211112238-ml-lab02

February 9, 2024

0.1 Gradient Descent

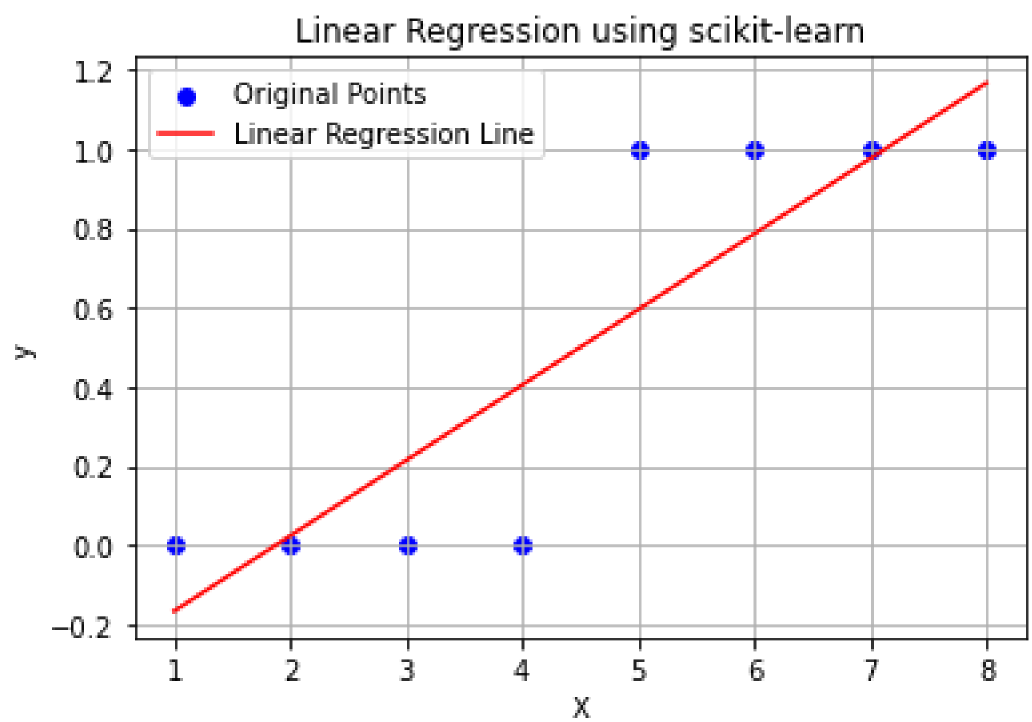
[ ]:
```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5, 6, 7, 8]
y = [0, 0, 0, 0, 1, 1, 1, 1]

plt.scatter(x, y, color='blue', marker='o')
plt.title('Scatter Plot of Points with Binary Class Labels')
plt.xlabel('x')
plt.ylabel('Class Label (y)
           ')
plt.yticks([0, 1], ['Class 0', 'Class 1'])
plt.grid(True)
plt.show()
```

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Logistic regression function with gradient descent
def gd(X, y):
    X = np.insert(X, 0, 1, axis=1)
    weights = np.ones(X.shape[1])
    lr = 0.5

    for _ in range(5000):
        y_hat = sigmoid(np.dot(X, weights))
        #
        weights = weights + lr * (np.dot((y - y_hat), X) / X.shape[0])

    return weights[1:], weights[0]

# Function to plot points
```

```
def plot_points(X, y, title="Scatter Plot"):
    plt.figure(figsize=(8, 6))
    class_0 = X[y == 0]
    class_1 = X[y == 1]
    plt.scatter(class_0, [0] * len(class_0), c='blue', label='Class 0',␣
↪marker='o')
    plt.scatter(class_1, [0] * len(class_1), c='red', label='Class 1',␣
↪marker='x')
    plt.title(title)
    plt.xlabel('X')
    plt.legend()
    plt.grid(True)

# Convert data to DataFrame
data = pd.DataFrame({'X': [1, 2, 3, 4, 5, 6, 7, 8], 'y': [0, 0, 0, 0, 1, 1, 1,␣
↪1]})

# Plot original points
plot_points(data['X'], data['y'], title="Scatter Plot of Points with Binary␣
↪Class Labels")

# Train logistic regression using gradient descent
weights, bias = gd(data['X'].values.reshape(-1, 1), data['y'].values)

# Plot logistic regression line
x_range = np.linspace(min(data['X']), max(data['X']), 100)
log_reg_line = sigmoid(weights * x_range + bias)

plt.plot(x_range, log_reg_line, color='green', label='Logistic Regression Line')
plt.legend()

plt.show()
```

[17]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate example data
X = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(-1, 1)  # Reshape to make it a␣
↪2D array(Column wise)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1])

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X, y)
```

```
# Get the slope (coefficient) and intercept
slope = model.coef_[0]
intercept = model.intercept_

# Plot the original points
plt.scatter(X, y, color='blue', marker='o', label='Original Points')

# Plot the linear regression line
x_range = np.linspace(min(X), max(X), 100)
y_pred = model.predict(x_range.reshape(-1, 1))
plt.plot(x_range, y_pred, color='red', label='Linear Regression Line')

plt.title('Linear Regression using scikit-learn')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```



0.2 Support Vector Machine

[19]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

[ ]:
```
import glob
import pandas as pd
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

# Function to perform SVM Regression with RBF kernel
def SVRRBFKernelMetrics(x_train, y_train, x_test, y_test, C, gamma):
    scaler = StandardScaler()
    x_train_scaled = scaler.fit_transform(x_train)
    x_test_scaled = scaler.transform(x_test)

    svr = SVR(kernel='rbf', C=C, gamma=gamma)
    svr.fit(x_train_scaled, y_train)

    y_pred = svr.predict(x_test_scaled)

    rmse = mean_squared_error(y_test, y_pred, squared=False)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    return rmse, mae, r2

folders = ["diabetes-5-fold", "machineCPU-5-fold", "mortgage-5-fold"]

for folder in folders:
    print(folder)

    # Define the file pattern
    file_pattern = "*tra.dat"
    training_files = glob.glob("./" + folder + "/" + file_pattern)
    file_pattern = "*tst.dat"
    testing_files = glob.glob("./" + folder + "/" + file_pattern)

    alpha_values = [2 ** i for i in range(0, 2, 2)]
    degree = [2, 3]

    for d in degree:
        svm_map = {}
```

```
        for C in alpha_values:
            for gamma in alpha_values:
                trmse = 0
                tmae = 0
                tr2 = 0

                for train_file, test_file in zip(training_files, testing_files):
                    df = pd.read_csv(train_file, delimiter=',', header=None,␣
↪comment='@')
                    df_test = pd.read_csv(test_file, delimiter=',',␣
↪header=None, comment='@')

                    x_train = df.iloc[:, :-1]
                    y_train = df.iloc[:, -1]
                    x_test = df_test.iloc[:, :-1]
                    y_test = df_test.iloc[:, -1]

                    rmse, mae, r2 = SVRRBFKernelMetrics(x_train, y_train,␣
↪x_test, y_test, C, gamma)

                    trmse += rmse
                    tmae += mae
                    tr2 += r2

                trmse /= 5
                tmae /= 5
                tr2 /= 5

                svm_map[(folder, d, C, gamma)] = (trmse, tmae, tr2)

        # Print or store the results as needed
        # for key, values in svm_map.items():
        #     print(f"{key}: RMSE={values[0]}, MAE={values[1]}, R2={values[2]}")
```

[ ]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Example true labels and predicted labels
y_true = [1, 2, 3, 4, 5, 6, 7, 8]
y_pred = [0, 0, 0, 0, 1, 1, 1, 1]

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Print the confusion matrix
```

```python
print("Confusion Matrix:")
print(cm)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=[0, 1, 2], yticklabels=[0, 1, 2], linewidths=.5)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

[ ]:

```python

```