Coinbase SVM Bitcoin Classifier

Can you generate Python code which fetches the historical candlesticks data from Coinbase's API using Requests and run a Support Vector Machine, imported from Scikit-Learn, on the data to classify if we should Buy or Sell Bitcoin.

Below is the Python code to fetch historical candlestick data from Coinbase's API using `**requests**` and then use a Support Vector Machine (SVM) from Scikit-Learn to classify whether to buy or sell Bitcoin. The classification is based on simple technical indicators like moving averages and price returns.

Make sure you have the necessary libraries installed (`**requests**`, `**pandas**`, `**scikit-learn**`, `**numpy**`, etc.) before running the code.

```
python
```

```python
import requests
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Fetch historical candlestick data from Coinbase
def fetch_candlesticks(product_id='BTC-USD', granularity=3600, limit=300):
    url = f'https://api.exchange.coinbase.com/products/{product_id}/candles'
    params = {'granularity': granularity, 'limit': limit}
    response = requests.get(url, params=params)
    response.raise_for_status()  # Raise an error for bad responses
    data = response.json()
    columns = ['time', 'low', 'high', 'open', 'close', 'volume']
    df = pd.DataFrame(data, columns=columns)
    df['time'] = pd.to_datetime(df['time'], unit='s')  # Convert to datetime
    df.sort_values('time', inplace=True)  # Sort by time ascending
    return df

# Generate features for the model
def generate_features(data):
    data['return'] = data['close'].pct_change()  # Price returns
    data['ma5'] = data['close'].rolling(window=5).mean()  # 5-period moving average
    data['ma20'] = data['close'].rolling(window=20).mean()  # 20-period moving average
    data['signal'] = np.where(data['return'] > 0, 1, 0)  # Binary labels (1: Buy, 0: Sell)
    data.dropna(inplace=True)  # Drop rows with NaN values
    return data

# Main script
def main():
    # Fetch and process data
    df = fetch_candlesticks()
    df = generate_features(df)

    # Prepare features and labels
    X = df[['return', 'ma5', 'ma20']].values
    y = df['signal'].values

    # Split into train/test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
2)
```

```
    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Train SVM classifier
    model = SVC(kernel='rbf', gamma='scale')
    model.fit(X_train, y_train)

    # Evaluate the model
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))

    # Example prediction
    last_row = X_test[-1].reshape(1, -1)
    prediction = model.predict(last_row)[0]
    action = 'Buy' if prediction == 1 else 'Sell'
    print(f"The model suggests to: {action}")

if __name__ == '__main__':
    main()
```

Explanation:

1. **Fetching Data**: The script fetches historical candlestick data (OHLCV) for Bitcoin using Coinbase's public API.
2. **Feature Engineering**: It computes simple features like returns and moving averages.
3. **Target Variable**: The target (`signal`) is a binary label indicating "Buy" (1) if the price return is positive and "Sell" (0) otherwise.
4. **SVM Classifier**: The model uses a radial basis function (RBF) kernel to classify the data.
5. **Evaluation**: The script splits the data into training and testing sets, scales the features, and prints a classification report for performance evaluation.

Let me know if you'd like to expand this with additional features or optimizations!