СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Практика на предприятии позволяет молодому специалисту применить полученные теоретические знания и ознакомиться со специальным оборудованием и программным обеспечением, применяемым при разработке, сопровождении и эксплуатации промышленных информационных систем. Преимуществом такого вида деятельности заключается в том, что студент находится под контролем руководителя, являющегося специалистом с обширным практическим опытом, что позволяет быстрее и лучше усваивать получаемые знания.

Основной задачей практики является автоматизация процессов производства. Автоматизация – одно из направлений научно-технического прогресса, использующее технические средства и математические методы с целью освобождения человека от участия в процессах получения, преобразования, передачи и использования энергии, материалов, изделий или информации, либо существенного уменьшения степени этого участия или трудоёмкости выполняемых операций. Она позволяет более эффективно использовать ресурсы предприятия и конкурировать с другими предприятиями этой же сферы.

Целью практики является разработка инструмента для автоматического разделения смешанных документов, что существенно уменьшает объем работы для людей, которые с этими документами работают.

Работа является актуальной в областях, где содержится большой объём работы с документами, которые хранятся в виде одного файла, что препятствует эффективному пользованию этими документами.

Основные этапы прохождения практики:

– ознакомление с базой прохождения практики;

– ознакомление со структурой организации, перечнем решаемых задач и внутренним распорядком;

– сбор информации индивидуального задания «Инструмент для автоматического разделения смешанных документов»;

– постановка задачи машинного обучения и сбор данных для её решения;

– решение задачи машинного обучения и сохранение полученной модели;

– создание веб-сервиса для обработки документов при помощи полученной модели;

– создание графического интерфейса для удобного использования созданного веб-сервиса;

– анализ полученного результата и написание отчёта.

Основным этапом прохождения практики является ознакомление с предприятием и понимание внутренних производственных процессов этого предприятия.

1 ХАРАКТЕРИСТИКА ПРЕДПРИЯТИЯ ИООО «ЭПАМ СИСТЕМЗ»

1.1 Общие сведения о предприятии

EPAM Systems – крупнейший разработчик проектного (заказного) программного обеспечения и один из ведущих игроков в области ИТ-консалтинга в Центральной и Восточной Европе. Компания основана в 1993 г. В штате более 47850 ИТ-специалистов, выполняющих проекты в более чем 35 странах мира. Отделения компании расположены в Республике Беларусь, Канаде, России, Украине, Казахстане, США, Венгрии, Польше, Великобритании, Германии, Швеции, Швейцарии.

Основные направления деятельности *EPAM*:

– ИТ-консалтинг;
– разработка программного обеспечения;
– интеграция приложений;
– проектирование и миграция приложений;
– тестирование программного обеспечения;
– создание выделенных центров разработки на базе *EPAM Systems*;
– разработка цифровых стратегий.

Клиентами *EPAM Systems* являются:

– *SAP*, *Microsoft*, *Oracle*;
– *Thomson Reuters*, *London Stock Exchange*, *MICEX*;
– БМЗ, Мозырский НПЗ, «Роснефть»;
– «росэнергоатом», Российский фонд федерального имущества;
– *Samsung America*, *AeroMexico*;
– ФНС России, Налоговый комитет Республики Казахстан;
– *The Coca-Cola Company*, *Colgate-Palmolive* и другие.

В 2018 году *EPAM* запустил *InfoNgen* 7.0, аналитическую платформу с элементами машинного обучения для исследования информации и проведения конкурентного анализа данных из более чем 200000 многоязычных веб-источников, а также *Telescope AI*, масштабируемую модульную платформу на базе искусственного интеллекта, которая помогает бизнесу получить всестороннее представление о внутренних процессах организации.

Компания запустила *EPAM SolutionsHub* – библиотеку программных продуктов, акселераторов и *OpenSource*-решений, а также *Open Source Contributor Index* – инструмент, который оценивает вклад коммерческих компаний в развитие решений с открытым исходным кодом.

EPAM стала одной из четырех технологических компаний, входящих в список *Forbes* «25 самых быстрорастущих публичных технологических компаний» каждый раз начиная с 2013 года, а также признана лидером в индустрии ИТ-сервисов в списке *Fortune* «100 самых быстрорастущих компаний» в 2019 и в 2020 годах.

По результатам деятельности за второй квартал, закончившийся 30 июня 2021 года, объем выручки компании вырос на 39.4 процента по сравнению с аналогичным показателем второго квартала 2019 года и составил 881.4

миллиона долларов США. Численность сотрудников *EPAM* составляет более 47850, в их числе 42800 производственных специалистов, что превышает показатель за аналогичный период прошлого года на 32.6 процентов [1].

1.2 История предприятия

В 1993 году Аркадий Добкин и Леонид Лознер основали компанию *EPAM*, глобального поставщика услуг разработки программного обеспечения. Штаб-квартира *EPAM* располагалась в квартире Аркадия в Нью-Джерси, а через океан, в доме Леонида в Минске, находился оффшорный центр разработки программного обеспечения.

В 1994 году *Bally of Switzerland*, один из ведущих мировых модных брендов одежды, стал первым серьёзным заказчиком *EPAM*, поручившим компании создать решение для управления данными на базе *Salesforce Automation* для поддержки проведения операций в Северной Америке.

В 1996 году появился первый глобальный корпоративный проект: решение для автоматизации для *Colgate-Palmolive*. Успех решения на базе *Salesforce Automation* для *Colgate-Palmolive* был замечен и оценён генеральным директором и основателем компании *SAP AG*, что открыло для *EPAM* двери в сферу сотрудничества с компаниями производителями программных продуктов.

В 2001 году *EPAM* расширил спектр своих услуг. В числе новых экспертиз появились цифровые технологии, создание *CRM*-систем и решений для электронной коммерции с возможностью внедрения в глобальном масштабе.

В 2001-2003 гг. в разгар кризиса *Dot-com*'ов *EPAM* впервые признан ведущими аналитиками отрасли как «законодатель моды» в области аутсорсинга услуг по разработке ПО, а также назван самой быстрорастущей компанией. В 2003 году *EPAM* реализовали первый комплексный проект по цифровому взаимодействию для туристического портала *Lasvegas.com*, также к *EPAM* присоединились *LintecProject*.

В 2004-2005 гг. *EPAM* расширил географию своего присутствия и вышел на новый рынок благодаря приобретению *Fathom Technology*, компании-разработчика ПО с головным офисом в Будапеште, Венгрия. С присоединением *Fathom EPAM* сделал стратегический переход от ориентации только на рынок Северной Америки к фокусу на более сбалансированный бизнес, основой для которого становится в том числе рост количества заказчиков в ЕС.

В 2006-2007 гг. *EPAM* получил инвестиции от *Siguler Guff*, чтобы финансировать свои планы роста и помочь компании более активно конкурировать во всем мире. *EPAM* приобрёл *VDI*, одну из ведущих компаний разработчиков программного обеспечения с центрами разработки и поддержки клиентов в России, что позволило расширить своё присутствие и усилить свои позиции на рынках стран СНГ.

В 2008-2009 гг. в разгар мирового экономического спада *EPAM* удвоил свои усилия, чтобы оптимизировать затраты и стать более эффективной компанией, выстраивая доверительные отношения с ключевыми клиентами и предлагая более эффективные услуги и решения. В 2008 году к *EPAM*

присоединились *B2Bits Corp* и *PlusMicro*. В 2009 году присоединилась *Rodmon Systems*.

В 2012-2013 гг. *EPAM* стал первой компанией разработчиком ПО с белорусскими корнями, которая выходит на Нью-Йоркскую фондовую биржу. *EPAM* приобрёл *Empathy Lab* и открыл новую практику по цифровой трансформации бизнеса, предлагая услуги в сфере дизайна, управления клиентским опытом и электронной коммерции. Также в этом году *EPAM* купил компанию *Thoughtcorp*, что дало возможность обеспечить постоянное присутствие в Канаде и расширить компетенции в области *Agile*, аналитики/*BI* и мобильных решений.

В 2014-2015 гг. *Forrester* впервые назвал EPAM лидером в области предоставления услуг по разработке ИТ-решений. Компания охватывала 17 стран, включая Китай, и насчитывало более 13 000 специалистов в различных областях и практиках. С приобретением *Alliance Global Services EPAM* вышел на индийский рынок, расширяя свою географию внедрения решений и усиливая экспертизу в сфере разработки программного обеспечения и услуг автоматизации.

В 2016-2017 гг. *EPAM* расширил свою географию, открыв представительства в Ирландии, Объединенных Арабских Эмиратах, на Филиппинах. *Forrester* назвал *EPAM* лидером в области разработки цифровых платформ, демонстрирующим лучшее понимание и выполнение задач из всех поставщиков, представленных в отчете. *Forbes* снова включил *EPAM* в список 25 самых быстрорастущих публичных технологических компаний. В 2016 году к *EPAM* присоединились *Dextrys* и *InfomatiX*.

В 2018 году *EPAM* отметил своё 25-летие. Приобретение компании *Continuum* и внедрение человеко-ориентированного подхода к дизайну и продуктовой разработке расширило потенциал в сфере инновационного консалтинга и продуктового дизайна и дополнило существующие направления цифрового и сервис-дизайна.

В 2019 году *EPAM* продолжил наращивать спектр комплексных услуг, чтобы помочь своим клиентам адаптироваться к изменениям, стать более гибкими и добиваться успеха на рынке в условиях цифровизации бизнеса. *EPAM* во второй раз вошёл в список *Fortune «*100 самых быстрорастущих компаний мира» и получил награду *Best in Biz Awards* в номинации «Лучшая программа корпоративной социальной ответственности» за большое внимание к развитию образования, сообществ и окружающей среды. Кроме того, к *EPAM* присоединились *test IO*, *Competentum*, *NAYA Technologies*, *Onsolve* и *Axsphere*.

В 2020 году благодаря приобретению компаний *Ricston* и *Deltix*, *EPAM* увеличили своё присутствие на рынке в качестве надёжного партнёра по цифровым решениям и дополнили свой опыт в области *API*, *Salesforce*, аналитики и обработки данных. Кроме того, *EPAM* стал одним из учредителей *MACH Alliance* и начал помогать продвигать цифровые экосистемы и технологии нового поколения. Второй год подряд *EPAM* вошёл в топ компаний, предоставляющих ИТ-услуги в списке «100 самых быстрорастущих компаний

мира» по версии журнала *Fortune*, и был признан одним из начинающих лидеров в сфере интеграции консалтинга и технологий [2].

1.3 Корпоративная политика

С 1993 года *EPAM* помогает мировым лидерам проектировать, разрабатывать и внедрять программное обеспечение, которое меняет мир.

Будучи глобальной компанией, *EPAM* заботится не только о сотрудниках, но и о мире. Развитие образования, защита окружающей среды, социальные программы – все это и многое другое входит в фокус внимания компании. Сделать окружающий мир лучше – задача, достойная *EPAM*.

Одним из главных преимуществ компании является её социальный пакет. Социальный пакет *EPAM* включает в себя:

– жилищную программу для сотрудников с белорусским гражданством. Сотрудники могут взять ссуду на постройку жилья, на пять лет под 7,5% годовых в долларах США. Размер ссуды не должен превышать 50000 долларов;

– программу переезда. EPAM оказывает помощь в переезде, а также помогает с визой как сотруднику, так и членам семьи (официальные супруги и дети);

– право официальному супругу и детям получить бесплатное обслуживание в медицинских центрах, обозначенных в корпоративной программе *EPAM*;

– восьмичасовой рабочий день, а также час обеда;

– услугу при устройстве друга сотрудника, при которой сотруднику выплачивается бонус от 100 до 1200 долларов США;

– доплату за сверхурочную работу;

– отпуск, зависящий от стажа работы. Сотрудники со стажем работы менее трёх лет получают 18 дней отпуска, а сотрудники, имеющие стаж более трёх лет, получают 20 дней отпуска;

– три оплачиваемых выходных в случае рождения, брака или смерти близких родственников;

– семь оплачиваемых больничных дней каждый календарный год при возникновении проблем со здоровьем;

– право на неоплачиваемый отпуск по семейным или другим уважительным причинам по согласованию со своим менеджером;

– отпуск по беременности и родам – 126 календарных дней, который может быть продлён сверх периода по согласованию с менеджером;

– выплаты по декретному отпуску, которые рассчитываются в соответствии с законодательством Республики Беларусь;

– выплату в размере 500 долларов США в случае потери близкого родственника;

– бесплатную стоматологическую помощь в медицинских центрах, обозначенных в корпоративной программе *EPAM*. В иных учреждения скидка может составлять от 10 до 50 процентов;

– бесплатные вакцины от различных заболеваний;

– возможность зарегистрировать своих детей в лучшие летние лагеря Беларуси;

– право на полную или частичную компенсацию образования, связанного с их профессиональной деятельностью;

– бесплатные курсы английского и немецких языков.

Исходя из всего вышеперечисленного, демократичная культура общения, возможности профессионального развития, яркие корпоративные праздники, многочисленные скидки и бонусы – это все ждёт любого сотрудника *EPAM*.

EPAM способствует улучшению технического образования среди детей и молодёжи посредством программ *EPAM eKids* и *EPAM University*.

Программа *EPAM eKids* предлагает детям изучить основы программирования на базе *Scratch.* Тренерами выступают сотрудники компании, которые проводят занятия в свободное от работы время. В настоящее время программа запущена в 15 странах.

EPAM также тесно сотрудничает с университетами в рамках программы *EPAM University*. С момента запуска программы в 2004 году компания подготовила более 23500 студентов из 40 университетов в шести странах.

В 2019 году образовательные программы *EPAM* были удостоены награды *Global SDG Award* в номинации «Качественное образование». Кроме того, компания получила награду *CEE Shared Services* в номинации «Лучшее сотрудничество университетов и бизнеса» за программу *EPAM University* в Беларуси [3].

1.4 Автоматизация деятельности предприятия

EPAM предоставляет работу тысячам рабочим из разных областей ИТ и обслуживающему персоналу.

Движущей силой предприятия являются разработчики, пишущие код для программного обеспечения различных компаний. Тестировщики программного обеспечения верифицируют этот код, а девопсы разворачивают приложения на различных серверах.

Автоматизация деятельности этих людей ведёт к ускорению и удешевлению процесса разработки, что способствует конкурентоспособности компании.

Основными инструментами автоматизации разработчиков являются интегрированные среды разработки (*IDE*). Интегрированные среды разработки были созданы для того, чтобы максимизировать производительность программиста благодаря тесно связанным компонентам, которые он использует в разработке. Это позволяет разработчику сделать меньше действий для переключения различных режимов, в отличие от дискретных программ разработки. Часто такие среды разработки включают в себя компилятор, функционал автодополнения кода (часто с использованием искусственного интеллекта), инструменты взаимодействия с базой данных, встроенные менеджеры пакетов для управления используемыми модулями приложения, механизм контроля версий и так далее.

Кроме работы над кодом, любому разработчику также необходимы инструменты автоматизации для коммуникаций с коллегами, ведения документов деятельности на предприятии, рабочие пространства, где отображаются выполняемые задачи и многие другие инструменты.

Для коммуникаций используют различные мессенджеры с функционалами видеосвязи и файлообменника, такие как *Skype*, *Teams*, *Webex* и *Zoom*.

Для ведения документов деятельности на предприятии часто используют инструменты компании *HCL* или свои собственные разработки.

Инструментами автоматизации процесса разработки являются *Jira*, *Trello* и многие другие.

Помимо разработчиков, у любого продукта есть менеджеры, которые управляют процессом его разработки и эксплуатации. Работа менеджера связана с огромным количеством общения и таким же объёмом работы с документами.

Каждый месяц создаются продукты для упрощения ведения документов предприятия, среди которых множество платных продуктов, предоставляющих слишком широкий спектр услуг, которые бывают излишними. Из-за этого предприятия создают собственное программное обеспечение, которое может выполнять необходимый для конкретной задачи функционал.

Поэтому в ходе практики задачей было создание такого программного обеспечения, что значительно упростило бы и ускорило ведение документов деятельности предприятия.

2 АНАЛИЗ ИНДИВИДУАЛЬНОЙ ЗАДАЧИ

2.1 Постановка исходной задачи

Индивидуальным заданием, выданным руководителем практики является разработка приложения в области автоматизации деятельности организации.

В любой организации много работы с различными документами, от приказов до различных таблиц и чеков. Весьма часто эти документы совмещаются в один большой *PDF* файл для дальнейшей отправки или печати. Затем их просто помещают в память системы и хранят в таком виде, что очень неудобно для людей, которые далее с этими документами работают.

Индивидуальным заданием является разработка веб-приложения для разделения одного большого документа на различные документы. Шаги реализации поставленной задачи:

– формирование набора данных для тренировки модели машинного обучения;

– формирование набора данных для проверки модели машинного обучения;

– первичный анализ данных;

– выбор и тренировка модели машинного обучения;

– написание веб-сервиса, принимающего запросы пользователя, использующего разработанную модель и возвращающего результат обработки запроса пользователю;

– написание пользовательского интерфейса взаимодействия с веб-сервисом;

– верификация разработанного программного обеспечения.

2.2 Постановка задачи машинного обучения

Основная задача заключается в верном разбиении смешанного файла на различные документы. Однако задача машинного обучения должна быть поставлена более формально.

Было придумано несколько способов решения задачи.

Первым способом является предсказывать является ли текущая страница первой в отдельном документе, если да, то отделять её от тех, что идут раньше. Способ не универсален, ведь есть документы, где каждая страница может быть первой.

Второй способ – предсказывать является ли страница последней в документе, и отделять все следующие. Точно такая же проблема, как и с первым способов.

Третий способ – совместить два предыдущих и смотреть на то является ли предыдущая страница последней и текущая страница первой другого документа. Аналогичная проблема как с первыми двумя. Однако можно также проверять наоборот. Является ли предыдущая первой и текущая последней, если эти оценки также дают положительные результаты значит что-то не так и необходимо сообщить об этом пользователю.

Четвёртый способ кардинально отличается от первых трёх, необходимо проверять, идут ли предыдущая страница и текущая друг за другом, если да, то проходим по документу дальше, если нет делим в этом месте.

Пятым способом является модификация четвёртого способа и добавление к нему проверок третьего способа, однако этот способ является весьма трудоёмким и сложным в настройке.

Из всех пяти возможных вариантов задачи машинного обучения был выбран четвёртый, так как имеет лучшую логику нежели остальные и очень хорошо решается мощными моделями машинного обучения, такими как рекуррентные нейронные сети.

Итого, задача машинного обучения – обучить модель, которая определяет идут ли друг за другом части текста.

2.3 Инструменты исследования моделей машинного обучения

Машинное обучение – это технология, которая помогает приложениям на основе искусственного интеллекта обучаться и выдавать результаты автоматически, без человеческого вмешательства.

При решении задачи машинного обучения необходимо собирать, систематизировать и анализировать данные, а затем на основе полученной информации создавать алгоритмы для искусственного интеллекта.

Для таких задач на данный момент лучше всего подходит язык программирования *python*. Он весьма прост и понятен, более того обладает отличной производительностью при обработке данных.

Также у этого языка есть множество библиотек, которые упрощают процесс написания кода и сокращают время разработки.

Одной из основных библиотек для машинного обучения является *pandas*.

Библиотека *pandas* предназначена для хранения файлов любого формата от *csv* файлов до таблиц баз данных. Кроме того, основным направлением использования этой библиотеки является обработка и анализ данных.

Основные возможности библиотеки *pandas*:

– наличие объекта *DataFrame*, предназначенного для манипулирования индексированными массивами двумерных данных;

– инструменты для обмена данными между структурами в памяти и файлами различных форматов;

– встроенные средства совмещения данных и способы обработки отсутствующей информации

– переформатирование наборов данных, в том числе создание сводных таблиц;

– срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных;

– вставка и удаление столбцов данных;

– слияние и объединение наборов данных;

– иерархическое индексирование позволяет работать с данными высокой размерности в структурах меньшей размерности;

– работа с временными рядами: формирование временных периодов и изменение интервалов.

Библиотека оптимизирована для высокой производительности и работает поверх другой библиотеки – *numpy*, наиболее важные части кода которой написаны на *Cython* и Си.

Numpy – основная библиотека для работы с научными вычислениями. Её преимущество в том, что она позволяет очень быстро совершать операции над массивами данных, например, параллельно перемножать матрицы или вычислять модуль вектора, складывать вектора друг с другом и так далее.

Помимо своей производительности, она включает ряд полезных функций и классов для манипулирования данными.

Кроме библиотеки *numpy,* есть библиотека для более продвинутых научных вычислений – *scipy*. Она более полезна для различных статистических и экономических вычислений.

При анализе данных ключевым звеном является их визуализация. Люди, производящие этот анализ, тяжело воспринимают массивы чисел, поэтому визуализация помогает делать правильные выводы и делиться ими с другими исследователями.

Есть много библиотек для визуализации данных, но самой базовой является *matplotlib*. Она удобна и проста в использовании. Кроме того, имеет множество функций для построения графического материала. Перечень графического материала, который можно делать благодаря этой библиотеке:

– графики;
– диаграммы разброса;
– столбчатые диаграммы и гистограммы;
– круговые диаграммы;
– ствол-лист диаграммы;
– контурные графики;
– поля градиентов;
– спектральные диаграммы.

Основной библиотекой для решения задач классического машинного обучения является *Scikit-learn*. Эта библиотека имеет огромный функционал исследования и преобразования данных. Но ключевой особенностью является наличие всех основных моделей машинного обучения. Кроме того, они имеет отличную оптимизацию и удобство пользования [4].

Ключевое преимущество этой библиотеки – это обширная документация и большое сообщество программистов.

Однако большинство задач с текстом решаются при помощи нейронных сетей. Для обучения нейронных сетей на *python* создан ряд других библиотек, таких как *Keras*, *Pytorch* и *Tensorflow* [5].

Keras – одна из самых простых и удобных библиотек для обучения нейронных сетей. Эта библиотека представляет из себя надстройку над другими библиотеками, однако в последних версиях она работает только с *Tensorflow*. Она содержит многочисленные реализации широко применяемых строительных блоков нейронных сетей, таких как слои, целевые и

передаточные функции, оптимизаторы, и множество инструментов для упрощения работы с изображениями и текстом.

Благодаря этой библиотеке, обучать различные нейронные сети на ядрах графического процессора максимально просто, следовательно, и разработка и исследование моделей происходит намного быстрее.

Самой основной библиотекой для проведения экспериментов с данными и моделями является *IPython* и среда *Jupyter Notebook*.

IPython – интерактивная оболочка для языка программирования *python*, которая предоставляет расширенную интроспекцию, дополнительный командный синтаксис, подсветку кода и автоматическое дополнение.

Jupyter Notebook – это интерактивная вычислительная среда на базе Интернета для создания документов *ipynb*.

Документ *Jupyter Notebook* – это документ *JSON*, содержащий упорядоченный список ячеек ввода и вывода, которые могут содержать код, текст (с использованием *Markdown*), математику, графики и мультимедийные данные.

Основным преимуществом этого инструмента является интерактивность. Разработчик пишет часть нужного ему кода, и он выполняется. Затем делает выводы по результату и пишет код дальше. Это намного удобнее для исследований чем использование обычных *python* скриптов.

Внешний вид этого инструмента изображён на рисунке 2.1.

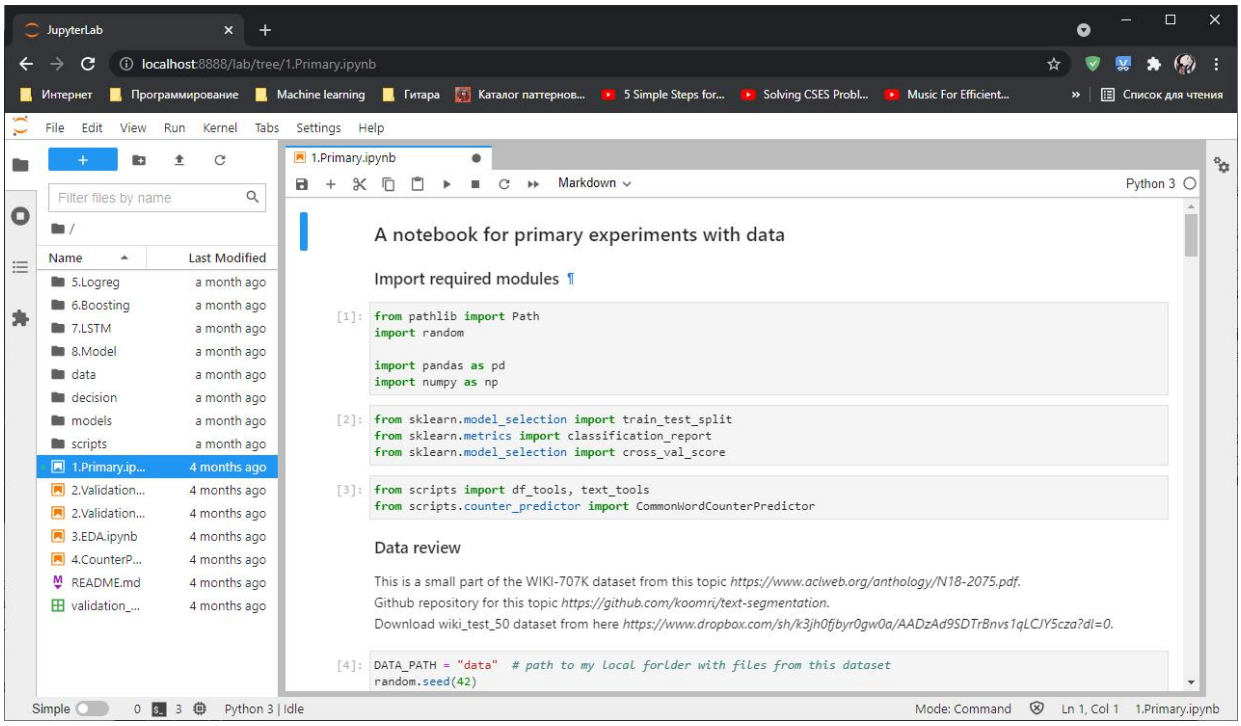

Рисунок 2.1 – Внешний вид *Jupyter Notebook*

Так как поставленной задачей является обработка текста, необходимы инструментами являются *spacy* и *nltk*. Это библиотеки для продвинутой обработки текста на *python*.

Обе библиотеки весьма похожи, однако *nltk* ориентируется на обучение и исследования естественного языка, а *spacy* является производственным инструментом. Также *spacy* позволяет писать собственные конвейеры обработки естественного языка с использованием глубокого обучения.

Основные характеристики *spacy*:

– неразрушающая токенизация;

– встроенная поддержка обучаемых компонентов конвейера, таких как распознавание именованных сущностей, теги частей речи, синтаксический анализ зависимостей, классификация текста, связывание сущностей;

– статистические модели для 17 языков;

– поддержка пользовательских моделей других библиотек;

– встроенные визуализаторы;

– простая упаковка готовой модели, развёртывание и управление рабочим процессом.

Кроме библиотек на языке *python* одним из важнейших инструментов для обучения нейронных сетей является *google colab*. Это бесплатный сервис от компании *google*, который позволяет любому пользователю использовать графические ускорители компании для обучения нейронных сетей.

Он предоставляет собственную версию рабочего пространства, похожего на *jupyter notebook* с некоторыми дополнительными преимуществами:

– автодополнение кода пользователя;

– интеграция с аккаунтом *google*;

– удалённая рабочая машина;

– облачное хранение данных.

После обзора всех основных инструментов для исследования моделей машинного обучения были выбраны следующие:

– *jupyter notebook* в качестве исследовательской среды;

– *pandas* для хранения и обработки данных;

– *numpy* для математических операций;

– *scikit-learn* как инструмент создания моделей классического машинного обучения;

– *keras* для создания нейронных сетей;

– *spacy* и *nltk* для обработки естественного языка;

– *google colab* как инструмент для обучения тяжёлых нейронных сетей.

Кроме этих инструментов, в работе также применялось и много более мелких и специфичных инструментов: *pathlib, re, pickle, os, sys, zipfile, csv* [6].

2.4 Средства реализации веб-сервиса

Для реализации веб-сервиса, который будет использовать обученную модель и обрабатывать файлы, полученные от пользователя, был выбран *python* фреймворк *flask*.

Flask – фреймворк для создания веб-приложений на языке программирования *Python*, использующий набор инструментов *Werkzeug*, а также шаблонизатор *Jinja2*. Относится к категории так называемых

микрофреймворков – минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

Несмотря на то, что *flask* является полноценным фреймворком для написания веб-приложений в проекте, он использован в качестве библиотеки для написания микросервисов [7].

Есть несколько подходов для написания микросервисов, самым популярным является *REST*.

REST (от англ. *Representational State Transfer* – «передача состояния представления») – архитектурный стиль взаимодействия компонентов распределённого приложения в сети. *REST* представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой гипермедиа-системы.

В определённых случаях (интернет-магазины, поисковые системы, прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в *REST* взаимодействуют наподобие взаимодействия клиентов и серверов во Всемирной паутине. Для веб-служб, построенных с учётом *REST*, применяют термин «*RESTful*».

В сети Интернет вызов удалённой процедуры при помощи *REST* представляет собой обычный *HTTP*-запрос, а необходимые данные передаются в качестве параметров запроса. После этого этот запрос обрабатывается сервисом и возвращает результат обработки клиенту. Результатом обработки могут быть различные данные, например: двоичные файлы, *JSON* файлы, *HTML* страницы [8].

Преимущества данного подхода:

– надёжность, которая достигается за счёт отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна;

– производительность за счёт использования кеширования;

– масштабируемость;

– прозрачность системы взаимодействия;

– простота интерфейсов взаимодействия;

– портативность компонентов;

– лёгкость внесения изменений.

Обращение к REST сервисам может осуществляться с любого устройства, например из браузера или мобильного телефона. Достаточно указать домен, к которому будет идти обращение, адрес получаемого ресурса, метод получения и параметры.

В данном проекте для реализации интерфейса пользователя был выбран веб-интерфейс, это значит, что пользователь будет взаимодействовать с сервисами через браузер.

2.5 Средства реализации веб-интерфейса

Существует много способов и инструментов для создания веб-интерфейса пользователя. Одним из способов является написание простых *HTML* страниц с

Javascript кодом, который осуществляет запросы к веб-сервису. Однако этот способ является весьма затратным по времени и совершенно не гибким.

Другой способ реализации веб-интерфейса – это создание веб-приложения при помощи веб-фреймворков. Одним из самых популярных фреймворков для создания веб-приложений является *Angular*. Этот фреймворк базируется на платформе *Node.js* [9].

Этот фреймворк был написан командой разработчиков *google* на *typescript* – языке от компании *microsoft*. Он является средой проектирования веб-приложений и платформой разработки для создания эффективных и сложных одностраничных приложений. Он может использоваться как для написания простых приложений, так и для целых корпоративных проектов.

В качестве платформы *Angular* включает:

– компонентный фреймворк для создания масштабируемых приложений;

– коллекцию хорошо интегрированных библиотек, охватывающих широкий спектр функций, включая маршрутизацию, управление формами, связь клиент-сервер и так далее;

– набор инструментов разработчика, которые помогают разрабатывать, тестировать и расширять код.

Благодаря низкой связности компонентов *Angular* приложения очень гибкие и масштабируемые.

Есть чёткое разграничение различных элементов фреймворка, например: сервисы используются для логики веб-приложения, а компоненты совершают динамическую обработку *HTML* страниц и используют эти сервисы.

Компонентная модель Angular предлагает сильную инкапсуляцию и интуитивно понятную структуру приложения. Компоненты также делают модульное тестирование более удобным и улучшают общую читаемость кода.

После того как были разобраны все инструменты для решения поставленной задачи, была начата разработка приложения.

3 РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

3.1 Создание набора данных для задачи машинного обучения

Задачей машинного обучения в данном проекте является предсказание идут ли две части текста друг за другом.

Для этой задачи было найдено несколько наборов данных, которые после обработки станут подходить под текущую задачу. Самым большим из них является набор *WIKI-70K*.

Этот набор был собран преподавателями университета *Tel-Aviv* для задачи сегментации текста. Он представляет собой статьи, выбранные из википедии, в которых были помечены места объединения разделов. Внешний вид файла из набора данных изображён на рисунке 3.1.

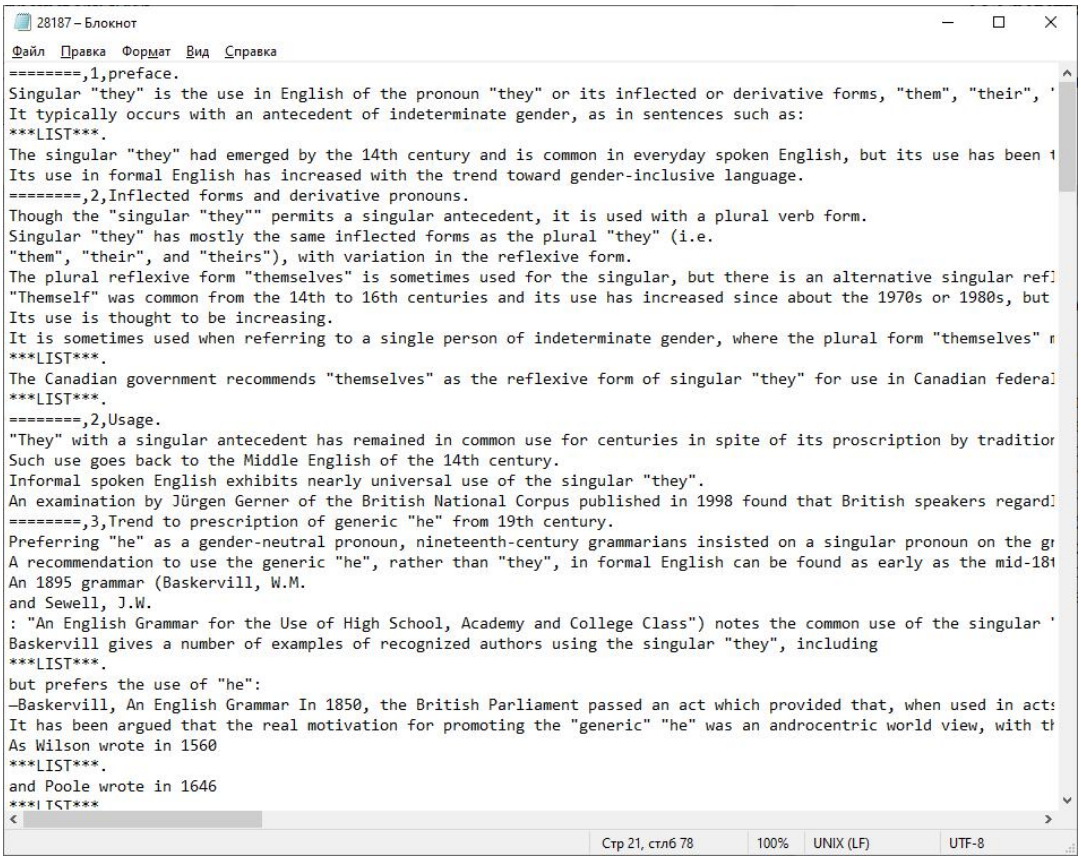

Рисунок 3.1 – Файл набора данных

Для обработки набора данных был написан алгоритм, который проходится по файлам и по статьям в них. Во время прохождения создаёт строку таблицы, состоящую из трёх столбцов, при этом записывает в первый столбец предыдущую статью, а во второй текущую.

Если статьи из одного документа, то устанавливает в третий столбец положительное значение, что означает что две части текста идут друг за другом. Если это две статьи из разных документов, то устанавливает отрицательное значение.

Кроме сохранения информации порядка, для каждой части текста в строке устанавливается номер документа, из которого эта часть была получена. Вид обработанного набора данных изображён на рисунке 3.2.

	first_part	first_doc_id	second_part	second_doc_id	is_in_order
0	[START]	-1	David Matas (born 29 August 1943) is the senio...	0	False
1	David Matas (born 29 August 1943) is the senio...	0	David Matas was born in Winnipeg; his grandpar...	0	True
2	David Matas was born in Winnipeg; his grandpar...	0	Matas served as a Law Clerk to the Chief Justi...	0	True
3	Matas served as a Law Clerk to the Chief Justi...	0	He ran for the Canadian House of Commons in th...	0	True
4	He ran for the Canadian House of Commons in th...	0	He has been actively involved as Director of t...	0	True
5	He has been actively involved as Director of t...	0	In 2006, with David Kilgour he released the Ki...	0	True

Рисунок 3.2 – Обработанный набор данных

Однако такой набор данных сильно смещён в сторону идущих друг за другом статей, то есть статей с положительной меткой будет намного больше, чем с отрицательной. Когда дело касается бинарной классификации, чем и является исходная задача, желательно уравнять получаемые классы.

Поэтому для дальнейшей обработки был написан очередной алгоритм, выполняющий генерацию строк с отрицательной меткой. Для генерации таких строк алгоритм берет части текста из разных документов, о чём свидетельствует уникальный номер документа части текста, объединяет эти части в одну строку и записывает отрицательную метку, после чего добавляет эту строку в набор данных. При этом алгоритм избегает появления дубликатов и перемешивает получившийся набора данных.

После такой обработки текста полученный набор данных был проверен при помощи базовой модель, которая показывает работоспособность этого набора и является нижней оценкой более совершенных моделей.

Базовая модель представляет из себя обычный алгоритм, который вычисляет количество одинаковых слов в обоих текстах и в зависимости от количества предсказывает метку. Если одинаковых слов больше двух, то метка положительная, иначе отрицательная.

Для данной модели необязательно создавать проверочный набор, так как эту модель не нужно обучать. Благодаря этому можно проверить качество модели на тренировочном наборе данных.

Полученная оценка классификации изображена на рисунке 3.3.

	precision	recall	f1-score	support
0	0.77	0.83	0.80	109
1	0.78	0.71	0.74	93
accuracy			0.77	202
macro avg	0.77	0.77	0.77	202
weighted avg	0.77	0.77	0.77	202

Рисунок 3.3 – Оценка базовой модели на случайной выборке

Из полученных результатов видно, что доля верных ответов составляет 77 процентов, что является неплохим результатом для такой простой модели.

Также если посмотреть на дополнительные характеристики видно, что отрицательные метки модель предсказывает лучше, чем положительные, хотя особых выводов из этого делать не стоит. Однако уже из этого стало понятно, что набор данных не является неадекватным для решения поставленной задачи и с ним можно работать.

После проверки тренировочного набора был сделан проверочный набор из других документов. Этот набор был закреплён для более точной оценки качества других моделей, то есть он был использован для проверки всех обученных моделей.

Количество записей в этом наборе равняется 10009, где 5005 записей – строки с положительной меткой, то есть части текста идущие по порядку, 5004 записи – строки с отрицательной меткой. Кроме того, из проверочного набора были удалены все дубликаты, которые могли бы помешать адекватной оценке работы моделей.

После создания набора был написан скрипт, выполняющий валидацию любой модели на созданном наборе данных. Первой моделью, качество которой было проверено на этом наборе данных была базовая модель. Оценка базовой модели на проверочном наборе изображена на рисунке 3.4.

	precision	recall	f1-score	support
0	0.76	0.82	0.79	5004
1	0.80	0.74	0.77	5005
accuracy			0.78	10009
macro avg	0.78	0.78	0.78	10009
weighted avg	0.78	0.78	0.78	10009

Рисунок 3.4 – Оценка базовой модели на тренировочном наборе

Полученные результаты были использованы для оценки более сложных моделей. Если обученная модель даёт результаты хуже базовой, она признаётся не релевантной.

Следующим этапом исследования было обучение более сложных моделей.

3.2 Выбор модели машинного обучения

3.2.1 Первой используемой моделью стала логистическая регрессия. Логистическая регрессия – это статистическая модель, используемая для прогнозирования вероятности возникновения некоторого события путём его сравнения с логистической кривой. Эта регрессия выдаёт ответ в виде вероятности двоичного события (1 или 0).

Любая модель машинного обучения принимает для предсказания массив чисел. Для того чтобы перейти от текстов к числам, был использован инструмент для создания векторного представления текста *TFIDF*.

TFIDF – статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

Сперва этот инструмент необходимо обучить на всех текстах из обучающего набора. После обучения он запомнит все используемые слова и назначит им значение веса в зависимости от частоты появления слова в текстах.

После применения этого инструмента к тексту создаётся вектор, размер которого равен количеству всех слов во всех текстах, а на места чисел в векторе, которые представляют конкретное слово из текста ставится значение веса этого слова. Если таких слов несколько значения суммируются.

Как и в любом инструменте машинного обучения в *TFIDF* существует множество параметров, которые необходимо настраивать и смотреть на качество итоговой модели в зависимости от этих параметров. Такими параметрами являются:

– стоп-слова – слова, которые необходимо игнорировать при эксплуатации инструмента;

– максимальное количество запомненных слов;

– приводить ли к нижнему регистру все слова и так далее.

В зависимости от этих параметров модель меняет свою сложность, что может сказаться на её качестве как в лучшую, так и в худшую стороны.

Итак, после преобразования двух частей текста, получается два вектора, однако модель принимает всего один. Из-за этого следующим этапом было определение как эти два вектора друг с другом совместить.

Есть не так много операций с векторами, были выбраны две из них. Первой операцией была конкатенация векторов, а второй – произведение. Эти две операции также были гиперпараметрами модели, которые было необходимо настроить.

После такого преобразования можно применять модель, в данном случае логистическую регрессию.

У логистической регрессии также много параметров, основными из которых являются:

– используемая регуляризация;

– максимальное количество итераций обучения;

– коэффициент регуляризации.

Из всех этих инструментов был создан конвейер обработки данных. Далее для оценки и настройки модели был использован механизм перекрёстной проверки *k-fold*. Перекрёстная проверка – это процедура повторной выборки, используемая для оценки моделей машинного обучения на ограниченной выборке данных.

Процедура перекрёстной проверки заключается в следующем:

– набор данных перемешивается случайным образом;

– набор данных делится на k групп;

– для каждой уникальной группы выбирается подгруппа для тестирования набора данных, а остальные подгруппы являются данными для обучения;

– модель обучается на данных для обучения и проверяется на тестовой выборке после чего сохраняются её параметры и оценка;

После такой процедуры можно найти такие параметры конвейера модели, при которых она выдаёт лучший результат.

Лучшие параметры логистической регрессии и её оценка на перекрёстной проверке изображены на рисунке 3.5.

Look at the best parameters and score

```
grid.best_params_
```

```
{'classifier__C': 100000.0,
 'classifier__max_iter': 500,
 'classifier__penalty': 'l2',
 'classifier__random_state': 42,
 'vectorizer__lowercase': True,
 'vectorizer__max_features': 100000,
 'vectorizer__ngram_range': (1, 1),
 'vectorizer__stop_words': None,
 'vectorizer__type': 'multiply'}
```

```
grid.best_score_
```

```
0.9168566029649542
```

Рисунок 3.5 – Лучшие параметры и оценка модели

Оценка модели на перекрёстной проверке оказалась очень оптимистичной. Произошёл огромный прирост в качестве относительно базовой модели. Для более несмещённой оценки модель была проверена на оставленном тестовом наборе данных. Результат проверки изображён на рисунке 3.6.

	precision	recall	f1-score	support
0	0.83	0.95	0.89	5004
1	0.94	0.81	0.87	5005
accuracy			0.88	10009
macro avg	0.89	0.88	0.88	10009
weighted avg	0.89	0.88	0.88	10009

Рисунок 3.6 – Результат проверки логистической регрессии на тестовом наборе данных

Отсюда видно, что несмещённая оценка модели равна 0.88, что также является хорошим результатом относительно базовой модели.

3.2.2 Следующей мощной проверенной моделью является градиентный бустинг на основе деревьев решений.

Они представляют собой иерархические древовидные структуры, состоящие из решающих правил вида «Если ..., то ...». Правила автоматически генерируются в процессе обучения на обучающем множестве и, поскольку они формулируются практически на естественном языке (например, «Если объём продаж более 1000 шт., то товар перспективный»), деревья решений как аналитические модели более вербализуемы и интерпретируемы, чем нейронные сети.

Однако деревья решений сами по себе являются весьма слабыми алгоритмами, поэтому их принято объединять во множества деревьев, которые подчиняются определённым правилам. Такие множества алгоритмов машинного обучения называются ансамблями.

Бустинг – это техника построения ансамблей, в которой предсказатели построены не независимо, а последовательно

Это техника использует идею о том, что следующая модель будет учится на ошибках предыдущей. Они имеют неравную вероятность появления в последующих моделях, и чаще появятся те, что дают наибольшую ошибку. Предсказатели могут быть выбраны из широкого ассортимента моделей, например, как в данном случае, деревья решений. Из-за того, что предсказатели обучаются на ошибках, совершенных предыдущими, требуется меньше времени для того, чтобы добраться до реального ответа. Но критерий остановки выбирается с осторожностью, иначе это может привести к переобучению.

Градиентный бустинг также как и логистическая регрессия принимает вектор чисел.

В данном случае было принято решение заняться разработкой признаков, а не применением алгоритмов векторизации текста.

Было просмотрено множество признаков текстов. Для каждого была построена гистограмма частот, показывающая насколько хорошо признак разделяет метки в данных.

Основным признаком было количество одинаковых слов в частях текста. Гистограмма частот изображена на рисунке 3.7.

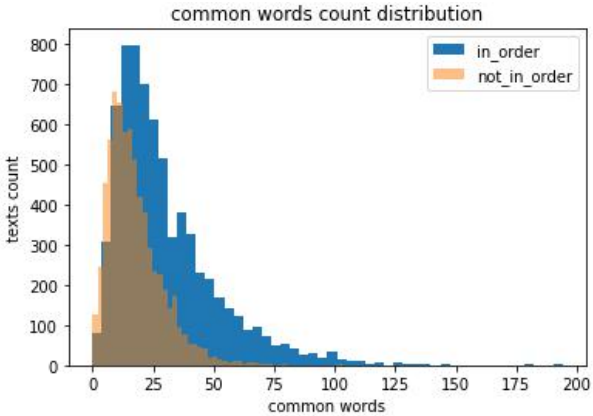

Рисунок 3.7 – Гистограмма частот для признака общих слов

Следующим значимым признаком был коэффициент Отиаи.

Коэффициент Отиаи – двоичная мера сходства, предложенная японским биологом Акирой Отиаи в 1957 году, в дальнейшем обобщённая и нашедшая применение в разнообразных приложениях и за рамками биологии.

Гистограмма частот для этого коэффициента изображена на рисунке 3.8.

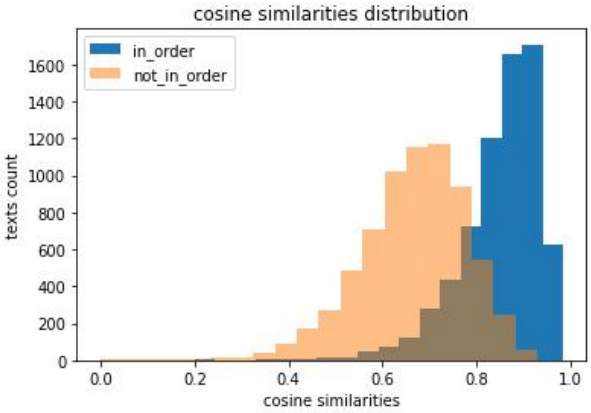

Рисунок 3.8 – Гистограмма частот для коэффициента Отиаи

Из рисунков 3.7 и 3.8 видно, что эти признаки хорошо разделяют элементы набора данных на те, которые идут за другом либо не идут. Идущие за другом помеченные синим цветом, не идущие – оранжевым.

Снизу на диаграмме изображено числовое значение признака, а слева количество текстов для определённого значения признака.

Для первого признака ключевым элементов было слово. Для второго – целый текст. Однако кроме этих единиц очень важными признаками являются количество общих комбинаций слов в текстах. Такие комбинации называются *n*–граммы и представляют собой множества, состоящие из *n* слов, идущих друг за другом.

На рисунке 3.9 изображено как набор данных разделяют биграммы и триграммы слева направо.

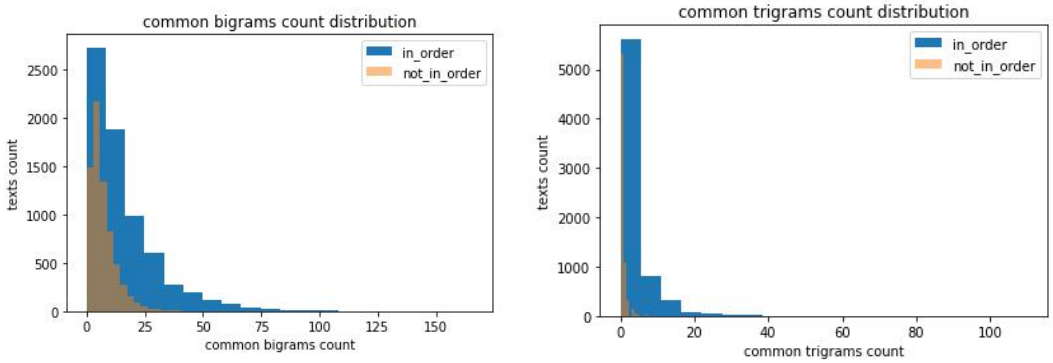

Рисунок 3.9 – Гистограммы частот для биграмм и триграмм

Как видно из рисунка, практически нет комбинаций текстов, где есть общие триграммы, но при этом они идут не друг за другом.

После этого был разработан признак, который показывает количество общих редких слов, которые встречаются реже чем среднее значение встречаемых слов в текстах.

Кроме этих признаков, были исследованы и другие, однако они не дали такого хорошего разделения данных.

Далее для каждой пары текстов были созданы признаки и записаны в таблицу. Внешний вид значений разработанных признаков изображён на рисунке 3.10.

	same_words	cossim	same_bigrams	same_trigrams	rare_common_entities
307	7	0.906825	3	0	1
5117	18	0.887478	20	10	3
6538	8	0.866975	16	5	2
12568	2	0.721436	4	0	2
7977	1	0.498309	0	0	3

Рисунок 3.10 – Разработанные признаки

Именно на этом наборе данных и был обучен бустинг. Для получения оптимальных гиперпараметров был также использован механизм перекрёстной проверки. Проверяемыми параметрами были:

– максимальное количество листьев дерева;

– минимальное количество экземпляров в одном листе;

– максимальная глубина дерева;

– максимальное количество деревьев;

– скорость обучения;

– максимальное количество итераций обучения.

После обучения и настройки параметров, были получены лучшие параметры и результаты модели на тренировочном и тестовом наборах. Результаты лучшей модели на тестовом наборе данных изображены на рисунке 3.11.

	precision	recall	f1-score	support
0	0.86	0.91	0.88	5004
1	0.90	0.85	0.88	5005
accuracy			0.88	10009
macro avg	0.88	0.88	0.88	10009
weighted avg	0.88	0.88	0.88	10009

Рисунок 3.11 – Результаты бустинга на тестовом наборе

Из рисунка 3.11 видно, что качество бустинга такое же как у регрессии.

Из этого можно сделать некоторые выводы. Вполне вероятно, что модель слишком простая для такой задачи и не может правильно обобщать результаты. Возможно, разработанных признаков недостаточно для более точного прогноза или в них находится шум, который уменьшает оценку модели. Но что чаще всего бывает, вполне вероятна проблема с данными. Первым делом была выбрана модель посложнее.

3.2.3 Чаще всего для работы с текстом применяются именно нейронные сети. Применяют глубокие и свёрточные нейронные сети, однако фаворитом являются рекуррентные сети.

Рекуррентные нейронные сети (*RNN*) – вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки. В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому сети *RNN* применимы в таких задачах, где нечто целостное разбито на части, например: распознавание рукописного текста или распознавание речи. Было предложено много различных архитектурных решений для рекуррентных сетей от простых до сложных. В последнее время наибольшее распространение получили сеть с долговременной и кратковременной памятью (*LSTM*) и управляемый рекуррентный блок (*GRU*).

Было решено применять именно *LSTM* в виду простой реализации в библиотеке *keras* и обширной документации [10].

Кроме выбора модели, также была добавлена предобработка данных:

– приведение всех слов к нижнему регистру;

– преобразование текста, такое как: удаление нестандартных символов, замена сокращений на полные формы, замена сленга на обычные слова и так далее;

– удаление стоп-слов.

Так как нейронная сеть также работает с числами, необходимо преобразовать последовательности слов в последовательности чисел. Для этого был использован инструмент *tokenizer*. Этот инструмент просто запоминает все слова в текстах и потом преобразовывает слова в последовательности в их численные эквиваленты. Если в последовательности есть слова, которые он не запомнил, просто устанавливается ноль на место этих слов.

Кроме того, что последовательности должны быть числовыми, они также должны быть одной длины. Для данной задачи было предложено использовать 100 последних слов для первой части текста и 100 первых слов для второй части текста. Если в тексте содержится меньше слов, то оставшееся место также заполняется нулями. Поэтому, если в сравниваемых текстах будет мало слов, точность классификации будет понижаться. Для того чтобы бороться с этим был создан дополнительный самый мощный признак – количество одинаковых слов в частях текста. Благодаря ему даже в небольших последовательностях если у них есть общие слова, классификатор может склоняться в сторону положительной классификации.

У нейронных сетей также есть настраиваемые параметры:

– количество циклов слоя памяти;

– количество нейронов на полносвязном слое;

– количество неактивных нейронов на каждом цикле слоя памяти;

– количество неактивных нейронов на одной итерации обучения полносвязного слоя;

– функция активации полносвязного слоя.

В ходе экспериментов были получены следующие оптимальные параметры: 192 цикла слоя памяти, 128 нейронов полносвязного слоя, 40 процентов неактивных нейронов в цикле слоя памяти, 40 процентов неактивных нейронов для полносвязного слоя, линейная функция активации *ReLU*.

Обучение сети остановилось на 72 эпохе, лучший результат был получен на 65 эпохе, после него семь эпох не было увеличения качества. На 65 эпохе качество модели на тренировочных данных было равно 0.9226, на проверочных – 0.9266, что свидетельствует о качестве полученной модели, так как она умеет обобщать данные и не переучилась на тренировочном наборе.

В результате проверки модели на тестовых данных были получены результаты, изображённые на рисунке 3.12.

	precision	recall	f1-score	support
0	0.94	0.89	0.91	5004
1	0.89	0.94	0.92	5005
accuracy			0.91	10009
macro avg	0.92	0.91	0.91	10009
weighted avg	0.92	0.91	0.91	10009

Рисунок 3.12 – Результат проверки нейронной сети на тестовых данных

Из рисунка 3.12 видно, что доля верных ответов увеличилась на 0.03 относительно прошлой модели, что является очень хорошим увеличением. Остальные 0.09 качества связаны с тем, что некоторые краевые случаи не были учтены. Однако такого качества классификации для работы оказалось вполне достаточно и в результате была выбрана эта модель как решение задачи машинного обучения.

3.2.4 После решения задачи машинного обучения было необходимо вернуться к исходной задаче, а именно разделение смешанного документа на части. Для проверки модели были использованы новости из *bbc news*. Для первоначальной проверки был собран набор данных из 5 новостей, смешанных в один документ. Модель верно разделила все 5 документов, что является хорошим результатом, хоть и на таком небольшом наборе. После такой первоначальной проверки модели была начата работа над следующим этапом разработки проекта.

3.3 Структура разработанного веб-сервиса

Для того чтобы использовать полученную модель был написан веб-сервис на *python* фреймворке *flask*.

Было создано четыре модуля с корневым модулем *app.py*, который является входной точкой в приложение.

Для использования модели был написан модуль *model_wrapper.py*, который представляет собой обёртку вокруг этой модели. Этот модуль выполняет инициализацию всех необходимых инструментов, настройку параметров и путей, загрузку модели нейронной сети и токенайзера и предоставляет для других модулей класс *Model*, экземпляр которого может предсказывать метку между двумя частями текста.

Для решения конкретной задачи разделения документа был написан модуль *splitter*. Он объявляет класс *Splitter*, который:

– инициализирует модель;

– получает путь *pdf* файла как параметр метода;

– открывает *pdf* файл, разделяет на страницы, считывает текст этих страниц;

– проходит по каждой комбинации последовательных страниц и при помощи модели ставит метки, и если метка является положительной делит *pdf* документ в этом месте;

– все разделённые документы сохраняет в папке с именем первоначального документа.

Для тестирования функционала был написан модуль *tests.py,* который позволил проверять функционал приложения без конкретного обращения пользователя к нему.

И в заключение был написан модуль, являющийся входной точкой в приложение – *app.py*. Этот модуль выполняет инициализацию всех необходимых модулей, в том числе *flask* и *splitter*. Кроме того, инициализирует веб-приложение и подключает возможность кросс-доменных запросов.

Были написаны два *REST* метода.

Первый метод – */process*. Метод является *post* методом, к нему пользователь обращается, когда загружает свой файл на обработку.

Этот метод получает файл из отправляемой формы, применяет к нему разделитель документа и сохраняет все разделённые документы в хранилище в формате архива.

Второй метод – */download*. Метод является *get* методом. Пользователь обращается к приложению и передаёт имя файла, которое он хочет получить. Метод ищет этот файл и отправляет пользователю архив в случае нахождения, иначе отправляет сообщение о неудаче поиска.

Эти методы помечены декораторами модуля *flask*. Когда происходит запуск приложения, *flask* ищет такие декораторы и выстраивает маршрут обращения к приложению. После выстраивания маршрута приложение запускается и начинает обрабатывать запросы пользователя.

3.4 Структура разработанного веб-интерфейса

Для того чтобы пользователь мог удобно взаимодействовать с разработанным веб-сервисом, был написан *Angular* веб-интерфейс.

Программа на этом фреймворке представляет собой совокупность компонентов, сервисов, страниц разметки и стилей.

В проекте содержатся две папки и множество файлов конфигурации.

Папка *node_modules* хранит все модули, подключённые в приложении. Файлы конфигурации содержат различные настройки приложения, от типа развёртывания до версий подключённых модулей. Главной папкой для разработчика является папка *src* – именно здесь находится весь исходный код приложения, в том числе и точка входа в приложение.

В разработанной программе точкой входа является файл *main.ts*, написанный на языке *typescript*. Этот язык используется для разработки, однако при сборке проекта он трансформируется в *javascript*.

В главном файле подключаются все основные модули и запускается главный модуль приложения *app.module*. Кроме главного модуля приложения в папке с этим файлом, находится главная *html* страница приложения – *index.html*, глобальные стили приложения – *styles.css*, полифилы приложения и файл конфигурации тестов.

Внутри папки с точкой входа находятся ещё три папки:

– *app* – здесь находится все компоненты, модули и сервисы приложения;

– *assets* – тут находятся ресурсы приложения;

– *environments* – в этой папке находятся настройки развёртывания приложения.

После того, как код в файле *main.ts* запускается, начинает выполнение главный модуль приложения *app.module*. В нём подключаются все необходимые модули, компоненты и сервисы, и, при помощи механизма внедрения зависимостей, передаются во все элементы приложения.

Этот модуль запускает главный компонент приложения *app.component.ts*, который определяет заголовок, разметку и стили главной страницы приложения, которые находятся в той же папке.

Кроме главных модуля и компонента в этой папке находятся сервисы, используемые в приложении, и модуль маршрутизации.

Модуль маршрутизации *app-routing.module* определяет все маршруты приложения и позволяет совершать навигацию в нём. В нём находится объявление всех допустимых маршрутов.

Единственным сервисом приложения является *file.service.ts*. Этот сервис отвечает за взаимодействие с веб-сервисом, написанным на *flask*. Именно он определяет методы обращения к веб-сервису и их параметры.

Маршрутизация приложения происходит при помощи тегов, встраиваемых в *html* страницы. Таким образом, в файле разметки главного компонента приложения находится два тега, ссылающихся на другие компоненты.

Первый тег ссылается на компонент *navigation.component.ts*. Он представляет собой верхнее меню навигации приложения. Весь его код содержится в папке *navigation*.

По сути, когда происходит сборка приложения, этот компонент встраивается на главную страницу и добавляет меню навигации с ссылками на другие компоненты приложения.

Второй тег является тегом модуля навигации, именно на его место будет подставляться текущий компонент, полученный в результате навигации пользователя по страницам.

Модуль навигации объявляет два пути, первый из которых используется по умолчанию, когда в строку запроса передан пустой путь. Этот путь ведёт к компоненту *home.component.ts*. Второй путь используется, когда в строку запроса передана строка */pdf*. Он ведёт к компоненту *pdf-splitter.component.ts*.

Компонент *home.component.ts* является компонентом, отвечающим за отображение первичной информации пользователю. В его файле разметки помещено сообщение о функционале приложения.

Файл *pdf-splitter.component.ts* является компонентом, отвечающий за загрузку документа и получения обработанного архива с разделёнными документами.

В его файле разметки содержится форма обработки запроса пользователя, а в коде содержится логики взаимодействия пользователя с приложением при помощи действий на этой форме.

При нажатии на кнопки на этой форме пользователь совершает операции по загрузке файлов на сервер, которым является локальный сервер с веб-сервисом и по получению обработанных файлов.

Именно этот компонент обращается к *REST* методам веб-сервиса, которые в свою очередь обращаются к модели и выполняют поставленную задачу.

Следующим этапом после написания приложения является проверка – решает ли оно поставленную задачу и насколько хорошо оно это делает.

4 ВЕРИФИКАЦИЯ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

4.1 Пример решения задачи разработанным приложением

Разработанный веб-сервис и веб-интерфейс находятся на локальной машине.

Для того чтобы запустить веб-сервис в папке с проектом, необходимо запустить команду *python -m flask run*. Путь к веб-сервису имеет адрес локальной машины и порт 5000.

Кроме разработанного веб-интерфейса, к веб-сервису можно обращаться вручную при помощи различных инструментов, таких как *postman*. Этот инструмент позволяет делать к любым адресным строкам любые *REST* запросы с параметрами. Это очень удобно для тестирования сервисов, особенно когда разработка веб-интерфейса ещё не закончена.

После запуска веб-приложения необходимо запустить приложение веб-интерфейса. Чтобы это его запустить в папке с проектом, надо вызвать командную панель и прописать там команду *ng serve*. Эта команда обращается к фреймворку *Angular* и разворачивает приложение.

Адрес построенного приложения интерфейса также является локальным, порт – 4200. При переходе из браузера по этому пути приложение возвращает главную страницу. Главная страница приложения изображена на рисунке 4.1.

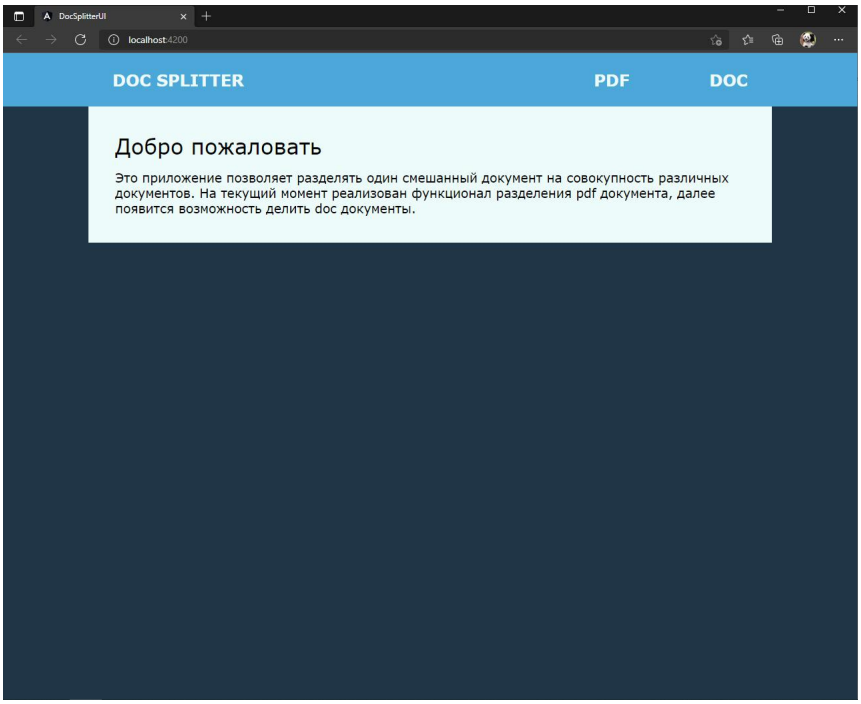

Рисунок 4.1 – Главная страница приложения

Основной функционал разработанного приложения находится во вкладке *PDF*, которая находится на панели навигации сверху.

При нажатии на эту вкладку происходит переход по пути */pdf*, который связан с компонентом, который обрабатывает *pdf* файлы пользователя.

Результат перехода на эту страницу изображён на рисунке 4.2.

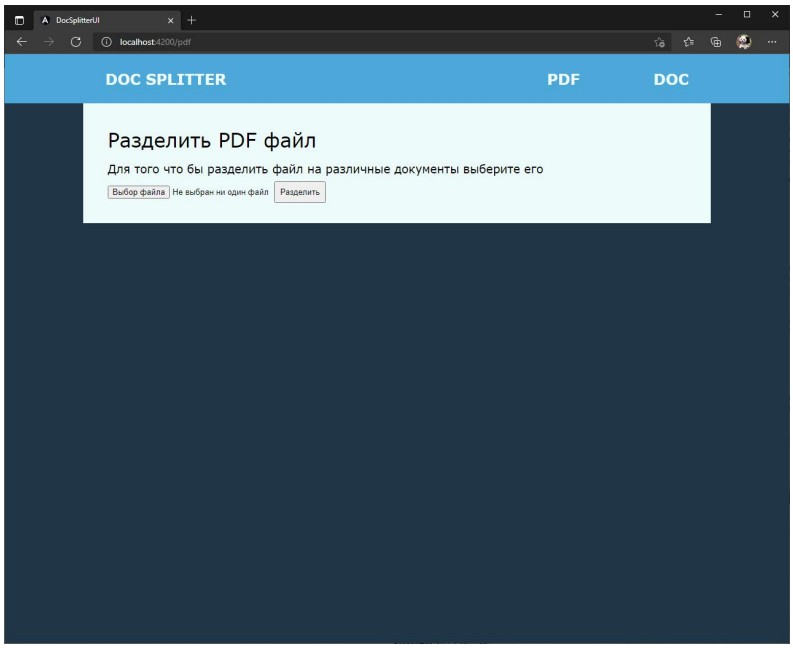

Рисунок 4.2 – Результат перехода на страницу *PDF*

На этой странице пользователю предлагается загрузить свой *pdf* файл на обработку. Окно выбора файла изображено на рисунке 4.3.

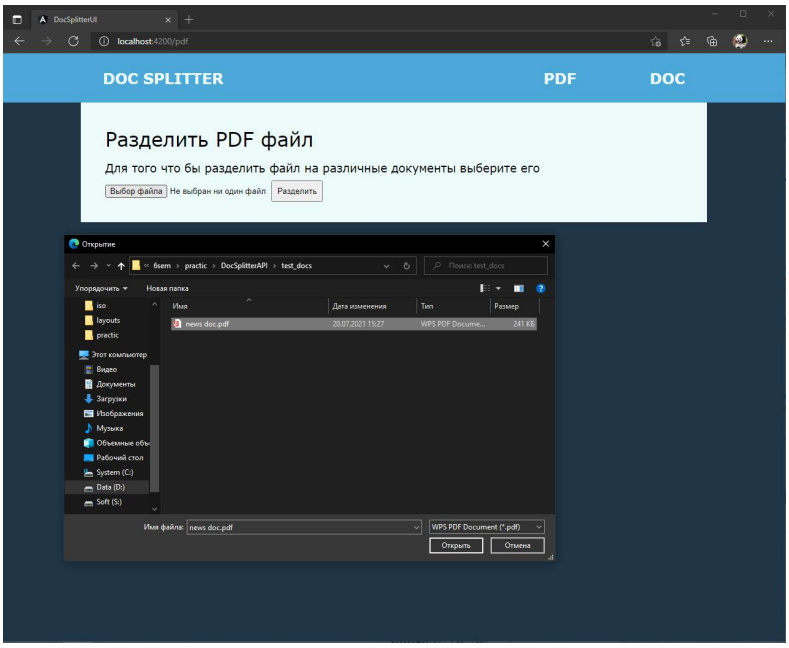

Рисунок 4.3 – Окно выбора загружаемого файла

После выбора файла на странице отображается имя выбранного файла. Чтобы начать обработку файла, необходимо нажать на кнопку «Разделить». Во время обработки файла пользователю будет выведена информация о том, что файл ещё обрабатывается. После окончания обработки документа, страница

обновляется и уведомляет пользователя о том, что обработка завершена и файл можно скачать, нажав на кнопку «Скачать». Страница при обработке и после завершения обработки изображена на рисунках 4.4 и 4.5.

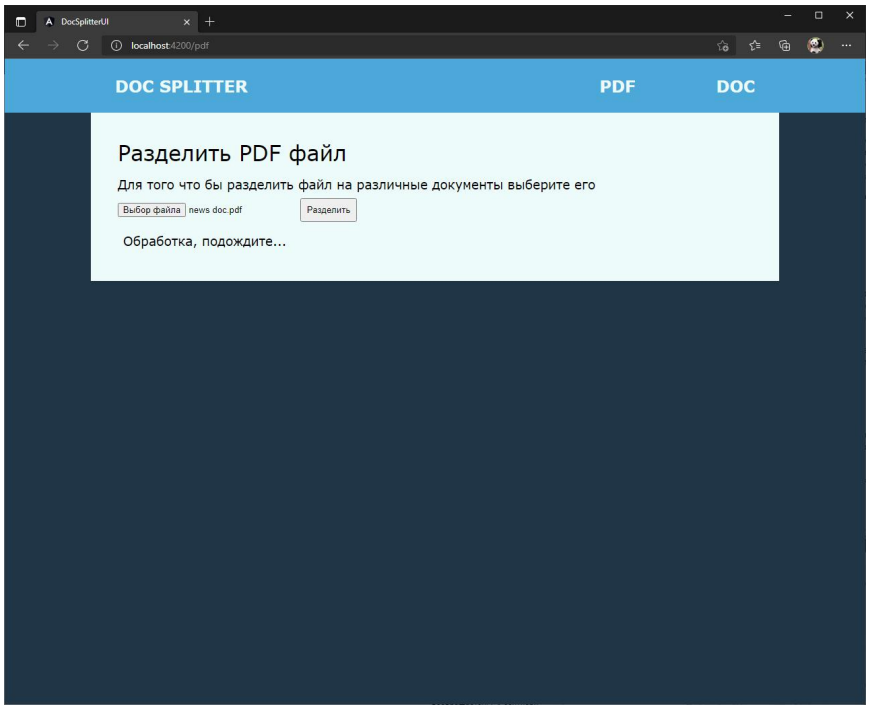

Рисунок 4.4 – Страница во время обработки файла

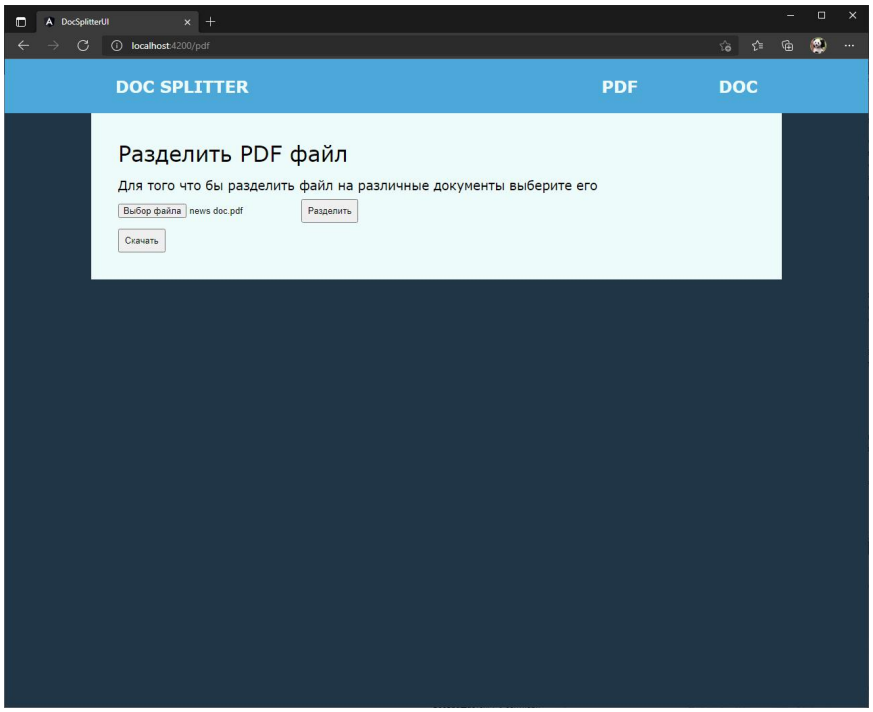

Рисунок 4.5 – Страница после обработки файла

Когда файл обработан, пользователь может нажать на кнопку скачать. В таком случае произойдёт запрос к сервису на получение архива, содержащего

файлы, полученные в результате разделения исходного файла пользователя. Имя архива генерируется автоматически. Он содержит папку с названием файла, загруженного пользователем, в которой находятся разделённые документы по порядку их следования.

Внешний вид архива и содержимое полученной папки изображены на рисунке 4.6.

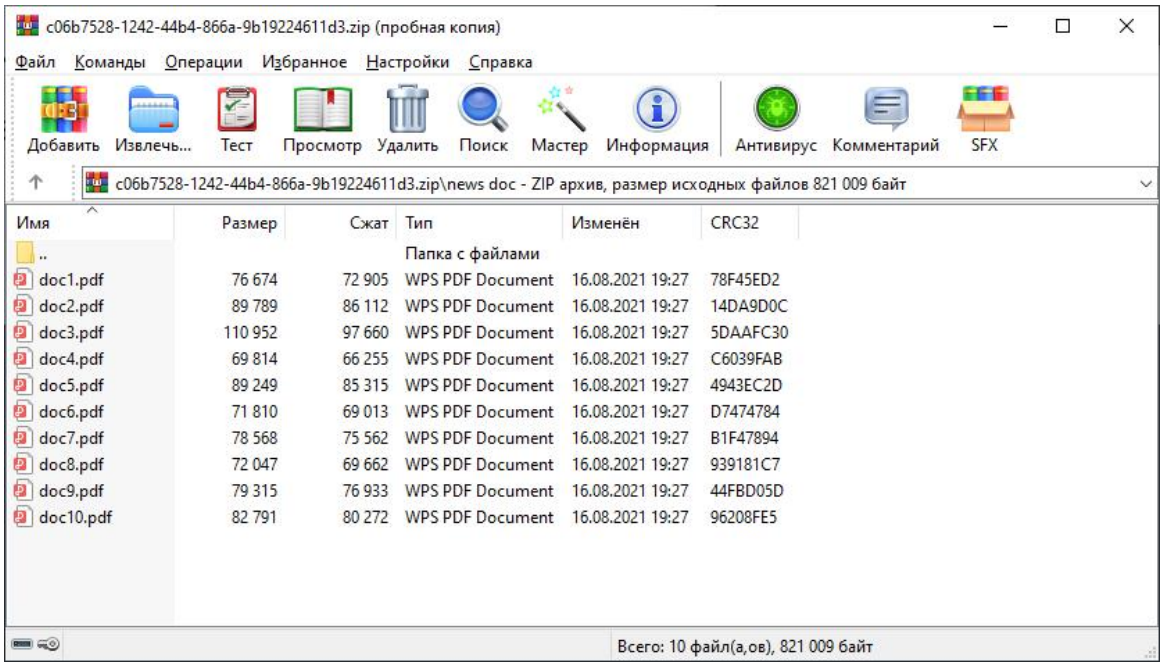

Рисунок 4.6 – Внешний вид содержимого полученного архива

После проверки процесса работы программы была проведена верификация разработанных сервисов.

4.2 Верификация результатов работы приложения

Для проверки программы был подготовлен *pdf* документ, содержащий 11 различных статей с сайта *BBC News*. Начало каждой новой статьи выделен жирным шрифтом. В идеале в результате работы приложение должно вернуть пользователю архив, содержащий 11 документов, в которых расположены различные новости.

В ходе проверки был получен архив, содержащий 10 файлов. Приложение не смогло корректно отделить третью новость от четвёртой. Был проведён анализ возможной ошибки модели и сделан вывод, что модель не смогла разделить эти две новости из-за того, что на последней странице третьей новости мало текста, из которого модель может сделать вывод к какой метке отнести две подряд идущих страницы.

В целом приложение с поставленной задачей справляется отлично. Дальнейшая доработка заключается в нахождении способа решения проблемы нехватки текста на страницах документов.

ЗАКЛЮЧЕНИЕ

Во время прохождения практики были изучены основные сведения о структуре предприятия. Были получены практические знания и навыки работы в организации.

Итогом практики стало приложение, разделяющее смешанные документы. Разработанное приложение призвано увеличить степень автоматизации тех областей, которые связаны с монотонной и рутинной работой с документами, которая отнимает много времени специалиста.

Вначале был проведён анализ предметной области. После этого была поставлена задача машинного обучения. Следующим шагом стал поиск подходящих данных, их очистка и первичный анализ.

После подготовки данных были проведены эксперименты с различными моделями, которые показали, что лучше всех для этой задачи подходит рекуррентная нейронная сеть.

После обучения нейронной сети был написан использующий её веб-сервис. Веб-сервис взаимодействует с пользователем посредством *REST* запросов.

Для удобства пользования инструментом был написан графический интерфейс пользователя, представляющий из себя веб-приложение, обращающееся к веб-сервису при помощи запросов. После отправки запросов приложение получает ответ и выводит его пользователю.

На последнем этапе создания программного комплекса было проведено тестирование, которое показало, что модель со своей задачей справляется отлично. Кроме того, был проведён анализ ошибок и обдуманы пути дальнейшего улучшения.

Благодаря практике, знания, полученные в университете, были обобщены и спроецированы на рабочий процесс, что поспособствовало лучшему их закреплению.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. EPAM Systems [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/EPAM_Systems. – Дата доступа: 02.07.2021.
2. История предприятия EPAM Systems [Электронный ресурс]. – Режим доступа: https://www.epam-group.ru/about/who-we-are/history. – Дата доступа: 02.07.2021.
3. Социальный пакет EPAM Systems [Электронный ресурс]. – Режим доступа: https://careers.epam.by/careers/benefits. – Дата доступа: 02.07.2021.
4. Плас, Джейк Вандер Python для сложных задач. Наука о данных и машинное обучение. Руководство / Плас Джейк Вандер. – М.: Питер, 2018. – 759 с.
5. Андреас, Мюллер Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными / Мюллер Андреас. – М.: Альфа-книга, 2017. – 487 с.
6. Мюллер А., Гвидо С. Введение в машинное обучение с помощью Python / Мюллер Андреас, Гвидо Сара – М.: Copyright, 2017. – 393 с.
7. Гринберг, М. Разработка веб-приложений с использованием Flask на языке Python / М. Гринберг. – М.: ДМК, 2016. – 272 с.
8. Олифер, В. Компьютерные сети. Принципы, технологии, протоколы. Учебник / В. Олифер, Н. Олифер. – М.: Питер, 2016. – 992 с.
9. М. Кантелон «Node.js в действии» - Питер, 2015. – 441 с.
10. Deep learning book [Электронный ресурс]. – Режим доступа: https://www.deeplearningbook.org. – Дата доступа: 03.07.2021.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг основных модулей веб-приложения

model_wrapper.py

```
from pathlib import Path
import keras
from keras.preprocessing.sequence import pad_sequences
import pickle
import re

import numpy as np

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords


MODEL_LOCATION = Path.cwd() / 'static/model_192_128.h5'
TOKENIZER_LOCATION = Path.cwd() / 'static/tokenizer.pickle'
MAX_SEQUENCE_LENGTH = 100
TRESSHOLD = 0.9


def normalize_text(text, lower=True, clean=True, remove_stopwords=True):
    if lower:
        text = text.lower()

    if clean:
        text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
        text = re.sub(r"what's", "what is ", text)
        text = re.sub(r"\'s", " ", text)
        text = re.sub(r"\'ve", " have ", text)
        text = re.sub(r"can't", "cannot ", text)
        text = re.sub(r"n't", " not ", text)
        text = re.sub(r"i'm", "i am ", text)
        text = re.sub(r"\'re", " are ", text)
        text = re.sub(r"\'d", " would ", text)
        text = re.sub(r"\'ll", " will ", text)
        text = re.sub(r",", " ", text)
        text = re.sub(r"\.", " ", text)
        text = re.sub(r"!", " ! ", text)
        text = re.sub(r"\/", " ", text)
        text = re.sub(r"\^", " ^ ", text)
        text = re.sub(r"\+", " + ", text)
        text = re.sub(r"\-", " - ", text)
        text = re.sub(r"\=", " = ", text)
        text = re.sub(r"'", " ", text)
        text = re.sub(r"\"", "", text)
        text = re.sub(r"(\d+)(k)", r"\g<1>000", text)
```

```
        text = re.sub(r":", " : ", text)
        text = re.sub(r" e g ", " eg ", text)
        text = re.sub(r" b g ", " bg ", text)
        text = re.sub(r" u s ", " american ", text)
        text = re.sub(r" 9 11 ", "911", text)
        text = re.sub(r"e - mail", "email", text)
        text = re.sub(r"j k", "jk", text)
        text = re.sub(r"\s{2,}", " ", text)

    text = re.split(r'\s+', text)

    if remove_stopwords:
        stops = stopwords.words("english")
        text = [word for word in text if word not in stops]

    return ' '.join(text)


def preprocess(parts1, parts2):
    processed_parts1 = []
    processed_parts2 = []
    for part1, part2 in zip(parts1, parts2):
        processed_parts1.append(normalize_text(part1))
        processed_parts2.append(normalize_text(part2))
    return processed_parts1, processed_parts2


def common_words_rate(first_part, second_part):
    common_rate = 0
    first_words = first_part.split()
    second_words = second_part.split()
    common = set(first_words) & set(second_words)
    for word in common:
        common_rate += first_words.count(word) / len(first_words) + second_words.count(word) /
len(second_words)
    return common_rate


def create_leaks(parts1, parts2):
    leaks = []
    for part1, part2 in zip(parts1, parts2):
        leaks.append(common_words_rate(part1, part2))
    return np.array(leaks)


def get_predicts(parts1, parts2, model, tokenizer):
    parts1, parts2 = preprocess(parts1, parts2)
    leaks = create_leaks(parts1, parts2)

    parts1 = tokenizer.texts_to_sequences(parts1)
    parts2 = tokenizer.texts_to_sequences(parts2)
```

```
    parts1 = pad_sequences(parts1, maxlen=MAX_SEQUENCE_LENGTH)
    parts2 = pad_sequences(parts2, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

    return np.array(model.predict([parts1, parts2, leaks]).reshape((-1, )))


class Model:
    def __init__(self):
        self.model = keras.models.load_model(MODEL_LOCATION)
        with open(TOKENIZER_LOCATION, 'rb') as loc:
            self.tokenizer = pickle.load(loc)

    def predict(self, texts1, texts2):
        texts1, texts2 = np.array(texts1), np.array(texts2)
        return get_predicts(texts1, texts2, self.model, self.tokenizer) < TRESSHOLD
```

splitter.py

```
import slate
from splitter.model_wrapper import Model
from pathlib import Path
from PyPDF2 import PdfFileWriter, PdfFileReader


class Splitter:
    def __init__(self):
        self.model = Model()

    def split(self, pdf_path: str):
        with open(pdf_path, 'rb') as fh:
            document = slate.PDF(fh, password='', just_text=1)
        splits = self.predict_splits(document)

        split_dir = Path(pdf_path.rstrip('.pdf'))
        split_dir.mkdir(parents=True, exist_ok=True)

        with open(pdf_path, "rb") as inp_file:
            input_pdf = PdfFileReader(inp_file)
            output = PdfFileWriter()
            output.addPage(input_pdf.getPage(0))

            doc_num = 1

            for i, split in zip(range(1, input_pdf.numPages + 1), splits):
                if split:
                    with open(split_dir / f'doc{doc_num}.pdf', "wb") as out_dir:
                        output.write(out_dir)
                        output = PdfFileWriter()
                        output.addPage(input_pdf.getPage(i))
                        doc_num += 1
                else:
                    output.addPage(input_pdf.getPage(i))
```

```
            with open(split_dir / f'doc{doc_num}.pdf', "wb") as out_dir:
                output.write(out_dir)

    def predict_splits(self, pdf_doc):
        pages = [page for page in pdf_doc]

        first_parts = []
        second_parts = []

        for i in range(1, len(pages)):
            first_parts.append(pages[i - 1])
            second_parts.append(pages[i])

        return self.model.predict(first_parts, second_parts)
```

app.py

```
import shutil
from flask import Flask, request, send_file
import zipfile
from flask_cors import CORS
from pathlib import Path
from splitter.splitter import Splitter
import os

# CONFIG
app = Flask(__name__)
CORS(app)

white_list = ['http://localhost:4200', 'http://localhost:5000']


spl = Splitter()


@app.after_request
def add_cors_headers(response):
    r = request.url_root
    if r in white_list:
        response.headers.add('Access-Control-Allow-Origin', r)
        response.headers.add('Access-Control-Allow-Credentials', 'true')
        response.headers.add('Access-Control-Allow-Headers', 'Content-Type')
        response.headers.add('Access-Control-Allow-Headers', 'Cache-Control')
        response.headers.add('Access-Control-Allow-Headers', 'X-Requested-With')
        response.headers.add('Access-Control-Allow-Headers', 'Authorization')
        response.headers.add('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT,
DELETE')
    return response


@app.route('/')
```

```
def index():
    pass


@app.route('/process', methods=["POST"])
def process():
    file = request.files['file']
    if file.filename.endswith(".pdf"):
        file_path = str(Path.cwd() / 'files' / file.filename)
        file.save(open(Path.cwd() / 'files' / file.filename, 'wb'))
        spl.split(file_path)

        path = Path.cwd() / 'files' / file.filename.rstrip('.pdf')
        with zipfile.ZipFile(str(path) + ".zip", 'w', zipfile.ZIP_DEFLATED) as ziph:
            for root, dirs, files in os.walk(path):
                for file in files:
                    ziph.write(os.path.join(root, file),
                               os.path.relpath(os.path.join(root, file),
                                               os.path.join(path, '..')))
        shutil.rmtree(path)
        return {"result": "ok"}


@app.route('/download', methods=["GET"])
def download():
    filename = request.args.get('filename')
    path = Path.cwd() / 'files' / (filename.rstrip('.pdf') + '.zip')
    if Path.exists(path):
        return send_file(path, as_attachment=True)
    else:
        return {'result': f"there is no file named {filename}"}


if __name__ == '__main__':
    app.run(debug=True)
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг основных компонентов веб-интерфеса

app.module

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import {HttpClientModule} from "@angular/common/http";

import { AppComponent } from './app.component';
import { NavigationComponent } from './navigation/navigation.component';
import {RouterModule} from "@angular/router";
import {AppRoutingModule} from "./app-routing.module";
import { PdfSplitterComponent } from './pdf-splitter/pdf-splitter.component';
import { HomeComponent } from './home/home.component';
import {FormsModule} from "@angular/forms";

@NgModule({
  declarations: [
    AppComponent,
    NavigationComponent,
    PdfSplitterComponent,
    HomeComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component

```
import { Component } from '@angular/core';

import {HttpClient} from "@angular/common/http";
import {OnInit} from "@angular/core";

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{
```

```
  title = 'Doc Splitter';

  constructor(private readonly http: HttpClient) {

  }

  ngOnInit() {

  }
}
```

file.service

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpHeaders} from "@angular/common/http";
import {Observable} from "rxjs";

@Injectable({
  providedIn: 'root'
})
export class FileService {
  private readonly baseUrl: string = 'http://localhost:5000';

  constructor(private http: HttpClient) { }

  process(file: File): Observable<any> {
    const formData = new FormData();
    formData.append('file', file, file.name);
    return this.http.post(`${this.baseUrl}/process`, formData);
  }

  download(file: File): Observable<any> {
    return this.http.get(`${this.baseUrl}/download?filename=${file.name}`, {responseType: 'blob' as
'json'});
  }
}
```

pdf-splitter.component

```
import { Component, OnInit } from '@angular/core';
import {FileService} from "../file.service";

@Component({
  selector: 'app-pdf-splitter',
  templateUrl: './pdf-splitter.component.html',
  styleUrls: ['./pdf-splitter.component.css']
})
export class PdfSplitterComponent implements OnInit {
  file?: File = undefined;
  processing: boolean = false;
  processed: boolean = false;
```

```
  constructor(private fileService: FileService) { }

  ngOnInit(): void {
  }

  onChange(event: Event) {
    this.processed = false;
    let input = event.target as HTMLInputElement;
    if(input.files) {
      this.file = input.files[0];
    }
  }

  onUpload() {
    this.processed = false;
    if(this.file) {
      this.processing = true;
      this.fileService.process(this.file)
        .subscribe((response: any) => {
          console.log(response);
          console.log(typeof (response));
          if(typeof (response) === 'object') {
            this.processing = false;
            this.processed = true;
          }
        })
    }
  }

  download() {
    if(this.file) {
      this.fileService.download(this.file)
        .subscribe(
          (response: any) => {
            let dataType = response.type;
            let binaryData = [];
            binaryData.push(response);
            let url = window.URL.createObjectURL(new Blob(binaryData, {type: dataType}));
            window.open(url)
          }
        )
    }
  }

}
```

ПРИЛОЖЕНИЕ В
(обязательное)
Охрана труда на рабочем месте инженера программиста

Охрана труда – система законодательных актов, социально-экономических, организационных, технических, гигиенических и лечебно-профилактических мероприятий и средств, обеспечивающих безопасность, сохранение здоровья и работоспособности человека в процессе труда. Научно-технический прогресс внёс серьёзные изменения в условия производственной деятельности работников умственного труда. Их труд стал более интенсивным, напряжённым, требующим значительных затрат умственной, эмоциональной и физической энергии.

Вся деятельность в области охраны труда регламентирована действующим законодательством Республики Беларусь, санитарными нормами и правилами, гигиеническими нормативами, предписаниями надзорных органов.

К комплексу мероприятий в области охраны труда относятся:

– проведение производственного лабораторного контроля за условиями труда на рабочих местах;

– разработка инструкций по охране труда и ознакомление с ними персонала;

– модернизация рабочих мест и технологического оборудования;

– создание безопасных условий труда.

Рабочее место – это часть пространства, в котором инженер осуществляет трудовую деятельность, и проводит большую часть рабочего времени. Рабочее место, хорошо приспособленное к трудовой деятельности инженера, правильно и целесообразно организованное, в отношении пространства, формы, размера обеспечивает ему удобное положение при работе и высокую производительность труда при наименьшем физическом и психическом напряжении.

При правильной организации рабочего места производительность труда инженера возрастает на значение от восьми до 20 процентов.

При организации рабочего места программиста на предприятии были соблюдены следующие основные условия:

– оптимальное размещение оборудования, входящего в состав рабочего места;

– достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения;

– хорошее естественное и искусственное освещение для выполнения поставленных задач;

– уровень акустического шума не превышал допустимого значения.

Главными элементами рабочего места программиста являются письменный стол и кресло. Основным рабочим положением является положение сидя.

Рабочая поза сидя вызывает минимальное утомление программиста. Рациональная планировка рабочего места предусматривает чёткий порядок и

постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, расположено в зоне лёгкой досягаемости рабочего пространства.

Моторное поле – пространство рабочего места, в котором могут осуществляться двигательные действия человека.

Максимальная зона досягаемости рук – это часть моторного поля рабочего места, ограниченного дугами, описываемыми максимально вытянутыми руками при движении их в плечевом суставе.

Важным моментом является также рациональное размещение на рабочем месте документации, канцелярских принадлежностей, что должно обеспечить работающему удобную рабочую позу, наиболее экономичные движения и минимальные траектории перемещения, работающего и предмета труда на данном рабочем месте.

При разработке оптимальных условий труда программиста необходимо учитывать освещённость, шум и микроклимат.

Помещения для работы программиста должны иметь естественное и искусственное освещение.

Площадь на одно рабочее место с видео-дисплейным терминалом (ВДТ) и ПЭВМ для взрослых пользователей должна составлять не менее 6,0 $м^2$, а объем не менее 20,0 $м^3$.

Искусственное освещение в помещениях эксплуатации ВДТ и ПЭВМ должно осуществляться системой общего равномерного освещения. В административно-общественных помещениях, в случаях преимущественной работы с документами, допускается применение системы комбинированного освещения.

В качестве источников света при искусственном освещении должны применяться преимущественно люминесцентные лампы.

Конструкция рабочего стола должна обеспечивать оптимальное размещение на рабочей поверхности используемого оборудования с учётом его количества и конструктивных особенностей (размер ВДТ и ПЭВМ, клавиатуры и др.), характера выполняемой работы. При этом допускается использование рабочих столов различных конструкций, отвечающих современным требованиям эргономики.

Конструкция рабочего стула (кресла) должна обеспечивать поддержание рациональной рабочей позы при работе на ВДТ и ПЭВМ, позволять изменять позу с целью снижения статического напряжения мышц шейно-плечевой области и спины для предупреждения развития утомления.

Тип рабочего стула (кресла) должен выбираться в зависимости от характера и продолжительности работы.

Рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья, при этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надёжную фиксацию.

Экран видеомонитора должен находиться от глаз пользователя на оптимальном расстоянии 600-700 мм, но не ближе 500 мм с учётом размеров алфавитно-цифровых знаков и символов.

Высота рабочей поверхности стола должна регулироваться в пределах 680-800 мм, при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм.

Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.

Клавиатуру следует располагать на поверхности стола на расстоянии не менее чем 300 мм от края, обращённого к пользователю или на специальной, регулируемой по высоте рабочей поверхности, отделённой от основной столешницы.

Помимо требований к организации рабочего места СанПиН 9-131 РБ 2000 устанавливает требования к микроклимату рабочей зоны: влажности, температуре, скорости потока воздуха и пр. Основной принцип нормирования микроклимата – создание оптимальных условий для теплообмена тела человека с окружающей средой.

К работе с ПК допускаются работники, не имеющие медицинских противопоказаний, прошедшие инструктаж по вопросам охраны труда, с группой по электробезопасности не ниже I.

Женщины со времени установления беременности и в период кормления грудью к выполнению всех видов работ, связанных с использованием ПК, не допускаются.

Работники должны соблюдать правила внутреннего трудового распорядка, установленные на предприятии, не допускать нарушений трудовой и производственной дисциплины.

Рабочий стол с учётом характера выполняемой работы должен иметь достаточный размер для рационального размещения монитора (дисплея), клавиатуры, другого используемого оборудования и документов, поверхность, обладающую низкой отражающей способностью.

Клавиатура располагается на поверхности стола таким образом, чтобы пространство перед клавиатурой было достаточным для опоры рук работника (на расстоянии не менее чем 300 мм от края, обращённого к работнику).

Чтобы обеспечивалось удобство зрительного наблюдения, быстрое и точное считывание информации, плоскость экрана монитора располагается ниже уровня глаз работника предпочтительно перпендикулярно к нормальной линии взгляда работника (нормальная линия взгляда – 15 град. вниз от горизонтали).