Agile Price Prediction: Real-time Learning and Forecasting

Anand Prakash, Asif Rahaman

21/04/2024

1 Introduction

This report presents a detailed overview of the Price Prediction System project. The system aims to predict prices based on various input features such as demand, cost, market size, economic conditions, and competition levels.

2 Dataset Description

The dataset used for training and testing the price prediction model consists of over 50,000 rows. Each row contains information about demand, cost, price, recession status, economy strength, competition level, and market size. It's important to note that this dataset was created by the author specifically for this project.

Listing 1: Sample Dataset

```
Demand,Cost,Price,Recession,Economy,Competition,Market Size
786,459,492.7059,True,Weak,High,1300
3784,2486,8825.5486,True,Medium,Low,3502
3765,1038,4943.6134,False,Weak,Low,7885
```

3 Data Preprocessing

The dataset undergoes preprocessing steps before being used for model training. Categorical variables like "Recession," "Economy," and "Competition" are encoded using LabelEncoder, while numerical variables are scaled using StandardScaler.

Listing 2: Data Preprocessing

```
# Encode categorical variables
encoder = LabelEncoder()
data["Recession"] = encoder.fit_transform(data["Recession"])
data["Economy"] = encoder.fit_transform(data["Economy"])
```

```python
data["Competition"] = encoder.fit_transform(data["Competition"])

# Scale numerical variables
scaler = StandardScaler()
data[["Demand", "Cost", "Market Size"]] = scaler.fit_transform(
    data[["Demand", "Cost", "Market Size"]]
)
```

4 Model Architecture

The Price Prediction model is implemented using PyTorch. It consists of multiple layers including fully connected layers, batch normalization layers, dropout layers, and activation functions like ReLU.

Listing 3: Price Predictor Model Architecture

```python
class PricePredictor(nn.Module):
    def __init__(self, input_dim):
        super(PricePredictor, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.bn1 = nn.BatchNorm1d(64)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(64, 128)
        self.bn2 = nn.BatchNorm1d(128)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(128, 64)
        self.bn3 = nn.BatchNorm1d(64)
        self.dropout3 = nn.Dropout(0.5)
        self.fc4 = nn.Linear(64, 32)
        self.bn4 = nn.BatchNorm1d(32)
        self.dropout4 = nn.Dropout(0.5)
        self.fc5 = nn.Linear(32, 1)

    def forward(self, x):
        x = F.relu(self.bn1(self.fc1(x)))
        x = self.dropout1(x)
        x = F.relu(self.bn2(self.fc2(x)))
        x = self.dropout2(x)
        x = F.relu(self.bn3(self.fc3(x)))
        x = self.dropout3(x)
        x = F.relu(self.bn4(self.fc4(x)))
        x = self.dropout4(x)
        x = self.fc5(x)
        return x
```

5 Kafka Integration

Kafka is utilized in the Price Prediction System for real-time data streaming and processing. The integration involves two main components: the Kafka Producer and the Kafka Consumer.

5.1 Kafka Producer

The Kafka Producer is responsible for sending data (in this case, request data) to a Kafka topic named "web-logs." The key-value pairs representing the request are serialized to JSON format before being sent to Kafka.

```
from kafka import KafkaProducer
import json

class KafkaHandler:
    def __init__(self, kafka_bootstrap_servers):
        self.kafka_bootstrap_servers = kafka_bootstrap_servers
        self.producer = KafkaProducer(bootstrap_servers=self.
            kafka_bootstrap_servers)

    def send_to_kafka(self, key, value):
        value = json.dumps(value)
        self.producer.send("web-logs", key=key.encode(), value=value.
            encode())
```

In this code:

- `KafkaHandler` initializes a Kafka Producer with the specified Kafka bootstrap servers.
- `send_to_kafka` method serializes the key-value pair to JSON format and sends it to the "web-logs" Kafka topic.

5.2 Kafka Consumer

The Kafka Consumer listens to the "web-logs" Kafka topic for incoming messages. It processes these messages, deserializes them from JSON format, and performs actions based on the received data.

```
from confluent_kafka import Consumer, KafkaError
import json

class TrafficProcessingSDK:
    def __init__(self, kafka_bootstrap_servers, group_id):
        self.kafka_handler = KafkaHandler(kafka_bootstrap_servers)
        self.group_id = group_id

    def consume_from_kafka(self):
        consumer = Consumer({
```

```
        "bootstrap.servers": self.kafka_handler.
            kafka_bootstrap_servers,
        "group.id": self.group_id,
        "auto.offset.reset": "earliest",
    })
    consumer.subscribe(["web-logs"])

    def msg_callback(msg):
        msg_str = msg.value().decode("utf-8")
        msg_dict = json.loads(msg_str)

        # Perform actions based on the received data
        print("Received␣message:␣{}".format(msg.value().decode("utf-8"
            )))
        total_rows = count_rows_in_csv()
        if total_rows % 5 == 0:
            train_model()

    try:
        while True:
            msg = consumer.poll(timeout=1.0)
            if msg is None:
                continue
            if msg.error():
                if msg.error().code() == KafkaError._PARTITION_EOF:
                    continue
                else:
                    print(msg.error())
                    break

            msg_callback(msg)
    finally:
        consumer.close()
```

In this code:

- `TrafficProcessingSDK` initializes a Kafka Consumer with the specified Kafka bootstrap servers and group ID.
- `consume_from_kafka` method subscribes to the "web-logs" Kafka topic, receives messages, and processes them using the `msg_callback` function.
- `msg_callback` function deserializes the received message from JSON format and performs actions based on the data, such as printing the message and triggering a model training process if certain conditions are met.

This Kafka integration allows the Price Prediction System to handle real-time data from incoming requests and perform necessary actions based on that data flow.

6 Conclusion

The Price Prediction System demonstrates the integration of machine learning models, real-time data processing through Kafka, and backend systems for serving predictions and handling data flow.

7 References

1. PyTorch Documentation: `https://pytorch.org/docs/stable/index.html`
2. Kafka Documentation: `https://kafka.apache.org/documentation/`
3. FastAPI Documentation: `https://fastapi.tiangolo.com/`