Министерство образования и науки РФ
Федеральное государственное автономное образовательное учреждение высшего образования
«Омский государственный технический университет»

Кафедра «Автоматизированные системы обработки информации и управления»

Отчёт по лабораторной работе №1
по дисциплине
«Современные инструментальные средства разработки программного обеспечения»

Выполнил:
Студент гр. ПИН-201
Негреев-Дашков З.Д.

Проверил:
Старший преподователь
Кабанов А.А.

Омск, 2024

Автоматизация обработки статистических данных о работе сервисов электронной коммерции

Требования к приложению

Краткое описание требований для приложения по анализу электронной коммерции:

1. Аутентификация и авторизация пользователей:
   - Пользователи должны иметь безопасные учетные данные.
   - Различные роли (администратор, аналитик) с различными уровнями доступа.
2. Импорт и интеграция данных:
   - Возможность импорта данных с различных платформ электронной коммерции.
   - Интеграция с базами данных или хранилищами данных для хранения информации.
3. Очистка и преобразование данных:
   - Инструменты для очистки и преобразования необработанных данных электронной коммерции.
   - Обработка отсутствующих значений и обеспечение согласованности данных.
4. Анализ продаж:
   - Отслеживание и анализ данных о продажах.
   - Создание отчетов о продажах товаров, дохода и трендах.
5. Анализ поведения клиентов:
   - Анализ взаимодействия и поведения клиентов на платформе.
   - Выявление паттернов и предпочтений.
6. Управление запасами:
   - Мониторинг и анализ уровней инвентаря.
   - Уведомление при низких уровнях запасов.
7. Визуализация данных:
   - Создание интерактивных и информативных панелей.
   - Визуальное представление ключевых показателей эффективности (KPI).
8. Мониторинг производительности:
   - Мониторинг производительности веб-сайта.
   - Анализ времени загрузки страниц и взаимодействия с пользователями.
9. Безопасность и соответствие:
   - Обеспечение безопасности данных и соответствие соответствующим регуляциям.

- Шифрование чувствительной информации.

10. Анализ в реальном времени:
    - Предоставление аналитики в реальном времени для мгновенных инсайтов.
    - Мониторинг живых потоков данных для быстрого принятия решений.
11. Настроенные отчеты:
    - Возможность пользователя создавать настраиваемые отчеты.
    - Экспорт отчетов в различные форматы (PDF, Excel).
12. Прогнозирование и предсказательный анализ:
    - Реализация алгоритмов прогнозирования продаж.
    - Предсказательный анализ поведения клиентов.
13. Мобильная совместимость:
    - Доступность через мобильные устройства для анализа в движении.
14. Функции совместной работы:
    - Возможность нескольким пользователям совместно работать над анализом.
    - Функции комментирования и обмена мнениями для получения инсайтов.
15. Масштабируемость:
    - Проектирование приложения с учетом возможности масштабирования при росте данных и требований пользователей.
16. Обработка ошибок и ведение журнала:
    - Реализация надежных механизмов обработки ошибок.
    - Система ведения журнала для отслеживания и анализа ошибок.
17. Обучение пользователей и поддержка:
    - Предоставление документации и обучающих ресурсов.
    - Предоставление поддержки пользователям по техническим вопросам.
18. Механизм обратной связи:
    - Включение механизма обратной связи для предоставления пользователями обратной информации.
    - Итеративное улучшение приложения на основе обратной связи пользователей.

Эти требования формируют основу для создания комплексного приложения по анализу электронной коммерции, охватывающего различные аспекты от обработки данных до опыта пользователя. Конкретные потребности бизнеса и его цели дополнительно уточнили бы эти требования.

Данные для анализа

1. Продажи:
   - Общие объемы продаж.
   - Продажи по категориям товаров/услуг.
   - Динамика продаж по времени.
2. Конверсия:
   - Коэффициент конверсии (отношение числа покупок к числу посетителей).
3. Средний чек:
   - Средняя сумма каждой транзакции.
4. Трафик и Посещаемость:
   - Общее количество посетителей.
   - Источники трафика (органический, прямой, реферальный).
5. Корзина товаров:
   - Среднее количество товаров в корзине.
6. Отказы и отмены заказов:
   - Процентное соотношение отказов от покупки или отмены заказов.
7. Компоненты клиента:
   - Регистрации новых клиентов.
   - Возвращение повторных клиентов.
8. Популярность товаров:
   - Топ-продаваемые товары/услуги.
9. Возвраты товаров:
   - Процент возвратов или обменов товаров.
10. Время нахождения на сайте:
    - Среднее время, проведенное пользователями на сайте.
11. Средний доход на пользователя:
    - Средний доход, генерируемый каждым пользователем.
12. Сезонные тенденции:
    - Анализ изменения показателей в различные времена года или в связи с определенными событиями (распродажи, праздники).
13. Эффективность маркетинговых кампаний:
    - ROI по маркетинговым кампаниям.

- Конверсия из рекламы.

14. Уровень запасов:

- Мониторинг уровня запасов и предупреждение о возможной нехватке.

15. Прогнозирование продаж:

- Алгоритмы для прогнозирования будущих продаж и спроса.

Эти данные могут быть важными для оценки общей эффективности бизнеса, выявления трендов, оптимизации операций и принятия стратегических решений в области электронной коммерции. Подход к анализу данных должен соответствовать конкретным потребностям и целям бизнеса.

Способы визуализации данных

Визуализация данных в сервисах электронной коммерции играет ключевую роль в обеспечении наглядности и понимания основных метрик и трендов. Вот несколько вариантов визуализации данных:

1. Линейные графики продаж:

- Показывают динамику продаж по времени, позволяя выявить сезонные тренды и изменения.

2. Круговые диаграммы доли продаж по категориям:

- Визуализация структуры продаж, выделяя доли различных категорий товаров.

3. Столбчатые диаграммы среднего чека:

- Иллюстрируют изменения средней суммы транзакции для лучшего понимания динамики покупок.

4. Графики конверсии:

- Отражают процесс конверсии по шагам, начиная от посещения сайта до завершения покупки.

5. Диаграммы корзины товаров:

- Показывают среднее количество товаров в корзине и взаимосвязь с объемами продаж.

6. Скаттер-плоты клиентской активности:

- Визуализация активности клиентов в зависимости от времени и сумм покупок.

7. Графики возвратов товаров:

- Визуализация процента возвратов и понимание причин возвратов.

8. Динамические дашборды:

- Комплексные дашборды с различными графиками и диаграммами для общего обзора ключевых метрик.

9. Графики уровня запасов:

- Мониторинг уровня запасов с течением времени и предупреждение о возможной нехватке.

10. Графики сезонных трендов:

- Визуализация изменений ключевых метрик в зависимости от сезонных факторов.

Выбор конкретных типов визуализации зависит от конкретных потребностей бизнеса и целей анализа. Комбинирование различных видов графиков может обеспечить полное понимание данных.

Архитектура нейронной сети

Для задачи предсказания спроса на товар, хорошим выбором может быть использование рекуррентных нейронных сетей (RNN) или их улучшенных версий, таких как долгосрочная краткосрочная память (LSTM) или gated recurrent unit (GRU). Эти архитектуры способны учитывать последовательность данных, что важно при анализе временных рядов, каким является спрос на товар.

Можно также рассмотреть использование сверточных нейронных сетей (CNN) для анализа пространственных шаблонов в данных, если такие шаблоны могут быть важными при предсказании спроса (например, визуальные характеристики товаров).

Комбинация различных слоев и типов архитектур (например, сочетание RNN и CNN) также может быть эффективной для улучшения точности предсказаний. Важным аспектом является также правильное представление данных и настройка гиперпараметров в соответствии с особенностями вашего набора данных.

Пример нейронной сети предсказания спроса на товар

Создание полноценной нейронной сети для предсказания спроса на товар включает в себя множество шагов, включая подготовку данных, выбор архитектуры сети, обучение и оценку модели. В данном случае, я предоставлю простой пример нейронной сети на языке Python, используя библиотеку TensorFlow и Keras.

```
####################################
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler

# Подготовка данных (здесь данные представлены для иллюстрации)

# Предположим, у вас есть данные о продажах товаров и других факторах, влияющих на спрос.
# X содержит признаки, такие как цена, рекламные расходы, и т.д.
# y содержит целевую переменную - объем продаж.

# Пример данных:
# X = np.array([[цена1, реклама1, ...], [цена2, реклама2, ...], ...])
# y = np.array([продажи1, продажи2, ...])

# Разделение данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Нормализация данных
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Построение модели нейронной сети
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1)  # Выходной слой без активации для регрессии
])

# Компиляция модели
model.compile(optimizer='adam', loss='mean_squared_error')

# Обучение модели
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)

# Оценка модели
loss = model.evaluate(X_test, y_test)
print(f'Loss on test data: {loss}')

# Предсказание спроса на новых данных
new_data = np.array([[новая_цена, новая_реклама, ...]])
scaled_new_data = scaler.transform(new_data)
predicted_sales = model.predict(scaled_new_data)
print(f'Predicted sales: {predicted_sales}')
####################################
```

Это пример простой нейронной сети для задачи регрессии. Пожалуйста, помните, что конкретная архитектура и параметры модели могут требовать

тщательной настройки в зависимости от характера ваших данных.

React компонент графика спроса товара

```
import React from 'react';
import { Line } from 'react-chartjs-2';

const DemandChart = ({ demandData }) => {
  const data = {
    labels: demandData.labels, // массив с метками времени (например, датами)
    datasets: [
      {
        label: 'Спрос на товар',
        fill: false,
        lineTension: 0.1,
        backgroundColor: 'rgba(75,192,192,0.4)',
        borderColor: 'rgba(75,192,192,1)',
        borderCapStyle: 'butt',
        borderDash: [],
        borderDashOffset: 0.0,
        borderJoinStyle: 'miter',
        pointBorderColor: 'rgba(75,192,192,1)',
        pointBackgroundColor: '#fff',
        pointBorderWidth: 1,
        pointHoverRadius: 5,
        pointHoverBackgroundColor: 'rgba(75,192,192,1)',
        pointHoverBorderColor: 'rgba(220,220,220,1)',
        pointHoverBorderWidth: 2,
        pointRadius: 1,
        pointHitRadius: 10,
        data: demandData.values, // массив с данными о спросе на товар
      },
    ],
  };

  return (
    <div>
      <h2>График спроса на товар</h2>
      <Line data={data} />
    </div>
  );
};

export default DemandChart;
```

Шаблон сервеиса получения данных от wildberries

Для создания Spring-сервиса, получающего данные о товарах из API Wildberries, вы можете использовать библиотеку RestTemplate для выполнения HTTP-запросов.

Важно учитывать, что доступ к API Wildberries может потребовать авторизации и использование соответствующих токенов.

Прежде всего, удостоверьтесь, что у вас есть зависимость на spring-boot-starter-web и добавьте необходимые библиотеки в файл pom.xml.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Пример кода для создания Spring-сервиса:

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
public class WildberriesApiService {

   @Value("${wildberries.api.url}")
   private String wildberriesApiUrl;  // URL API Wildberries

   @Value("${wildberries.api.token}")
   private String wildberriesApiToken;  // Токен для доступа к API

   private final RestTemplate restTemplate;

   public WildberriesApiService(RestTemplate restTemplate) {
      this.restTemplate = restTemplate;
   }

   public String getWildberriesData() {
      // Формируем URL для запроса
        String apiUrl = wildberriesApiUrl + "/someEndpoint";  // Замените "/someEndpoint" на реальный путь API

      // Добавляем токен в заголовки запроса
      HttpHeaders headers = new HttpHeaders();
      headers.setBearerAuth(wildberriesApiToken);

      // Выполняем GET-запрос и получаем ответ в виде строки
        ResponseEntity<String> response = restTemplate.exchange(apiUrl, HttpMethod.GET,
```

```
new HttpEntity<>(headers), String.class);

        // Обрабатываем ответ (может потребоваться парсинг JSON, обработка ошибок и т.д.)
        String responseBody = response.getBody();

        return responseBody;
    }
}
```

В вашем application.properties или application.yml укажите URL API и токен:

```
wildberries.api.url=https://api.wildberries.ru
wildberries.api.token=your_api_token
```

Не забудьте настроить RestTemplate в конфигурации вашего приложения. В новых версиях Spring Boot предпочтительно использовать RestTemplateBuilder:

```
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class AppConfig {

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder restTemplateBuilder) {
        return restTemplateBuilder.build();
    }
}
```

Обратите внимание, что это основной пример, и вы, вероятно, захотите дополнительно обработать ответ API, добавить логирование и учесть аспекты безопасности.

Чтобы добавить сохранение данных в базу данных PostgreSQL в вашем Spring-сервисе, нужно внести несколько изменений. В приведенном ниже примере я буду использовать Spring Data JPA для работы с базой данных. Убедитесь, что у вас есть зависимость на spring-boot-starter-data-jpa в вашем pom.xml.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
```

</dependency>

Теперь приступим к изменениям в вашем коде.

1. Создайте Entity:

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class WildberriesData {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String data;  // Замените на реальные поля данных

    // Геттеры и сеттеры

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```

2. Создайте Repository:

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface WildberriesDataRepository extends JpaRepository<WildberriesData, Long>
{
    // Дополнительные методы, если необходимо
}
```

3. Измените ваш сервис:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
@Service
public class WildberriesApiService {

    @Autowired
    private WildberriesDataRepository wildberriesDataRepository;

    // ... (остальной код)

    public String getWildberriesData() {
        // ... (остальной код)

        // Обрабатываем ответ (может потребоваться парсинг JSON, обработка ошибок и т.д.)
        String responseBody = response.getBody();

        // Сохраняем данные в базу данных
        WildberriesData wildberriesData = new WildberriesData();
        wildberriesData.setData(responseBody);
        wildberriesDataRepository.save(wildberriesData);

        return responseBody;
    }
}
```

Теперь ваш сервис будет сохранять данные в базу данных PostgreSQL при каждом запросе к API Wildberries. Убедитесь, что вы правильно настроили параметры подключения к базе данных в файле application.properties или application.yml.