Programming Assignment-6 (PH227): Support Vector Machines (SVMs) Classifier

Objective

This assignment aims to provide hands-on experience in implementing and understanding the Support Vector Machine (SVM) algorithm. You will build an SVM classifier for both linearly and non-linearly separable datasets, and visualize the decision boundaries and margins learned by the model.

Part 1: Linear SVM Classifier

Problem Description

In this section, you will implement a Linear SVM classifier from scratch. You will work with a 2D linearly separable dataset and visualize the learned decision boundary and margin.

Instructions

Complete the following steps for implementing and evaluating the Linear SVM classifier:

1. **Dataset Loading:**
    - Load 2D dataset (Dataset-1) with two classes that are linearly separable.
2. **Initial Visualization:**
    - Create a scatter plot of the Dataset-1, with data points color-coded as per their respective classes.
    - Ensure the plot is well-labeled with appropriate axis titles (e.g., "Feature 1," "Feature 2") and a descriptive title (e.g., "Linearly Separable Dataset-1").
    -
2. **Data Splitting:**
    - Split the dataset into training and testing sets (e.g., 80% training, 20% testing).
3. **SVM Algorithm Implementation (from scratch):**
    - Implement the core components of a Linear SVM classifier. Your implementation should include:

        - **Initialization:** Set learning rate, and number of iterations.
        - **Weight and Bias Initialization:** Initialize weights `w` and bias `b` to small random values.
        - **Cost Function:** Implement the hinge loss function.
          (Hinge loss is a loss function used in machine learning for training SVM classifiers
        - **Gradient Descent:** Implement the gradient descent algorithm to update `w` and `b`. The update rules should differentiate between correctly classified points within the margin, misclassified points, and correctly classified points outside the margin.
        - **Fit Method:** A `fit` method that iteratively updates `w` and `b` using gradient descent on the training data.
        - **Predict Method:** A `predict` method that classifies new data points based on the learned `w` and `b`.
4. **Model Training:**
    - Train your custom Linear SVM model on the training data.
5. **Visualization:**
    - Plot the training data points, color-coded by class.
    - Visualize the decision boundary (hyperplane) learned by your Linear SVM model.
    - Draw the margins, which are the lines parallel to the decision boundary and pass through the support vectors.
    - Ensure axes are labeled and the plot has a title.
6. **Model Testing:**
    - Evaluate your trained Linear SVM model on the testing data.
    - Calculate and report the accuracy of the model on both training and testing sets.

Part 2: Non-linear SVM Classifier

Problem Description

In this section, you will extend your SVM implementation to handle non-linearly separable data using a kernel trick. You will work with a 2D non-linearly separable dataset (circular boundary example) and visualize the decision boundary after projection.

Instructions

Complete the following steps for implementing and evaluating the Non-linear SVM classifier:

1. **Dataset Loading:**
   - Load 2D dataset (Dataset-2) with two classes that are non-linearly separable.
2. **Initial Visualization:**
   - Create a scatter plot of the Dataset-2, with data points color-coded according to their respective classes. This visualization should clearly illustrate the nonlinear separability of the data.
   - Ensure the plot is well-labeled with appropriate axis titles (e.g., "Feature 1," "Feature 2") and a descriptive title (e.g., "Non-linearly Separable Dataset-2").
3. **Data Splitting:**
   - Split the dataset into training and testing sets (e.g., 80% training, 20% testing).
4. **Feature Transformation (Kernel Trick Concept):**
   - To handle non-linear separability, project the 2D data into a higher-dimensional space (e.g., 3D). For circular data, you could use a transformation like `(x, y) -> (x, y, x^2 + y^2)`.
   - Implement this feature transformation using function.
5. **SVM Algorithm Implementation (from scratch - reuse linear SVM core):**
   - Adapt your Linear SVM implementation from Part 1 to work with the transformed (higher-dimensional) data. The core SVM algorithm (cost function, gradient descent) remains the same, but it now operates on the new feature space.
6. **Model Training:**
   - Train your custom SVM model on the transformed training data.
7. **Visualization:**
   - **3D Visualization:** For the transformed data, create a 3D scatter plot showing the data points, color-coded by class.
   - **Decision Plane:** Visualize the decision plane learned by your SVM in this 3D space.
   - **Margin in 3D:** Draw the margins around the decision plane.
   - **2D Decision Boundary:** Project the 3D decision plane back into the original 2D space to visualize the non-linear decision boundary as a curve.
   - Ensure axes are labeled and the plot has a title.
8. **Model Testing and Evaluation:**
   - Evaluate your trained Non-linear SVM model on the testing data (after applying the same feature transformation to the test set).

    - Calculate and report the accuracy of the model on both transformed training and testing sets.

Submission Guidelines

- Submit a single Jupyter Notebook (`.ipynb`) file containing all your code.
- Clearly separate Part 1 and Part 2 with markdown headings.
- Ensure all code blocks are executed and produce the expected outputs.
- Add comments to explain your code, especially for custom SVM implementations and complex logic.
- Ensure your code is well-structured and easy to read.
- For all plots, include appropriate titles, axis labels, and legends where necessary.

Evaluation Criteria

Your assignment will be evaluated based on the following:

- **Correctness:** Proper implementation of the Linear and Non-linear SVM algorithms from scratch.
- **Functionality:** The code runs without errors and produces expected outputs.
- **Visualization Quality:** Clarity and effectiveness of generated plots (decision boundaries, margins in both 2D and 3D).
- **Code Quality:** Readability, comments, and adherence to Python best practices.
- **Understanding:** Demonstrated understanding of the underlying concepts of SVM, including the hinge loss, gradient descent, and the kernel trick.

Due Date

Please submit your completed Jupyter Notebook by Nov 5, 2025 11:59 PM.