logistic_regression

Efficiency of logistic regression with sample data from scikit-learn

Kunal Khurana

2024-05-20

Table of contents

Explanation and steps for logistic regression

Probabilities are utilized instead of specific values in this approach which is not the case for linear regression. Instead of mean square error, cross-entropy is employed.

The Gradient Descent method is applied for LogisticRegression as well.

Weight calculation involves subtracting the gradient from the current weight.

Steps: (i) Training - Initialize weight and bias as zero. (ii) Given a data point - predict result, calculate error, use gradient descent to determine new weight and bias, repeat n times. (iii) Testing - input values into the equation, select label based on probability.

The same equation as in linear regression is utilized, integrated into the sigmoid function.

model building

```
# creating a class LogisticRegression
import numpy as np

# Creating a sigmoid function as we'll be using it
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

class LogisticRegression:
    def __init__(self, lr=0.001, n_iters=1000):
        self.lr = lr
        self.n_iters = n_iters
        self.weights = None
        self.bias = None


    # always start by adding fit and predict funciton
    def fit(self, X, y):
        # Initializing weights and bias
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)  # assigning zeros as weights
        self.bias = 0

        # Gradient Descent
        for _ in range(self.n_iters):
            linear_pred = np.dot(X, self.weights) + self.bias
            predictions = sigmoid(linear_pred)

            # Gradient calculation
            dw = (1 / n_samples) * np.dot(X.T, (predictions - y))
            db = (1 / n_samples) * np.sum(predictions - y)

            # Update weights and bias
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        linear_pred = np.dot(X, self.weights) + self.bias
        y_pred = sigmoid(linear_pred)
        class_pred = [0 if i <= 0.5 else 1 for i in y_pred]
        return class_pred
```

testing

```
# testing how accurate it is with breast_cancer dataset from scikit_learn

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load data
bc = datasets.load_breast_cancer()
X, y = bc.data, bc.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234
```

```python
# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and fit the logistic regression model
clf = LogisticRegression(lr=0.01, n_iters=1000)
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

# Accuracy function
def accuracy(y_pred, y_test):
    accuracy = np.sum(y_pred == y_test) / len(y_test)
    return accuracy

# Calculate accuracy
acc = accuracy(y_pred, y_test)
print(f'Accuracy: {acc:.2f}')
```

```
Accuracy: 0.94
```