SVM Model Integrated with Website using FastAPI

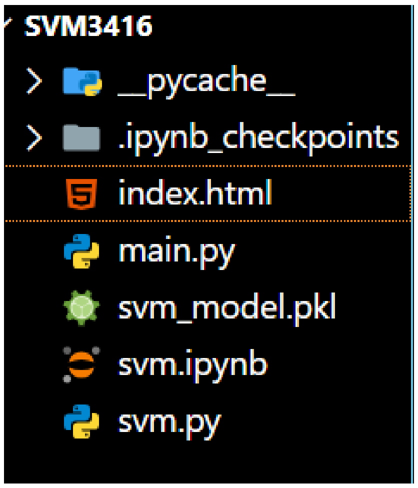
Name- Barsha Baibhabi
Roll No - 22053416

1. Overview

This project implements a Support Vector Machine (SVM) model for wine classification using the Wine dataset from `sklearn.datasets`. The dataset consists of 13 numerical features that help classify wines into three categories (0, 1, or 2). The trained model is deployed using FastAPI, and a simple HTML frontend allows users to input feature values and get predictions in real-time.

Files in the Project:

1. **`svm_model.ipynb`** – Trains the model and saves it as a pickle file.
2. **`main.py`** – FastAPI backend for handling predictions.
3. **`index.html`** – Web frontend for user input and displaying results.
4. **svm.py** - Python script version of the Jupyter Notebook.
5. **Svm_model.pk**l - Saved logistic regression models.



```
SVM3416
> __pycache__
> .ipynb_checkpoints
  index.html
  main.py
  svm_model.pkl
  svm.ipynb
  svm.py
```

2. Installation & Setup

Prerequisites

Ensure you have Python 3 installed. Install the required dependencies:

```
pip install fastapi uvicorn scikit-learn numpy pandas
```

Running the Application

1. Train the model and generate a pickle file by running `svm_model.ipynb`.
2. Start the FastAPI backend:

```
uvicorn main:app --host 127.0.0.1 --port 8005 --reload
```

3. Open `index.html` in a browser and test predictions.

3. Training the Model & Generating Pickle File

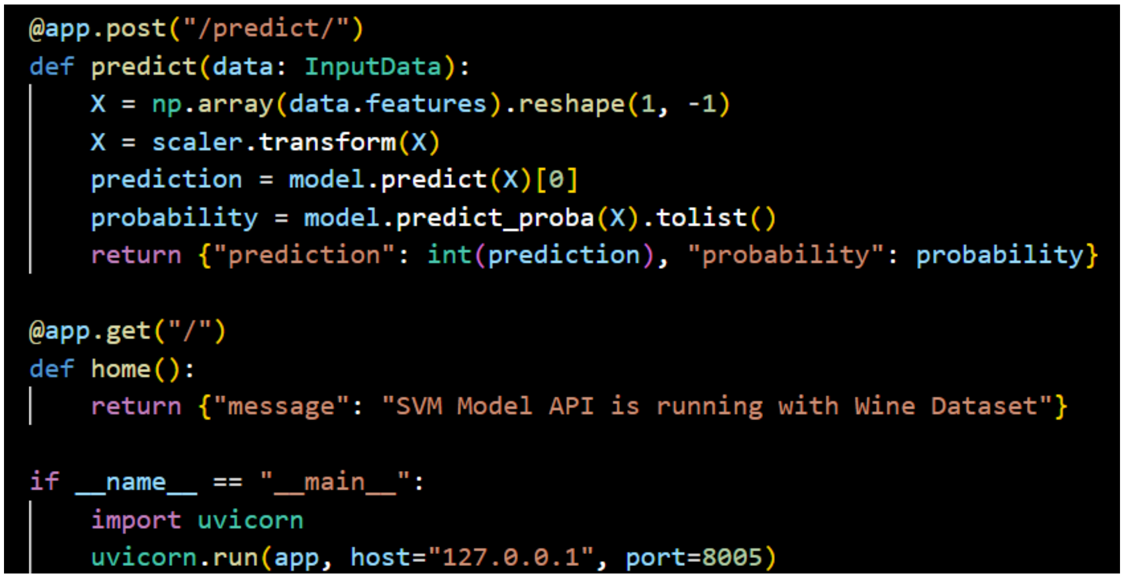
The `svm_model.ipynb` notebook does the following:

- Loads the Wine dataset from `sklearn.datasets`.
- Splits the data into training and test sets.
- Scales the features using `StandardScaler()`.
- Trains an SVM classifier (SVC) with a linear kernel.
- Saves the trained model and scaler as a pickle file (`svm_model.pkl`).

4. FastAPI Backend (`main.py`)

The backend is implemented using **FastAPI** to:

- Load the **trained model** and **scaler** from `svm_model.pkl`.
- Accept **feature inputs** from the frontend via a POST request.



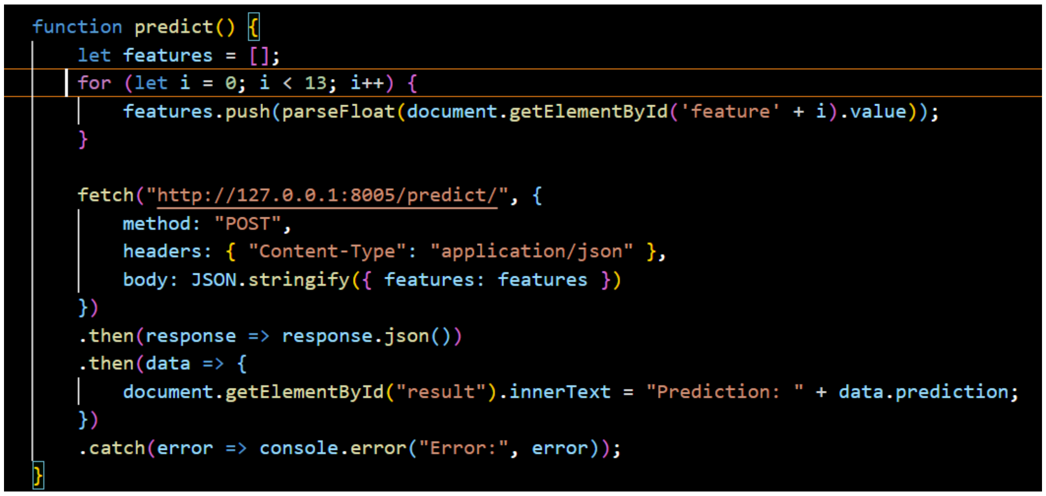
```python
@app.post("/predict/")
def predict(data: InputData):
    X = np.array(data.features).reshape(1, -1)
    X = scaler.transform(X)
    prediction = model.predict(X)[0]
    probability = model.predict_proba(X).tolist()
    return {"prediction": int(prediction), "probability": probability}

@app.get("/")
def home():
    return {"message": "SVM Model API is running with Wine Dataset"}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8005)
```

- Scale input features and return the **predicted wine class** (0, 1, or 2) along with probabilities.
- Runs on `http://127.0.0.1:8005/`

5. Frontend (`index.html`) with Inline CSS & JavaScript

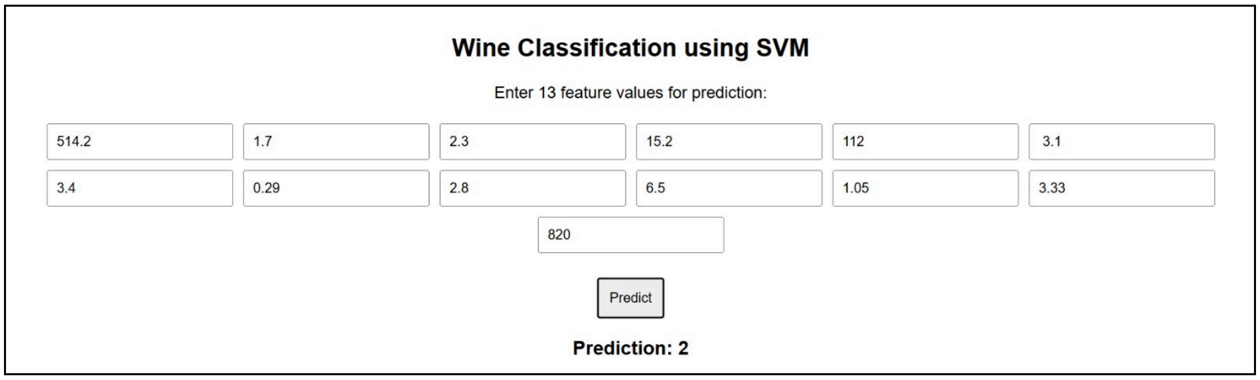
The frontend:

- Provides an input form with 13 fields for feature values.
- Uses JavaScript to send AJAX requests to the FastAPI backend.



```
function predict() {
    let features = [];
    for (let i = 0; i < 13; i++) {
        features.push(parseFloat(document.getElementById('feature' + i).value));
    }

    fetch("http://127.0.0.1:8005/predict/", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ features: features })
    })
    .then(response => response.json())
    .then(data => {
        document.getElementById("result").innerText = "Prediction: " + data.prediction;
    })
    .catch(error => console.error("Error:", error));
}
```

- Displays the predicted class on the webpage.



Wine Classification using SVM

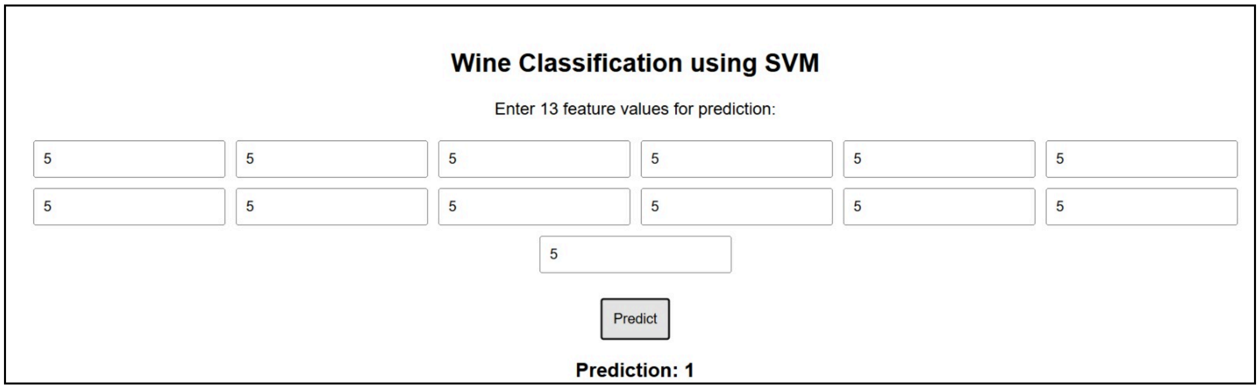
Enter 13 feature values for prediction:

514.2	1.7	2.3	15.2	112	3.1
3.4	0.29	2.8	6.5	1.05	3.33
820					

Prediction: 2

6. Testing the Integration

- Enter 13 numerical feature values in `index.html`.
- Click Predict to send a request to `FastAPI`.


Wine Classification using SVM
Enter 13 feature values for prediction:
5
5
5
5
5
5
5
5
5
5
5
5
5
Predict

Prediction: 1

- The API returns the predicted class with probabilities.
- The result is displayed on the webpage.

---

7. Conclusion

This project successfully integrates an SVM classification model with a FastAPI backend and a simple web frontend. It provides a foundation for further enhancements, such as improved UI, database integration, and additional model tuning.