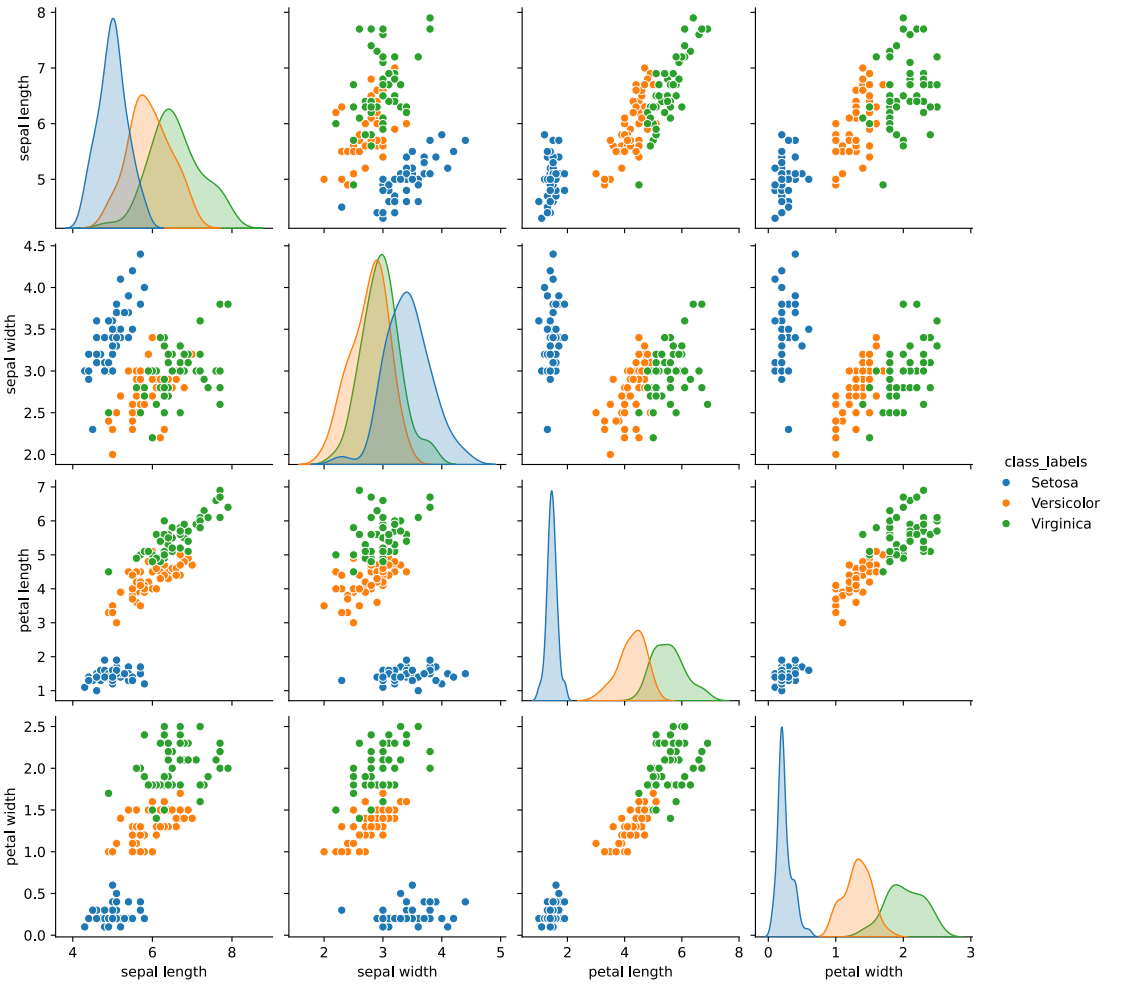
Iris setosa, Iris versicolor և Iris virginica : Յուրաքանչյուր նմուշ նկարագրվում է չորս թվային հատկանիշներով՝ բաժակի երկարություն, բաժակի լայնություն, ծաղկաթերթի երկարություն և ծաղկաթերթի լայնություն: Այս հատկանիշները հիմք են տալիս երեք տեսակների տարբերակման համար: Հավաքածուի պարզությունն ու հավասարակշռված բնույթը այն դարձնում են իդեալական դասակարգման տեխնիկան ուսումնասիրելու և մեքենայական ուսուցման մոդելների գործունակությունը գնահատելու համար:



Ստորև ներկայացված է Iris տվյալների բազան scikit-learn- ից ներմուծելու կոդը.

```python
from sklearn import datasets

iris = datasets.load_iris()

print(len(iris.data))
print(iris.data[:5])
print(iris.target)

#output--------------------------------------------------------------
# 150
#[[5.1 3.5 1.4 0.2]
# [4.9 3.  1.4 0.2]
# [4.7 3.2 1.3 0.2]
# [4.6 3.1 1.5 0.2]
# [5.  3.6 1.4 0.2]]
#[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
# 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
# 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
# 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

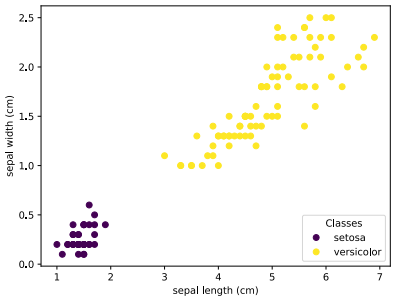
Թվային փվյալների դիտումը կարող է լինել ճնշող և դժվար մեկնաբանելի, հատկապես երբ գործ ունենք բազմաթիվ թվային արժեքների հետ: Տվյալների հավաքածուի օրինաչափությունները և հարաբերությունները ավելի լավ հասկանալու համար մենք կարող ենք պատկերել փվյալները՝ օգտվելով գրադարաններից: Այս վիզուալ ներկայացումները հեշտացնում են օրինաչափությունների հայտնաբերումը, առանձնահատկությունները համեմատելը և պատկերացումներ ձեռք բերելը, ինչը անմիջապես ակնհայտ չե թվերից:



Պատկերից մենք կարող ենք նկատել հստակ տարբերություն կապույտ կետերի և մնացած կետերի միջև: Սա թույլ է տալիս մեզ պատկերացնել մի գիծ, որը կարող է առանձնացնել կապույտ կետերը մնացածից: Այժմ հենց դա էլ կանենք սակայն ամբողջ փվյալները ավելորդ են, դրա համար կվերցնենք միայն բաժակի լայնությունն ու երկարությունը, այնույեպև կիսենք փվյալները սովորելու և սպուգելու ցուցակների՝

```
X = iris.data[:, 2:]
y = (iris.target != 0) * 1


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2)
```

Տվյալների հավաքածուից սովորելու համար մեզ անհրաժեշտ է Պերցեպտրոն դաս: Պերցեպտրոն-ը պարզ, բայց հզոր ալգորիթմ է երկուական դասակարգման առաջադրանքների համար: Այն աշխատում է՝ գտնելով գծային սահման, որը բաժանում է տարբեր դասեր տվյալների մեջ՝ դարձնելով այն հիմք հանդիսացող մեքենայական ուսուցման ավելի բարդ մոդելները հասկանալու համար:

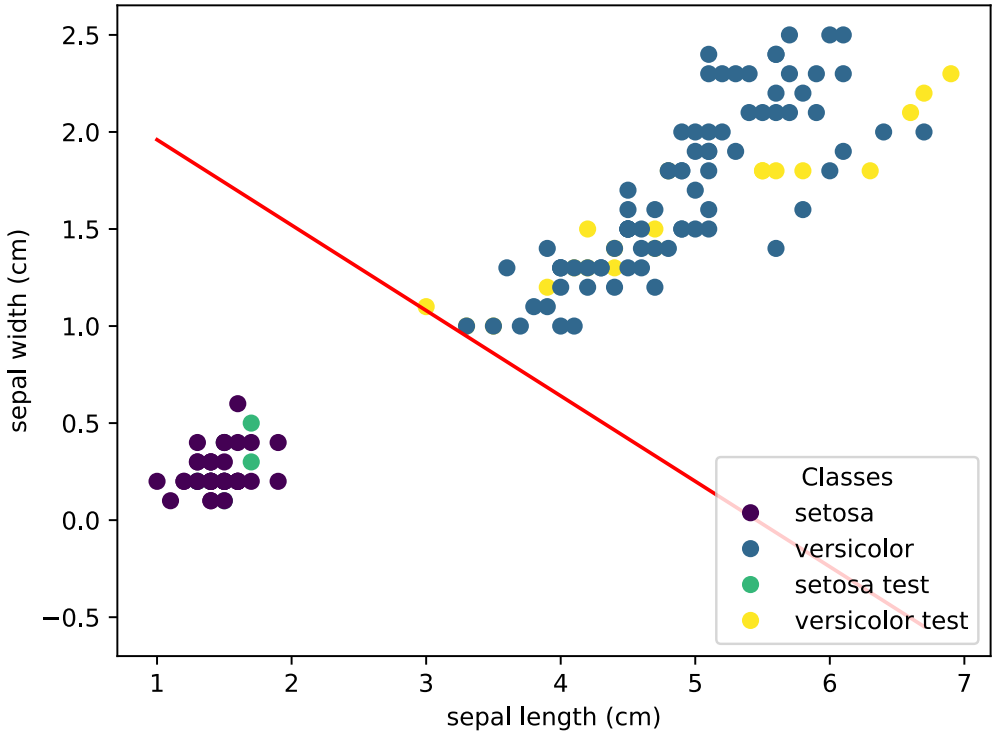



```python
import numpy as np

class Perceptron:
    def __init__(self, learning_rate=0.01, n_iterations=2000):
        self.lr = learning_rate
        self.n_iterations = n_iterations

    def fit(self, X, y):
        self.w = np.zeros(X.shape[1])
        self.b = 0
        for _ in range(self.n_iterations):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.w) + self.b
                y_predicted = self.activation_function(
    linear_output)
                update = self.lr * (y[idx] - y_predicted)
                self.w += update * x_i
                self.b += update

    def activation_function(self, x):
        return np.where(x >= 0, 1, 0)

    def predict(self, X):
        linear_output = np.dot(X, self.w) + self.b
        return self.activation_function(linear_output)
```

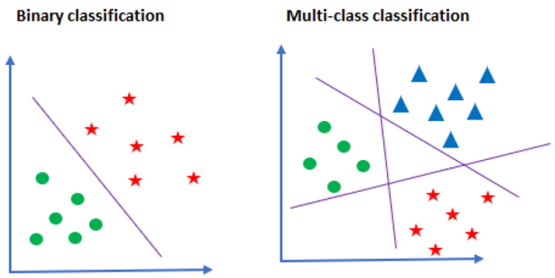
Այժմ մենք պատրաստ ենք վարժեցնել մոդելը: Սփորև բերված է Պերցեպտրոն-ի սովորելու և փորձարկելու կոդը՝

```python
model = Perceptron()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

accuracy = np.mean(predictions == y_test)
print(f"Accuracy: {accuracy * 100:.2f}%")

#output-----------
#Accuracy: 100.00%
```

Վարժեցրած մոդելով մենք այժմ կարող ենք պատկերել փվյալների հավաքածուն որոշման սահմանագծի հետ միասին: Այս տեսողական ներկայացումը ցույց է տալիս, թե որքան լավ է Պերցեպտրոն-ը բաժանում դասերը:



Հաջորդ քայլը բոլոր երեք դասերի միջև տարբերակումն է: Դրա համար մենք կօգտագործենք One-vs-Rest (OvR) ալգորիթմը: OvR -ն աշխատում է յուրաքանչյուր դասի համար առանձին երկուական դասակարգիչ պատրաստելով, մեկ դասը դիտարկելով որպես դրական, իսկ մնացած դասերը որպես բացասական: Այնուհետև որպես կանխատեսում ընտրվում է ամենաբարձր վստահության միավոր ունեցող դասը:



Այնուամենայնիվ, Iris փվյալների բազայի վրա OvR կիրառելիս մենք գտնում ենք, որ կանխատեսման ճշգրտությունը ցածր է: Դա տեղի է ունենում այն պատճառով, որ մնացած երկու դասերը (Iris versicolor և Iris virginica) գծայինորեն բաժանելի չեն, ինչը նշանակում է, որ ոչ մի ուղիղ գիծ չի կարող պատշաճ կերպով բաժանել իրենց փվյալների կետերը հատկանիշի տարածության մեջ:

Այս խնդիրը լուծելու համար մենք պետք է թարմացնենք մեր Պերցեպտրոն մոդելը բազմաշերտ պերցեպտրոնի (MLP): MLP -ն ավելի առաջադեմ նեյրոնային ցանց է, որն օգտագործում է նեյրոնների բազմաթիվ շերտեր և ոչ գծային ակտիվացման ֆունկցիաներ: Սա թույլ է տալիս նրան սովորել ավելի բարդ որոշումների սահմանները, ինչը նրան հնարավորություն է տալիս արդյունավետորեն մշակել ոչ գծային բաժանվող փվյալները:

Մենք կիրականացնենք MLP դասը՝ օգտագործելով PyTorch -ը, որը հայտնի open source խորը ուսուցման գրադարան է: PyTorch -ը ճկունությամբ և հեշտությամբ նեյրոնային ցանցեր կառուցելու և վարժեցնելու գործիքներ է տրամադրում, ինչը այն դարձնում է իդեալական մեքենայական ուսուցման մոդելների փորձարկումների համար: Ստորև բերված է MLP դասի կոդը.

```python
import torch
import torch.nn as nn
import torch.optim as optim

class irisMlp(nn.Module):
    def __init__(self,input_size = 4,hidden_size = 100, output_size = 3):
        super(irisMlp,self).__init__()

        self.hidden = nn.Linear(input_size,hidden_size)
        self.relu = nn.ReLU()
        self.output = nn.Linear(hidden_size,output_size)

    def forward(self,x):
        x = self.hidden(x)
        x = self.relu(x)
        x = self.output(x)
        return x

    def train_model(self, train_loader, learning_rate=0.001, epochs=25):
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(self.parameters(), lr=learning_rate)

        # Training loop
        self.train()  # Set the model to training mode
        for epoch in range(epochs):
            running_loss = 0.0
            for inputs, labels in train_loader:
                # Zero the parameter gradients
                optimizer.zero_grad()

                # Forward pass
                outputs = self(inputs)
                loss = criterion(outputs, labels)

                # Backward pass and optimize
                loss.backward()
                optimizer.step()

                # Track loss
                running_loss += loss.item()

            # Print the average loss for the epoch
            print(f"Epoch [{epoch + 1}/{epochs}], Loss: {running_loss / len(train_loader):.4f}")

    def test_model(self, test_loader):
        criterion = nn.CrossEntropyLoss()

        # Testing loop
        self.eval()  # Set the model to evaluation mode
        correct = 0
        total = 0
```

```
        test_loss = 0.0

        with torch.no_grad():
            for inputs, labels in test_loader:
                # Forward pass
                outputs = self(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()

                # Get predictions
                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        # Print accuracy and average loss
        print(f"Test Accuracy: {100 * correct / total:.2f}%")
        print(f"Test Loss: {test_loss / len(test_loader):.4f}")
```

Հաջորդը, մենք սպեղծում ենք IrisMLP դասը, որն օգտագործում է MLP մոդելը՝ Iris փվյալների բազան դասակարգելու համար: Մենք սկսում ենք փվյալները բաժանելով վերապատրաստման և թեստավորման հավաքածուների՝ օգտագործելով train_test_split -ը, որպեսզի ապահովվի, որ մոդելը արդյունավետ կերպով սովորում է չտեսնված փվյալների վրա գնահատվեն: Այնուհետև մենք սպեղծում ենք DataLoader օբյեկտները ուսուցման, և՛ փորձարկման համար, որոնք հեշտացնում են փվյալների հավաքումն ու խառնումը մարզման ընթացքում: Ի վերջո, մենք ուսուցանում ենք մոդելը վերապատրաստման փվյալների վրա և գնահատում ենք դրա կատարումը թեստավորման փվյալների վրա:

```
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
iris = datasets.load_iris()
model = irisMlp()

# Load and preprocess the Iris dataset
iris = datasets.load_iris()
inputs = torch.from_numpy(iris.data).float()
labels = torch.from_numpy(iris.target).long()

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(inputs, labels,
    test_size=0.3, random_state=42)

# Create DataLoader for train and test sets
train_loader = DataLoader(TensorDataset(X_train, y_train),
   batch_size=16, shuffle=True)
test_loader = DataLoader(TensorDataset(X_test, y_test), batch_size
   =16, shuffle=False)

# Train and test the model
model.train_model(train_loader)
model.test_model(test_loader)

# output--------------------
# Epoch [1/25], Loss: 1.2660
# Epoch [2/25], Loss: 1.0942
# ...
# Epoch [24/25], Loss: 0.4267
# Epoch [25/25], Loss: 0.4254
# Test Accuracy: 100.00%
# Test Loss: 0.3458
```

Եզրափակելով՝ մենք ուսումնասիրեցինք Iris փվյալների բազան, պատկերացրինք դրա օրինաչափությունները և կառուցեցինք մոդելներ՝ դրա փվյալները դասակարգելու համար: Երկուական դասակարգման համար Perceptron ով իսկ, երեք դասի համար MLP , օգտագործելով PyTorch -ը՝ ոչ գծային բաժանելիությունը կարգավորելու և բոլոր երեք դասերը դասակարգելու համար: Այս գործընթացը ընդգծում է նեյրոնային ցանցերի հզորությունը դասակարգման բարդ խնդիրների լուծման գործում: