Support Vector Machine (SVM) Model for Titanic Survival Prediction

NAME: Ankita Satapathy
ROLL NO: 22053400

1. Introduction

This project implements a Support Vector Machine (SVM) model to predict passenger survival on the Titanic. The model is built using Python and the scikit-learn library, with a custom implementation of SVM. The application provides an API using FastAPI and a front-end interface for user interaction.

2. Files in the Submission

i. SVM3400.ipynb: Implements the SVM model using a custom approach in Jupyter Notebook and contains the model training process.
ii. SVM3400.py: Implements the model in a Python script for backend integration. Generated in vscode code terminal using command to facilitate backend integration.

```
PS C:\Users\KIIT\Desktop\22053400\5) SVM> jupyter nbconvert --to script SVM3400.ipynb
```

iii. main.py: Defines the FastAPI server to handle prediction requests.
iv. svm_titanic_model.pkl: The trained model serialized using pickle.
v. index.html: Front-end UI for user input and displaying predictions.

3. Installation and Setup

i. Prerequisites

The following Python packages are required:

- NumPy (numpy): Used for numerical computations and matrix operations in model training.
- Pandas (pandas): Used for loading and preprocessing the Titanic dataset.
- Scikit-learn (sklearn): Provides ML utilities such as data preprocessing and model evaluation.
- FastAPI (fastapi): Used to create a web API for serving predictions.
- Pickle (pickle): Used to save and load the trained model for reuse.
- Uvicorn (uvicorn): Used to run the FastAPI server asynchronously.

ii. Installing Dependencies

Run the following command in the terminal to install dependencies:

```
PS C:\Users\KIIT\Desktop\22053400\5) SVM> pip install fastapi uvicorn numpy pandas scikit-learn
```

4. Model Implementation

The CustomSVM class in SVM3400.py implements a Support Vector Machine using gradient descent. The model learns a decision boundary for classifying passengers as "Survived" or "Not Survived."

```python
class CustomSVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, epochs=1000):
        self.learning_rate = learning_rate
        self.lambda_param = lambda_param
        self.epochs = epochs
        self.w = None
        self.b = None
    def fit(self, X, y):
        """Train the SVM model using gradient descent"""
        n_samples, n_features = X.shape
        y_transformed = np.where(y == 0, -1, 1)
        self.w = np.zeros(n_features)
        self.b = 0
        for _ in range(self.epochs):
            for idx, x_i in enumerate(X):
                condition = y_transformed[idx] * (np.dot(x_i, self.w) - self.b) >= 1
                if condition:
                    self.w -= self.learning_rate * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.learning_rate * (2 * self.lambda_param * self.w - np.dot(x_i, y_transformed[idx]))
                    self.b -= self.learning_rate * y_transformed[idx]
    def predict(self, X):
        """Predict class labels"""
        predictions = np.dot(X, self.w) - self.b
        return np.where(predictions >= 0, 1, 0)
```

5. Training the Model

- The dataset used is the Titanic Dataset from DataScienceDojo.
- Relevant features include Pclass, Age, SibSp, Parch, and Fare, while the target variable is Survived.
- Data is preprocessed, including handling missing values and normalizing features.
- The SVM model is trained using gradient descent with regularization.
- The trained model and scaler are saved using pickle

6. API Implementation

The FastAPI-based backend (main.py) loads the trained model and provides an endpoint to make predictions:

i. API Setup

```python
app = FastAPI()
```

ii. CORS Configuration- To allow front-end requests:

```python
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

iii. Loading the Trained Model

```python
with open("svm_titanic_model.pkl", "rb") as f:
    data = pickle.load(f)
    model = data["model"]
    scaler = data["scaler"]
```

iv. Defining the API Endpoint

```python
@app.post("/predict/")
async def predict(data: InputData):
    features = np.array(data.features).reshape(1, -1)
    features_scaled = scaler.transform(features)
    prediction = model.predict(features_scaled)[0]
    class_name = "Survived" if prediction == 1 else "Not Survived"
    return {"prediction": class_name}
```

7. Running the Application (FastAPI Server)

To start the API server, run the following command:

```
PS C:\Users\KIIT\Desktop\22053400\5) SVM> uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\KIIT\\Desktop\\22053400\\5) SVM']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [19996] using WatchFiles
```

8. Front-End Implementation

The index.html file provides a simple UI for users to input passenger details and get predictions using the API. It has inline CSS and JavaScript. The JavaScript function sends a request to the FastAPI backend:

```javascript
async function predict() {
    let features = [];
    let featureIds = ["feature0", "feature1", "feature2", "feature3", "feature4"];
    for (let id of featureIds) {
        let value = document.getElementById(id).value;
        if (value === "" || isNaN(value)) {
            document.getElementById("result").innerText = "⚠️ Please enter valid numbers!";
            return;
        }
        features.push(parseFloat(value));
    }
    try {
        let response = await fetch("http://127.0.0.1:8000/predict/", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ features: features })
        });
        let data = await response.json();
        let survivalStatus = data.prediction === 1 ? "🎉 Survived!" : "❌ Did Not Survive.";
        document.getElementById("result").innerText = "🚢 Prediction: " + survivalStatus;
    } catch (error) {
        document.getElementById("result").innerText = "❌ Error: " + error.message;
    }
}
```

9. Conclusion

This project successfully implements an SVM model to predict passenger survival on the Titanic. The model is integrated with a FastAPI backend and utilizes NumPy for numerical computations and Pandas for data preprocessing. The trained model is saved using Pickle and deployed via Uvicorn for real-time predictions.