First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

Introduction

The Perceptron algorithm is a fundamental building block in machine learning, representing one of the earliest supervised learning methods for classification tasks. In this project, we apply Python's scikit-learn library to train a Perceptron classifier on the well-known Iris dataset—a standard for learning and demonstrating classification concepts.

The workflow includes:

- Loading and understanding the dataset
- Preparing and filtering features and labels
- Splitting data into training and testing sets
- Standardizing features for optimal learning
- Training and evaluating the model
- Visualizing the learned decision boundary

This hands-on project not only reinforces core ML concepts but also prepares us for more sophisticated classifiers by highlighting both the strengths and weaknesses of linear models.

Theoretical Background

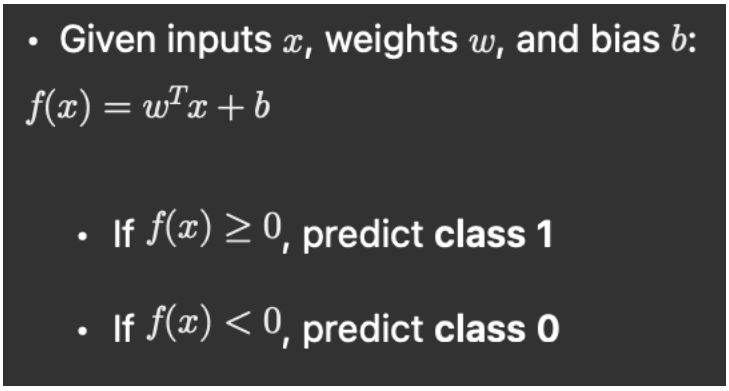
What is a Perceptron?

A Perceptron is a type of linear classifier—a mathematical model that attempts to find a hyperplane (straight line in 2D, plane in 3D, etc.) that separates two classes. The Perceptron algorithm iteratively adjusts the weights assigned to each input feature based on the prediction error until the model converges (if possible).

First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

Mathematical Formulation:



Key Properties:

- Binary classifier: Only distinguishes between two classes.
- Linear: Can only separate data with a straight line (or hyperplane).
- Deterministic output: Gives hard class labels (not probabilities).
- Sensitive to feature scale: Standardization is necessary for good results.

Limitations

- Only works if the data is linearly separable (can be perfectly split by a straight line).
- Fails on data where classes overlap in feature space.
- Cannot output probabilities—only labels.
- Sensitive to feature scaling and outliers.

Why Use the Iris Dataset?

- Simple and ideal for visualization.
- Contains three species (we focus on two for binary classification).
- Using petal length and width as features, Setosa and Versicolor are linearly separable.

First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

Data Preparation and Workflow

Step 1: Loading and Exploring the Data

We first load our dataset (from a CSV or using scikit-learn's built-in loader).

If using a CSV, assign appropriate column names and check for data consistency:

```
# Perceptron on Iris dataset using scikit-learn

from sklearn import datasets
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import pandas as pd


# 1. Load and filter the data
iris = pd.read_csv('/Users/shivesh/Desktop/PythonProject/Python Machine Learning Textbook/Chapter 2/Perceptron (Iris DataSet)/iris.data.csv')

# Assign column names
iris.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']

# Check the data
print("\nGetting to know the dataset:")
print(iris.columns)
print(iris.shape)
print(iris.head())
print(iris.describe())
print("\nCheck for null values:")
print(iris.isnull().sum())

# Now you can use the columns as intended
X = iris[['petal_length', 'petal_width']].values
y = iris['species'].values

# Keep only Setosa and Versicolor for binary classification
mask = (y == 'Iris-setosa') | (y == 'Iris-versicolor')
X = X[mask]
y = y[mask]
```

First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

```
Getting to know the dataset:
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
(149, 5)
   sepal_length  sepal_width  petal_length  petal_width      species
0           4.9          3.0           1.4          0.2  Iris-setosa
1           4.7          3.2           1.3          0.2  Iris-setosa
2           4.6          3.1           1.5          0.2  Iris-setosa
3           5.0          3.6           1.4          0.2  Iris-setosa
4           5.4          3.9           1.7          0.4  Iris-setosa
       sepal_length  sepal_width  petal_length  petal_width
count    149.000000   149.000000    149.000000   149.000000
mean       5.848322     3.051007      3.774497     1.205369
std        0.828594     0.433499      1.759651     0.761292
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.400000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000

Check for null values:
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

Step 2: Feature and Label Selection

For binary classification, select only the two classes (Setosa and Versicolor):

```python
# Now you can use the columns as intended
X = iris[['petal_length', 'petal_width']].values
y = iris['species'].values

# Keep only Setosa and Versicolor for binary classification
mask = (y == 'Iris-setosa') | (y == 'Iris-versicolor')
X = X[mask]
y = y[mask]

# Encode the classes as integers: 0 for Setosa, 1 for Versicolor
y = np.where(y == 'Iris-setosa', 0, 1)
```

First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

Step 3: Splitting the Data

Why split into training and testing sets?

- To evaluate the model's ability to generalize to unseen data.
- Training data is used to fit the model; testing data is for evaluating its predictive performance.

```
# 2. Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    *arrays: X, y, test_size=0.3, random_state=1, stratify=y
)
```

Step 4: Feature Standardization

Why standardize?

- Perceptron and most ML algorithms are sensitive to the scale of input features.
- Standardization (zero mean, unit variance) ensures fair weight updates.

```
# 3. Standardize features
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

Training the Perceptron

The model is created and trained as follows:

```
# 4. Train Perceptron
ppn = Perceptron(eta0=0.1, random_state=1)
ppn.fit(X_train_std, y_train)
```

Evaluating the Model

After training, we make predictions and measure performance:

```
# 5. Predict and evaluate
y_pred = ppn.predict(X_test_std)
print(f"Misclassified samples: {(y_test != y_pred).sum()}")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
```

```
Misclassified samples: 0
Accuracy: 1.00
```

Interpretation:

- Misclassified samples: Number of test points incorrectly predicted.
- Accuracy: Proportion of correct predictions.

Visualizing the Decision Boundary

Understanding the graph:

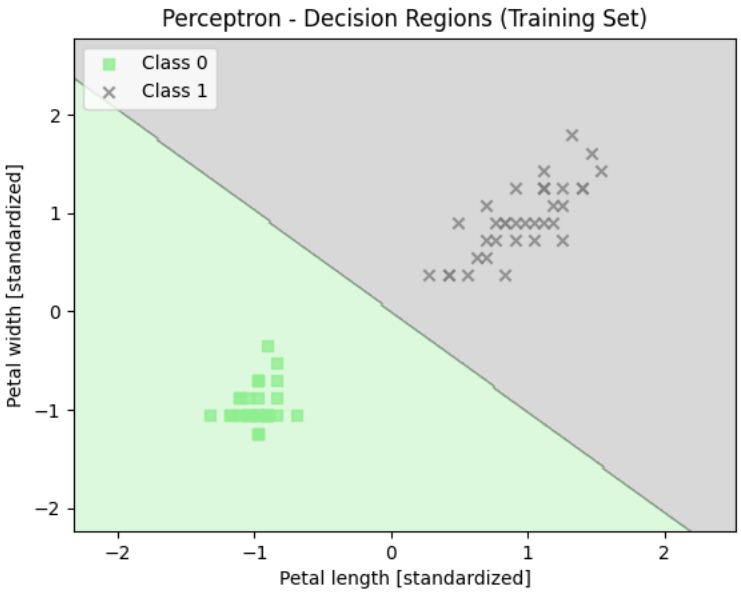
- Shows the decision surface learned by the Perceptron.
- Different colors represent different predicted classes.
- The boundary is a straight line, showing linear separability.

First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

```python
# 6. Plot decision regions
def plot_decision_regions(X, y, classifier, resolution=0.02):  1 usage
    markers = ('s', 'x')
    colors = ('lightgreen', 'gray')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf( *args: xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim( *args: xx1.min(), xx1.max())
    plt.ylim( *args: xx2.min(), xx2.max())
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=colors[idx], marker=markers[idx], label=f'Class {cl}')

plot_decision_regions(X_train_std, y_train, classifier=ppn)
plt.xlabel('Petal length [standardized]')
plt.ylabel('Petal width [standardized]')
plt.legend(loc='upper left')
plt.title('Perceptron - Decision Regions (Training Set)')
plt.show()
```



First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

Interpreting the Results

- If accuracy = 1.00 (100%) and misclassified samples = 0, the model perfectly learned the separation.
- The plot should show two clusters, each assigned a different class color, with a clear linear boundary.
- If there are any misclassifications, they will lie near or on the decision boundary.

Advantages and Limitations

Advantages:

- Fast and simple to implement.
- Effective for linearly separable data.
- Lays foundation for understanding neural networks.

Limitations:

- Only works for binary (two-class) tasks.
- Fails on non-linearly separable data.
- No probabilistic output.
- Sensitive to feature scaling and outliers.

How are these limitations solved?

- Logistic Regression (next topic) provides probability estimates.
- SVMs, Kernel methods, and Neural Networks handle non-linear boundaries and multiclass problems.

First Steps with scikit-learn: Training a Perceptron on the Iris Dataset

By: Shivesh Raj Sahu

Advanced Notes & Professional Tips

- Use cross-validation for more robust performance estimates.
- For larger/more complex datasets, explore pipelines and model serialization.
- Try multiclass problems (all three iris species) for further learning.
- Use more features and try other linear classifiers for comparison.

References

- Python Machine Learning by Sebastian Raschka (Textbook)
- scikit-learn documentation
- Iris dataset description
