

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	7
1.1 Машинное обучение. Задача классификации	7
1.2 Подготовка данных и извлечение признаков	11
1.3 Машина опорных векторов	13
1.4 Анализ имеющихся решений	15
1.5 Постановка задач исследования	16
2 РАЗРАБОТКА АРХИТЕКТУРЫ ПРОГРАММЫ	17
2.1 Требования к реализуемой программе	17
2.2 Проектирование архитектуры	18
2.2.1 Проектирование модуля обучения	20
2.2.2 Проектирование модуля классификации	22
2.2.3 Проектирование API	23
3 ОПИСАНИЕ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММЫ	24
3.1 Модуль обучения	24
3.2 Модуль классификации	27
3.3 Модуль API	29
4 ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ПРОГРАММЫ	30
4.1 Тестирование программы	30
4.2 Тестирование модели	31
ЗАКЛЮЧЕНИЕ	32
Библиографический список	34

ВВЕДЕНИЕ

Актуальность темы. В конце 20-го века с развитием информационных технологий стало возможным делать покупки не выходя из дома, возможности для этого предоставляют интернет-магазины. С ростом количества интернет-магазинов появились и системы мониторинга цен и предложений конкурентов для предпринимателей, владеющих интернет-магазинами. Такой мониторинг позволяет автоматически формировать цены на товары в магазине в соответствии с текущей ситуацией на рынке. Одной из важнейших задач таких систем является установления связей между товарами предпринимателя и его конкурентов. Автоматизация подобных процессов позволяют людям сэкономить время и ресурсы.

Целью работы является создание программы поиска аналогов на основе машинного обучения. Такая программа сможет без ручного внесения в базу устанавливать связи между товарами владельца интернет-магазина и его конкурентов. Таким образом, нет необходимости отслеживать появление новых товаров у конкурентов и заносить новую связь в базу, программа найдет соответствие сама, обучившись на большом наборе подобных связей.

Для достижения цели исследования был сформулирован следующий ряд **задач**:

- провести обзор машинного обучения;
- определить требования к программе;
- осуществить проектирование и разработку программы;
- произвести тестирование приложения.

Объем и структура работы. Работа содержит 40 страниц печатного текста, 15 рисунков, 2 таблицы, список литературы, включающий 18 источников. Работа состоит из введения, четырех частей и заключения. Во введе-

нии обоснована актуальность работы, определены цель и задачи исследования. В первой части произведено краткое описание области машинного обучения, а также обзор существующих решений. Вторая часть содержит описание структуры программы и выбранных средств разработки. Третья глава описывает реализацию программы. Четвертая глава описывает тестирование разработанной программы и характеристики работы. В заключении приведены основные результаты работы и возможные направления для дальнейшего развития.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Машинное обучение. Задача классификации

Машинное обучение – группа методов искусственного интеллекта, которые используются компьютерными системами для эффективного выполнения конкретной задачи без использования явных инструкций, вместо этого обучаясь на множестве решений сходных задач.

Машинное обучение подразделяется на обучение по прецедентам, что означает выявление общих закономерностей по частным эмпирическим данным, и дедуктивное обучение, при котором формируется база знаний на основе формализованных знаний экспертов.

В данной работе речь пойдет об обучении по прецедентам, одним из подтипов которого является обучение с учителем. Обучение с учителем – самый распространенный случай, оно производится на наборе обучающих примеров, каждый из которых представляет собой пару ”объект, ответ”. В процессе обучения выводится функциональная зависимость ответа от объекта и строится алгоритм, который позволяет отображать входные данные в выходные. Когда множество возможных ответов конечно, говорят о задачах классификации и распознавания образов.

Задача классификации – ответ принадлежит конечному множеству, он называется меткой класса. Класс представляет собой множество объектов, которые соответствуют данному значению метки. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется выборкой. Классовая принадлежность остальных объектов неизвестна. Требуется построить алгоритм, способный классифицировать произвольный объект из исходного множества.

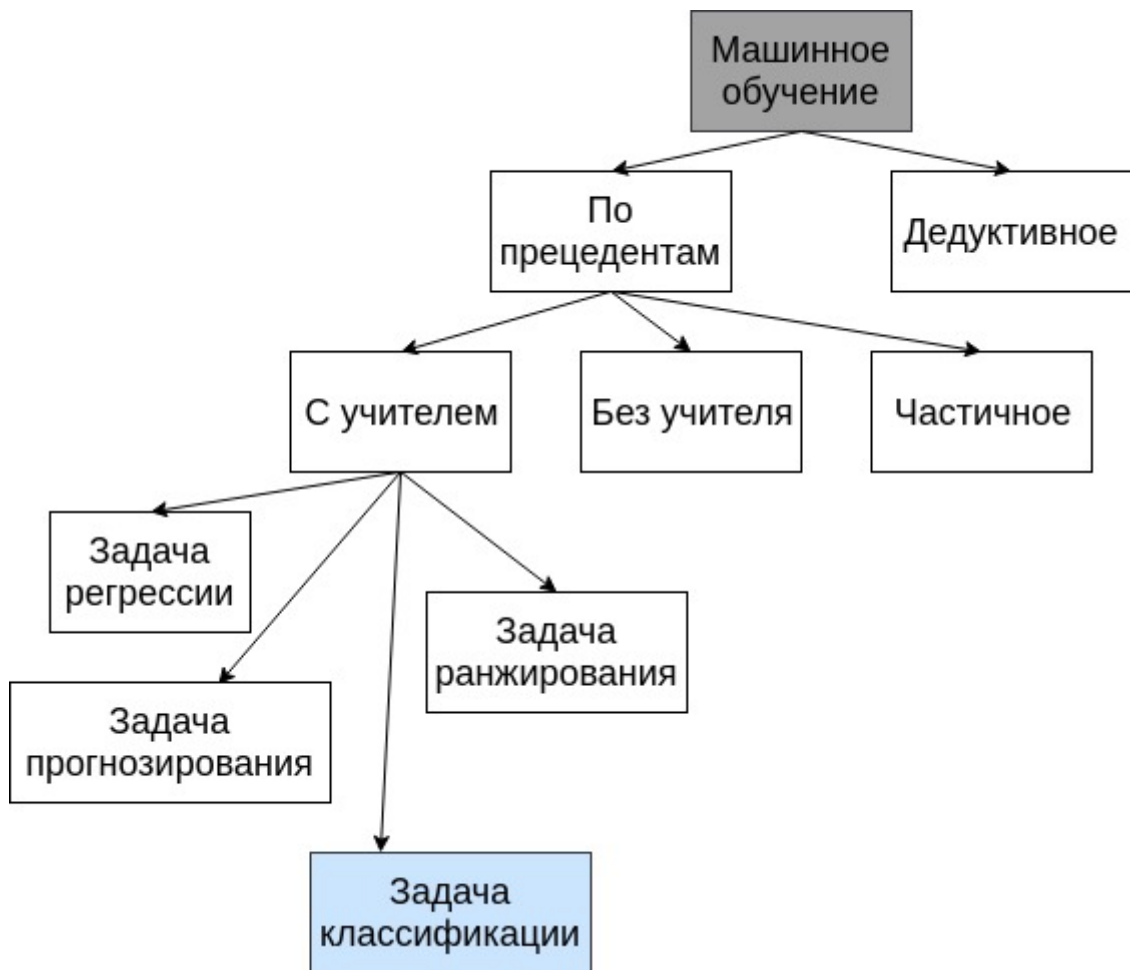


Рисунок 1.1 – Машинное обучение

На рисунке 1.1 представлено дерево способов и задач, которые решаются машинным обучением. Дерево неполное, так как существует очень много способов машинного обучения. Поэтому на данном рисунке показана основная ветвь, имеющая прямое отношение к поставленной задаче.

Классификатором называется отображение $\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$, где $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ — конечное множество меток классов. Под C_i можно также понимать множество объектов, которые относятся к классу с номером i . Знак "крышки" означает, что $\hat{c}(x)$ — оценка истинной, но неизвестной функции $c(x)$. Обучающие примеры для классификатора имеют вид пар $(x, c(x))$, где $x \in \mathcal{X}$ — объект, а $c(x)$ — истинный класс, к которому принадлежит этот объект. Под обучением классификатора понимается выявление функции \hat{c} , которая как можно лучше аппроксимирует c не только на обучающем наборо-

ре, но в идеале и на всем пространстве объектов.[1]

Типы классификации:

- Бинарная классификация – требуется определить к какому из двух классов относится объект, простой в техническом отношении случай, который служит основой для решения более сложных задач;
- Многоклассовая классификация – более двух классов, задача классификации становится более трудной, решающая граница не столь очевидна, как в первом случае;
- Непересекающиеся классы – объект относится только к одному из множества классов;
- Пересекающиеся классы – объект может относиться одновременно к нескольким классам;
- Нечёткие классы – определяется степень принадлежности объекта каждому из классов.

Типы входных данных:

- Признаковое описание — каждый объект – это набор признаков, признаки могут быть числовыми или нечисловыми;
- Матрица расстояний между объектами. Каждый объект описывается расстояниями до всех остальных объектов обучающей выборки. С этим типом входных данных работают методы ближайших соседей, парзенковского окна, потенциальных функций;
- Временной ряд или сигнал представляет собой последовательность измерений во времени;
- Изображение или видеоряд.

Также входные данные могут представляться в виде графов, текстов. Они приводятся к первому или второму типу путём предварительной обработки данных и извлечения признаков.

Классификация текстов (документов) — одна из задач информационного поиска, заключающаяся в определении принадлежности текста к одному из нескольких классов на основании содержания текста. Классификацию документов можно организовать с помощью методов машинного обучения. При таком подходе сохраняется необходимость разметки обучающего множества, то есть человек заранее связывает некоторые документы с классами, тем самым формируя обучающий корпус. Таким образом, классификация документов является примером обучения с учителем, в роли учителя выступает человек, размечающий тексты.

Этапы обработки:

- Индексация документов. Переход к числовой модели, преобразование документов в вектора и определение весов слов;
- Построение и обучение классификатора. Могут использоваться различные методы машинного обучения: решающие деревья, наивный байесовский классификатор, нейронные сети, метод опорных векторов и др.;
- Оценка качества классификации.

1.2 Подготовка данных и извлечение признаков

В рамках данной работы речь идет о классификации текстов или документов. Документы, которые требуется классифицировать, – названия моделей конкурента, а множество классов – это множество названий моделей владельца магазина. Входные данные в обучающем корпусе являются текстами, которые необходимо преобразовать и представить в виде, в котором на них сможет обучаться выбранная модель. С использованием всех объектов-документов в наборе обучающих данных составляется словарь размера n – множество всех признаков, встретившихся в документах. Процесс выделения из текстов признаков с использованием разделителя называется токенизацией.

Далее каждый документ представляется вектором размера n , в котором каждому признаку из словаря ставится в соответствие количество вхождений данного признака в документ.

После преобразования текстовых документов к векторам происходит переход к частотности. TF (term frequency — частота слова) — отношение числа вхождений некоторого слова к общему числу слов документа. Оценивается важность слова t_i в пределах отдельного документа.

$$\text{tf}(t, d) = \frac{n_t}{\sum_k n_k},$$

где n_t — число вхождений слова t в документ, а в знаменателе общее количество слов в документе.

Еще одно уточнение меры — IDF (inverse document frequency — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается во всех документах корпуса данных. Таким образом, чем чаще

слово встречается в пределах всех коллекции, тем меньше становится вес таких слов. Для каждого слова существует только одно IDF в пределах коллекции.

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|},$$

где $|D|$ — общее число документов в обучающем корпусе, а в знаменателе логарифма число документов, в которых встречается t .

$$\text{tf-idf}(t, D) = \text{tf}(t, d) \times \text{idf}(t, D),$$

Мера TF-IDF это произведение TF и IDF. После нормализации наибольший вес получают слова, которые часто встречаются в пределах какого-либо документа, но редко во всех коллекции документов.

1.3 Машина опорных векторов

Существует множество алгоритмов, которые используются для обучения текстовых классификаторов. Одним из них является метод опорных векторов, выбранный мной в соответствии с результатами исследования, проведенными Т. В. Батурой в статье "Методы автоматической классификации текстов".[2] В этой статье проводится сравнение нескольких алгоритмов и выявляется, что одно из лучших соотношений характеристик качества достигается при использовании метода опорных векторов (для коллекции текстов объемом 1 000–2 000 текстов точность 80–85 %, полнота 83–87 %).

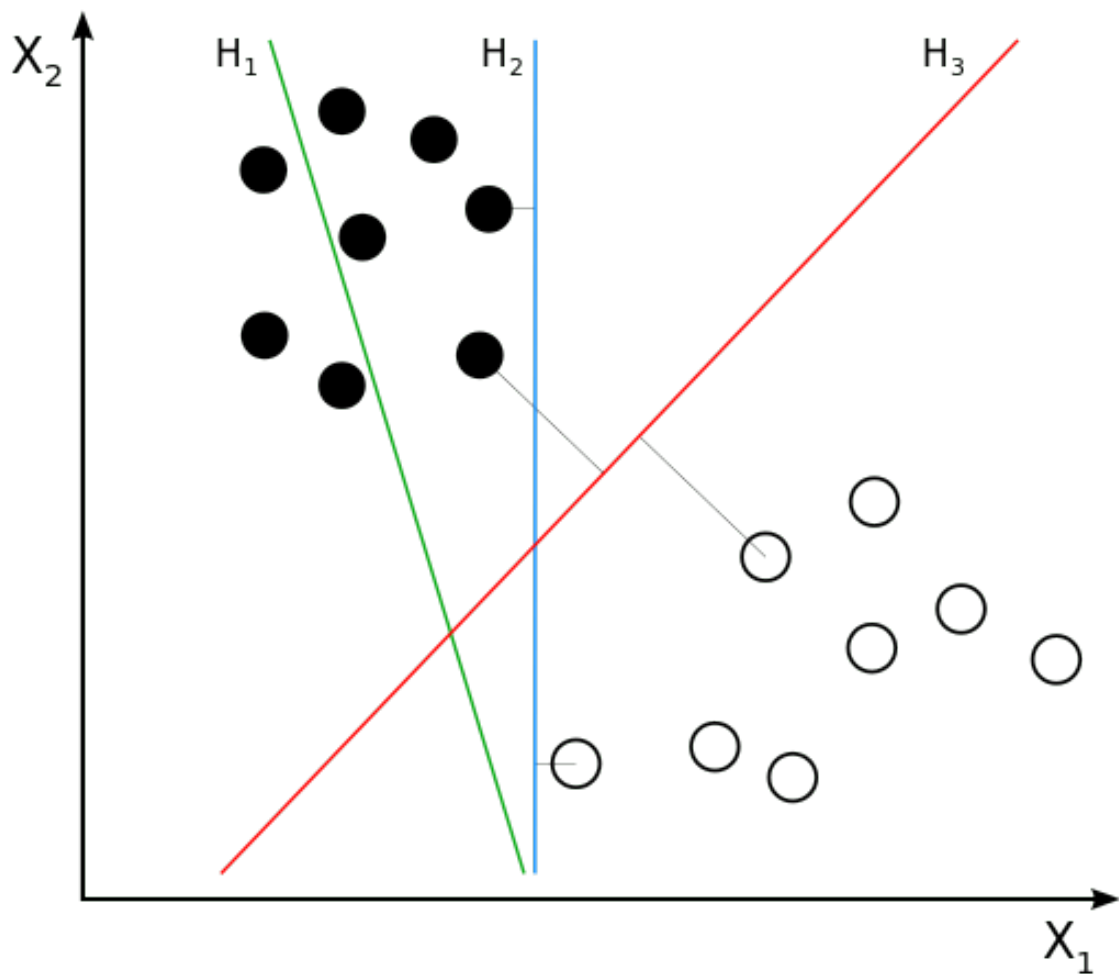


Рисунок 1.2 – Метод опонных векторов

Машина опорных векторов (метод опорных векторов) – это модель на основе обучения с учителем. Существует множество векторов размера n . Каждый из них представляется точкой в n -мерном пространстве. Основная задача состоит в том, чтобы найти гиперплоскость размерности $(n - 1)$, которая сможет разделить точки в указанном пространстве, тем самым обозначая их принадлежность к разным классам. Причем расстояние от ближайших к гиперплоскости точек до самой гиперплоскости, которое называется зазором, должно быть максимальным, так как предполагается, что это обеспечит наибольшую точность при классификации. В случае линейной неразделимости, то есть когда невозможно найти гиперплоскость, разделяющую множество точек, все вектора переводятся в пространство более высокой размерности и определяется разделяющая гиперплоскость.[3]

В случае мультиклассовой классификации применяется решающее правило, основанное на разбиении задачи на бинарные по схеме ”один против остальных” (one-vs-rest). Обучается n классификаторов, где n – количество классов. Классификатор с самым высоким значением функции выхода используется при классификации.

На рисунке 1.2 показано множество векторов, изображенных в виде точек, и несколько гиперплоскостей, разделяющих эти точки на два класса. Точки, ближайšie к гиперплоскостям, называются опорными векторами. По расстоянию от гиперплоскости до опорных векторов (перпендикуляр изображен серым цветом) определяется величина зазора. На рисунке оптимальной гиперплоскостью является H_3 , так как зазор является максимальным.

1.4 Анализ имеющихся решений

Одним из имеющихся решений является Elasticsearch. **Elasticsearch** – свободная программная поисковая система, написанная на языке Java, которая используется большим числом крупных сайтов и компаний, например, GitHub, Foursquare, SoundCloud и другие. Данная система может использоваться для решений многих задач, одной из которых является сопоставление товаров для систем мониторинга цен конкурентов.[4] Однако для организации связей товаров Elasticsearch требует четкого задания классов синонимов, то есть групп слов, которые будут распознаваться как один и тот же признак в названии товара. Такой способ не подходит, если в поиске участвует большой объем названий товаров из разных категорий и невозможно вручную установить связи между словами из-за их количества и неопределенности.

1.5 Постановка задач исследования

В рамках данной работы поставлена задача разработать программу, которая сможет находить связи между товарами, основываясь на предсказаниях обученного на корпусе данных классификатора. Для решения поставленной цели необходимо решить следующие задачи:

- Определить требования к программе;
- Спроектировать архитектуру программы;
- Реализовать программу в соответствии с разработанной архитектурой;
- Провести тестирование.

2 РАЗРАБОТКА АРХИТЕКТУРЫ ПРОГРАММЫ

2.1 Требования к реализуемой программе

Требования к программам и системам можно разделить на функциональные и нефункциональные. Функциональные требования позволяют обозначить функционал, которым должна обладать программа для удовлетворения просьб заказчика. Нефункциональные требования — это дополнительные атрибуты качества, которые важны для разработчика и заказчика.

К разрабатываемой программы были поставлены следующие функциональные требования:

- Программа должна иметь функционал для сохранения и загрузки обученной модели;
- Программа не должна находить связь, если в запросе был указан товар, который не существует в базе клиента;
- Предсказания должны осуществляться с точностью более 90%;
- Время ответа классификатора не должно превышать 0.1 с.

Нефункциональные требования:

- Программа должна иметь API для возможности интеграции в приложение.

2.2 Проектирование архитектуры

Для разработки программы была выбрана модульная архитектура. Модульная архитектура — архитектура, при которой программа организована как совокупность независимых друг от друга блоков, структура и поведение которых подчиняются определенным правилам. Использование такой архитектуры позволяет упростить тестирование и масштабирование, так как каждый модуль имеет строго определенные задачи, которые можно дополнять и тестировать отдельно от остальных модулей.

Использование данной архитектуры позволило разбить программу на три фрагмента: обучение, классификация, API. Структура программы представлена на рисунке 2.1.

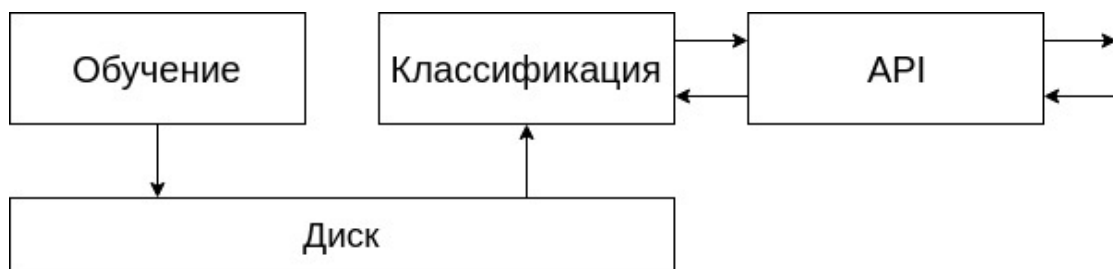


Рисунок 2.1 – Структура программы

Одной из основных задач разработки является выбор языка программирования. Для реализации программы был выбран Python. Python — высокоуровневый язык программирования общего назначения. Отличительной чертой Python является минималистичность синтаксиса, что повышает производительность и скорость разработчика, а также читаемость кода.[5] Простота Python в совокупности с большим количеством библиотек как для работы с данными, так и для машинного обучения делают его очень популярным инструментом для математических расчетов и задач, связанных с анализом и машинным обучением.

Модули программы:

- Процесс обучения представлен модулем на языке Python, взаимодействие с которым происходит через методы класса с помощью интерпретатора данного языка через командную строку. Используется для построения и обучения классификатора. Результатом работы является запись на диск файла обученного классификатора и вспомогательных файлов, которые в последствии используются модулем классификации;
- Модуль классификации в качестве входных данных принимает документ, класс которого требуется определить, а также загружает с диска файл классификатора. Выходными данными является метка, которую определил классификатор. Взаимодействие с модулем происходит через методы класса с помощью интерпретатора Python через командную строку напрямую или же через API;
- API — средство для общения клиента с обученным классификатором с целью получения предсказания. Взаимодействие происходит через HTTP-запросы, входные данные — это документ, требующий классификации, а выходные — класс.

2.2.1 Проектирование модуля обучения

Для подготовки данных, построения и обучения классификатора была использована библиотека **Scikit-learn**. [6] Это наиболее популярная бесплатная библиотека для машинного обучения на языке Python. Она реализует различные алгоритмы классификации, регрессии и кластеризации, включая машины опорных векторов, случайные леса, повышение градиента, k-средних и DBSCAN, и предназначена для взаимодействия с математическими библиотеками NumPy и SciPy. В ее возможности также входит преобразование данных, извлечение признаков, нормализация, удаление шумовых слов.

NumPy — библиотека с открытым исходным кодом для языка программирования Python, основными возможностями которой являются поддержка многомерных массивов и поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами [7].

Еще одним инструментом, использующимся совместно с Scikit-learn является библиотека для обработки и анализа данных **pandas**, которая строится поверх NumPy. Представляет специальные структуры данных для манипулирования индексированными массивами двумерных данных, которые были использованы при загрузке и подготовке корпуса обучающих данных [8].

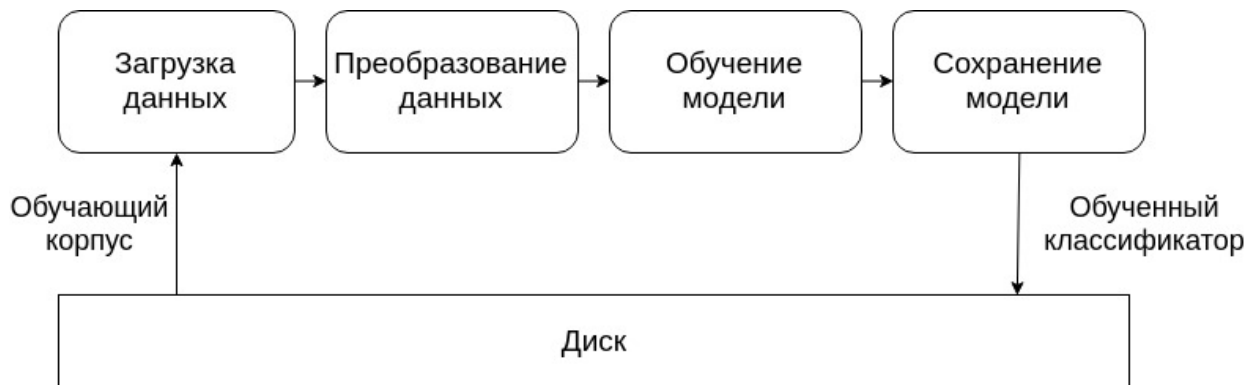


Рисунок 2.2 – Структура модуля обучения

Для сохранения и загрузки файлов обученного классификатора была использована библиотека **Joblib**. **Joblib** — это набор инструментов для облегченной конвейерной обработки в Python. В частности, инструменты дампа и загрузки данных[9].

Структура модуля обучения представлена на рисунке 2.2. Данный модуль с помощью описанных выше библиотек решает ряд задач: загрузку обучающих данных из файла, подготовку данных, извлечение признаков, нормализацию, построение и обучение классификатора, сохранение классификатора для дальнейшего использования. Этот модуль работает независимо от остальных модулей программы и используется только для создания обученной модели, поэтому взаимодействие с ним происходит с помощью интерфейса командной строки через интерпретатор Python.

2.2.2 Проектирование модуля классификации

Преобразование данных перед классификацией происходит так же, как и в обучающем модуле, посредством библиотеки Scikit-learn. Входными данными является текстовый документ, который необходимо классифицировать. Он представляется в виде вектора, после чего передается классификатору. Результатом классификации является метка класса, к которому относится входной документ, представленная в форме строки.

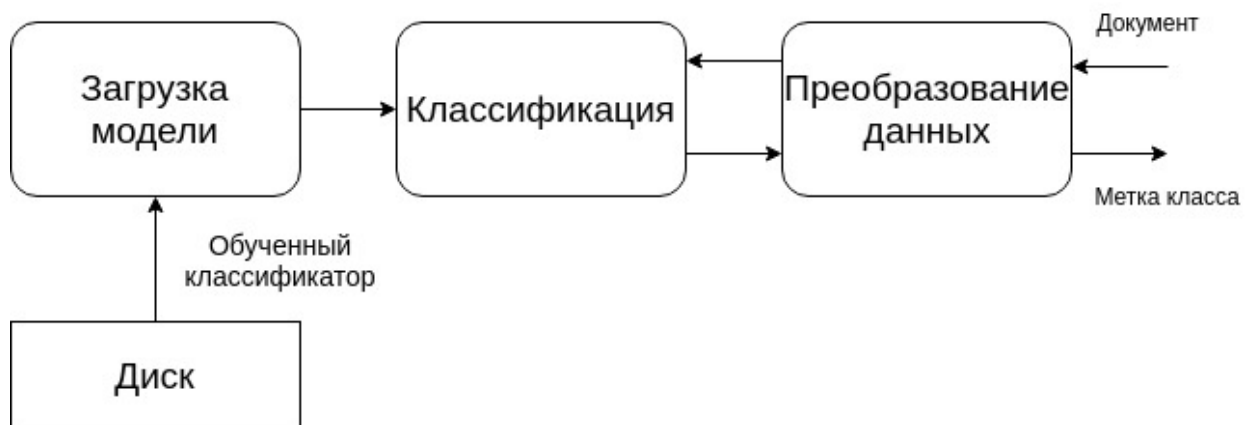


Рисунок 2.3 – Структура модуля обучения

Структура модуля классификации представлена на рисунке 2.3. Чтобы осуществить классификацию, необходимо сначала загрузить обученный классификатор и вспомогательные файлы (модели для преобразования входных данных) с диска. После загрузки модели классификатора с ним осуществляется взаимодействие с помощью интерфейса командной строки через интерпретатор Python или через модуль API.

2.2.3 Проектирование API

Для соответствия программы нефункциональному требованию о возможности интеграции в другое приложение был разработан модуль API. Он представляет из себя веб-сервер, созданный с использованием Flask. **Flask** — фреймворк для создания веб-приложений на языке программирования Python, относится к категории микрофреймворков — минималистичных каркасов веб-приложений, предоставляющих лишь самые базовые возможности[10]. Этот фреймворк в рамках данной работы был выбран для создания API из-за его минималистичности, простоты использования и запуска.

Взаимодействие клиента с API происходит через HTTP-запросы. HTTP — протокол прикладного уровня передачи данных, в настоящий момент используется для передачи произвольных данных. Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

- Клиентов, которые инициируют соединение и посылают запрос;
- Серверов, которые ожидают соединения и запроса, производят необходимые действия и возвращают клиенту сообщение с результатом.

Серверная часть API извлекает требующий классификации документ из запроса, передает его модулю классификации, и, получив ответ классификатора, возвращает этот ответ клиенту посредством HTTP.

3 ОПИСАНИЕ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММЫ

Программа представляет собой набор модулей на языке Python. Далее будет описан процесс разработки каждого из модулей программы.

3.1 Модуль обучения

Модуль обучения представлен классом на языке Python, который содержит метод для загрузки корпуса данных, метод, в котором происходит преобразование данных корпуса, построение и обучение модели, а также метод сохранения обученной модели на диск.

Для преобразования набора обучающих данных в массив векторов используется класс библиотеки Scikit-learn `CountVectorizer`. Он принимает в качестве аргумента конструктора разделитель, производит токенизацию всех документов коллекции, на этой основе составляет словарь токенов коллекции и конвертирует все документы в вектора, содержащие количество вхождений токена в каждый документ.

В данном классе также реализован метод для загрузки обучающей коллекции средствами библиотеки `pandas`. В листинге 3.2 представлен пример построения объекта `CountVectorizer` и перехода к векторам.

Листинг 3.1 – Использование `CountVectorizer`

```
1 X, y = search.read_dataset();  
2  
3 t0 = time()  
4 self.vectorizer.fit(X)  
5 print('vectorizer fitted in %fs' % (time() - t0,))  
6
```

```
7 X = self.vectorizer.transform(X)
```

Для нормализации значений векторов был использован класс той же библиотеки `TfidfTransformer`. Он преобразовывает значения в каждом векторе коллекции согласно статической мере TF-IDF, используя значения, которые были установлены объектом класса `CountVectorizer`. В листинге 3.3 представлено построение объекта `TfidfTransformer` и нормализация значений векторов.

Листинг 3.2 – Использование `TfidfTransformer`

```
1 t0 = time()
2 self.tfidf.fit(X)
3 print('tfidf fitted in %fs' % (time() - t0,))
4
5 X = self.tfidf.transform(X)
```

Выбранный для классификации алгоритм машины опорных векторов в библиотеке `Scikit-learn` реализован классом `LinearSVC`[6]. Поддерживает мультиклассовую классификацию по схеме `one-vs-the-rest`. В листинге 3.4 показан вызов метода класса `LinearSVC`, с помощью которого строится классификатор.

Листинг 3.3 – Использование `LinearSVC`

```
1 t0 = time()
2 self.clf.fit(X, y)
3 print("done in %fs" % (time() - t0))
```

Одним из параметров конструктора `LinearSVC` является метод регуляризации. Была выбрана регуляризация через манхэттенское расстояние или L_1 , так как в этом случае была достигнута наибольшая точность. Регуляризация — метод добавления некоторых дополнительных ограничений к усло-

вию с целью предотвратить переобучение. То есть, если в процессе обучения в получающихся многочленах слишком большие коэффициенты, к модели применяется штраф.

$$L_1 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i a_i^2$$

Все объекты указанных классов являются полями класса модуля обучения и создаются с необходимыми параметрами в его конструкторе.

В соответствии с функциональными требованиями в данном модуле был создан метод для сохранения объектов CountVectorizer, TfidfTransformer и LinearSVC на диск. Внутри него был использован метод **dump** (value, filename), который принадлежит библиотеке Joblib[9].

3.2 Модуль классификации

Модуль классификации, как и модуль обучения, является классом на языке Python. Этот модуль предназначен для загрузки с диска обученного классификатора и использование этого классификатора для предсказаний.

Возможность загрузки с диска обученной модели — одно из функциональных требований. В методе, представляющем этот функционал, использовался метод **load** (filename), принадлежащий библиотеке Joblib[9].

Еще одним функциональным требованием к программе являлось то, что программа не должна находить связь, если в запросе был указан товар, который не существует в базе клиента. Для реализации этого механизма было положено, что слова, которые ни разу не встречались в корпусе данных, будут иметь отрицательный вес. Это основывается на предположении, что все слова, которые не являются ”важными” словами, то есть идентификационными номерами или названиями моделей, уже встречались в коллекции документов, на которой был обучен классификатор, и занесены в словарь. Номера и названия, которые содержатся в документах обучающей коллекции, являются ”важными” и также занесены в словарь. Если же слово не содержится в словаре, оно не часто употребляемое в рамках данного корпуса, соответственно, является важным элементом документа, который необходимо классифицировать. Но связи с такими документами не должны быть найдены, поэтому все слова входного документа проверяются на наличие их в словаре и, если они отсутствуют, им выставляется отрицательный вес.

Листинг 3.4 – Предсказание

```
1 t0 = time()  
2  
3 predicted = self.clf.predict(input_vectors)[0]  
4  
5 spaces = self.clf.decision_function(input_vectors)[0]
```



```
6  
7 print("\nprediction done in %fs" % (time() - t0))  
8  
9 space = spaces[np.where(self.clf.classes_ == predicted)]  
10 proba = 1 / (1 + exp(-space))
```

После обработки входных данных происходит классификация с помощью метода класса `LinearSVC` **predict** (self, X), который принимает один или несколько векторов и возвращает соответствующие метки класса. В листинге 3.5 представлен фрагмент кода, отражающий этот процесс. Помимо получения метки класса извлекается вероятность данного предсказания, чтобы оценить его точность. Это делается с помощью метода класса `LinearSVC` **decision_function** (self, X), который возвращает расстояние от векторов X до гиперплоскостей для каждого возможного класса. Далее из всех этих расстояний выбирается расстояние до предсказанного класса, после чего в соответствии с функцией предсказания расстояние преобразовывается в вероятность.

3.3 Модуль API

Модуль API это небольшое приложение с использованием Flask. В листинге 3.6 представлена обработка GET-запроса по адресу `"/search"`, параметром `"q"` которого является строка-документ, который необходимо классифицировать.

Листинг 3.5 – Пример обработки запроса

```
1 app = Flask(__name__)
2
3 @app.route('/search', methods=['GET'])
4 def search():
5     input = request.args.get('q')
6     predicted, proba = mlsearch.predict(input)
7     return jsonify(predicted=predicted, proba=proba)
8
9 if __name__ == '__main__':
10     app.run()
```

Код, который обрабатывает запрос, обращается к методу модуля классификации **predict**, это происходит после загрузки обученной модели с диска. После классификации клиент получает ответ в формате JSON, содержащий метку класса и точность данного предсказания.

4 ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ПРОГРАММЫ

4.1 Тестирование программы

Для тестирования программы был создан отдельный модуль.

4.2 Тестирование модели

ЗАКЛЮЧЕНИЕ

В рамках проведённой работы были получены следующие результаты:

- 1.
- 2.
- 3.

Автор планирует продолжать работу над проектом и развивать его, добавляя новые функции и исправляя возможные ошибки, которые могут быть найдены во время эксплуатации приложения.

Библиографический список

1. Flach Peter. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. Москва: ДМК Пресс, 2015. 400 с.
2. Батура Т. В. Методы автоматической классификации текстов. Программные продукты и системы. 2017. №1. URL: <https://cyberleninka.ru/article/n/metody-avtomaticheskoy-klassifikatsii-tekstov> (дата обращения: 01.04.2019).
3. Вьюгин В. В. Математические основы теории машинного обучения и прогнозирования. Москва, 2013. 387 с.
4. Elasticsearch Reference [Электронный ресурс]. Официальный сайт проекта Elasticsearch. 2019. URL: <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html> (дата обращения: 12.04.2019).
5. Python Reference [Электронный ресурс]. Официальный сайт Python. 2019. URL: <https://www.python.org/> (дата обращения: 02.04.2019).
6. Scikit-learn: Machine Learning in Python [Электронный ресурс]. Официальный сайт проекта Scikit-learn. 2019. URL: <https://scikit-learn.org/> (дата обращения: 10.12.2018).
7. NumPy Reference [Электронный ресурс]. Официальный сайт проекта NumPy. 2019. URL: <https://www.numpy.org/> (дата обращения: 10.12.2018).
8. pandas: Python Data Analysis Library [Электронный ресурс]. Официальный сайт проекта pandas. 2019. URL: <https://pandas.pydata.org/> (дата обращения: 10.12.2018).

9. Joblib: running Python functions as pipeline jobs [Электронный ресурс].
Официальный сайт проекта Joblib. 2019. URL: <https://joblib.readthedocs.io/> (дата обращения: 10.12.2018).
10. Flask: A Python Microframework [Электронный ресурс]. Официальный сайт проекта Flask. 2019. URL: <http://flask.pocoo.org> (дата обращения: 30.09.2018).