svm.py

March 28, 2025

[ ]:
```python
"""Support Vector Machine (SVM) model."""

# This source code is modified by Arman Sayan.

# Last Edit: March 28, 2024

import numpy as np


class SVM:
    def __init__(self, n_class: int, lr: float, epochs: int, reg_const: float):
        """Initialize a new classifier.

        Parameters:
            n_class: the number of classes
            lr: the learning rate
            epochs: the number of epochs to train for
            reg_const: the regularization constant
        """
        self.w = None  # Weight matrix of shape (D, C), initialized during␣
 ↪training
        self.alpha = lr
        self.epochs = epochs
        self.reg_const = reg_const
        self.n_class = n_class

    def calc_gradient(self, X_train: np.ndarray, y_train: np.ndarray) -> np.
 ↪ndarray:
        """Calculate gradient of the svm hinge loss.

        Inputs have dimension D, there are C classes, and we operate on
        mini-batches of N examples.

        Parameters:
            X_train: a numpy array of shape (N, D) containing a mini-batch
                of data
            y_train: a numpy array of shape (N,) containing training labels;
```

```
            y[i] = c means that X[i] has label c, where 0 <= c < C

        Returns:
            the gradient with respect to weights w; an array of the same shape
                as w
        """
        N, D = X_train.shape

        # Compute class scores for all samples
        scores = X_train @ self.w  # Shape (N, C)

        # Extract the scores of the correct classes
        correct_class_scores = scores[np.arange(N), y_train].reshape(-1, 1)

        # Compute the margins for all classes
        margins = np.maximum(0, scores - correct_class_scores + 1)

        # Zero-out the margins for the correct classes
        margins[np.arange(N), y_train] = 0  # Ignore correct class

        # Binary indicator: 1 where margin > 0
        indicator = (margins > 0).astype(float)  # Indicator for incorrect␣
↪classes

        # For each example, subtract total count of violations from the correct␣
↪class column
        indicator[np.arange(N), y_train] = -np.sum(indicator, axis=1)  # Adjust␣
↪correct class

        # Compute gradient and add L2 regularization
        grad = (X_train.T @ indicator) / N + self.reg_const * self.w  # Add␣
↪regularization
        return grad

    def train(self, X_train: np.ndarray, y_train: np.ndarray):
        """Train the classifier.

        Hint: operate on mini-batches of data for SGD.

        Parameters:
            X_train: a numpy array of shape (N, D) containing training data;
                N examples with D dimensions
            y_train: a numpy array of shape (N,) containing training labels
        """
        N, D = X_train.shape

        # Initialize weights randomly if not already initialized
```

```python
        if self.w is None:
            self.w = np.random.randn(D, self.n_class) * 0.01  # Initialize small random weights

        # Perform gradient descent for a number of epochs
        for epoch in range(self.epochs):
            # Compute the gradient of the current loss
            gradient = self.calc_gradient(X_train, y_train)

            # Update weights using the gradient
            self.w -= self.alpha * gradient  # Update weights

            # Every 100 epochs, compute and print the average hinge loss
            if epoch % 100 == 0:
                scores = X_train @ self.w  # (N, C)
                correct_class_scores = scores[np.arange(N), y_train].reshape(-1, 1)  # (N, 1)
                margins = np.maximum(0, scores - correct_class_scores + 1)  # (N, C)
                margins[np.arange(N), y_train] = 0  # Zero out correct class
                loss = np.mean(np.sum(margins, axis=1))  # Hinge loss averaged over batch
                print(f"Epoch {epoch}: Loss = {loss:.4f}")


    def predict(self, X_test: np.ndarray) -> np.ndarray:
        """Use the trained weights to predict labels for test data points.

        Parameters:
            X_test: a numpy array of shape (N, D) containing testing data;
                N examples with D dimensions

        Returns:
            predicted labels for the data in X_test; a 1-dimensional array of
                length N, where each element is an integer giving the predicted
                class.
        """
        # Compute class scores and return the index of the highest score (best class)
        return np.argmax(X_test @ self.w, axis=1)
```