Assignment5_9628_jzeiders

December 1, 2024

1 John Zeiders (jzeiders) - Assignment 5

[ ]:

```
# Import required libraries
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix

# Set random seed for reproducibility
np.random.seed(42)
```

[ ]:

```
class PegasosSVM:
    def __init__(self, lambda_param=0.01, n_epochs=20, random_state=42):
        """
        Initialize Pegasos SVM classifier

        Args:
            lambda_param (float): Regularization parameter
            n_epochs (int): Number of epochs for training
            random_state (int): Random seed for reproducibility
        """
        self.lambda_param = lambda_param
        self.n_epochs = n_epochs
        self.random_state = random_state
        self.beta = None

    def _shuffle_data(self, X, y):
        """Shuffle the data while maintaining correspondence between X and y"""
        indices = np.arange(len(y))
        np.random.shuffle(indices)
        return X[indices], y[indices]

    def fit(self, X, y):
        """
        Fit the SVM model using the Pegasos algorithm

        Args:
            X (np.ndarray): Training features of shape (n_samples, n_features)
            y (np.ndarray): Training labels of shape (n_samples,)
        """
        np.random.seed(self.random_state)

        n_samples, n_features = X.shape

        # Initialize beta (including bias term)
        self.beta = np.zeros(n_features)

        t = 0  # Iteration counter

        # Training loop over epochs
        for epoch in range(self.n_epochs):
            # Shuffle data at the beginning of each epoch
            X_shuffled, y_shuffled = self._shuffle_data(X, y)

            # Loop over each training example
            for i in range(n_samples):
                t += 1
                eta_t = 1 / (self.lambda_param * t)  # Learning rate

                # Calculate the decision value
                decision_value = y_shuffled[i] * np.dot(X_shuffled[i], self.beta)

                # Check for margin violation
                if decision_value < 1:
                    # Update beta with both gradient and regularization
                    self.beta = (1 - eta_t * self.lambda_param) * self.beta + \
                              eta_t * y_shuffled[i] * X_shuffled[i]
                else:
                    # Update beta with only regularization
                    self.beta = (1 - eta_t * self.lambda_param) * self.beta

                # Projection step
                norm_beta = np.linalg.norm(self.beta)
                threshold = 1 / np.sqrt(self.lambda_param)
                if norm_beta > threshold:
                    self.beta = (threshold / norm_beta) * self.beta

            # Print progress
            if (epoch + 1) % 5 == 0 or epoch == 0:
                train_acc = np.mean(self.predict(X) == y)
                print(f"Epoch {epoch + 1}/{self.n_epochs}, Training Accuracy: {train_acc:.4f}")

    def predict(self, X):
        """
        Predict class labels for samples in X

        Args:
            X (np.ndarray): Features of shape (n_samples, n_features)

        Returns:
            np.ndarray: Predicted class labels (-1 or 1)
        """
        scores = np.dot(X, self.beta)
        return np.where(scores > 0, 1, -1)

    def evaluate(self, X, y):
        """
        Evaluate the model and print confusion matrix

        Args:
            X (np.ndarray): Features
            y (np.ndarray): True labels

        Returns:
            float: Classification error rate
        """
        y_pred = self.predict(X)
        conf_matrix = confusion_matrix(y, y_pred, labels=[-1, 1])
        error_rate = np.mean(y_pred != y)

        print("\nConfusion Matrix:")
        print("Predicted")
        print("     -1    1")
        print("-1", conf_matrix[0])
        print(" 1", conf_matrix[1])
        print(f"\nError Rate: {error_rate:.4f}")

        return error_rate
```

[ ]:

```
def load_and_preprocess_data(filepath):
    """
    Load and preprocess the data

    Args:
        filepath (str): Path to the CSV file

    Returns:
        tuple: Preprocessed features and labels
    """
    data = pd.read_csv(filepath)
    X = data.iloc[:, :-1].values  # All columns except the last

    # Add bias term
    X = np.hstack((X, np.ones((X.shape[0], 1))))

    # Convert labels
    y = data.iloc[:, -1].values
    y[y == 5] = -1  # Convert 5 to -1
    y[y == 6] = 1   # Convert 6 to 1

    # Validate labels
    unique_labels = np.unique(y)
    if set(unique_labels) != {-1, 1}:
        raise ValueError(f"Labels must be -1 and 1, but found {unique_labels}")

    return X, y
```

[ ]:

```
# URLs to datasets
train_url = 'https://liangfgithub.github.io/Data/coding5_train.csv'
test_url = 'https://liangfgithub.github.io/Data/coding5_test.csv'

# Load and preprocess data
print("Loading and preprocessing data...")
X_train, y_train = load_and_preprocess_data(train_url)
X_test, y_test = load_and_preprocess_data(test_url)

# Initialize and train model
svm = PegasosSVM(lambda_param=0.01, n_epochs=20, random_state=42)

print("\nTraining SVM...")
svm.fit(X_train, y_train)

# Evaluate model
print("\nTraining Set Evaluation:")
train_error = svm.evaluate(X_train, y_train)

print("\nTest Set Evaluation:")
test_error = svm.evaluate(X_test, y_test)
```

```
Loading and preprocessing data…

Training SVM…
Epoch 1/20, Training Accuracy: 0.7100
Epoch 5/20, Training Accuracy: 0.9700
Epoch 10/20, Training Accuracy: 1.0000
Epoch 15/20, Training Accuracy: 1.0000
Epoch 20/20, Training Accuracy: 1.0000
```

```
Training Set Evaluation:

Confusion Matrix:
Predicted
     -1    1
-1 [100   0]
 1 [  0 100]

Error Rate: 0.0000

Test Set Evaluation:

Confusion Matrix:
Predicted
     -1    1
-1 [280  20]
 1 [  6 294]

Error Rate: 0.0433
```