Санкт-Петербургский государственный университет

ВАРЕННИКОВА Анастасия Станиславовна

Выпускная квалификационная работа

Автоматическое выявление полезной обратной связи из комментариев к обучающим материалам платформы "JetBrains Academy"

Уровень образования: бакалавриат

Направление 45.03.02 «Лингвистика»

Основная образовательная программа СВ.5106. «Прикладная, компьютерная и математическая лингвистика (английский язык)»
Профиль «Прикладная, компьютерная и математическая лингвистика (английский язык)»

Научный руководитель:
к.ф.н., доцент, Кафедра математической лингвистики,
Митренина Ольга Владимировна
Рецензент:
преподаватель, Кафедра теоретической и прикладной лингвистики, Балтийский государственный технический университет им. Д. Ф. Устинова,
Мамаев Иван Дмитриевич

Санкт-Петербург
2022

Аннотация

В работе исследуются методы классификации текстов, совмещающих в себе естественный язык и части кода на языке программирования Python. В практической части рассматривается реализация программы для автоматической сортировки комментариев студентов обучающей платформы "JetBrains Academy". Алгоритм возвращает только полезную обратную связь к обучающим материалам на сайте компании, на основе которых можно внести улучшения в теорию и практические задания. В программе использует один из четырех методов машинного обучения с учителем, описанных в теоретической главе, в сочетании с дополнительной проверкой на наличие в текстах определенных n-грамм, указывающих на релевантность комментария. Также в работе подробно описывается очистка исходного размеченного датасета, предобработка комментариев в нем и векторизация текстовых данных.

Ключевые слова: Python, естественный язык, датасет, машинное обучение с учителем, классификация, векторизация, n-граммы, регулярные выражения

The paper investigates the text classification methods that combine natural language and code blocks written in Python programming language. The study focuses on a Python program which automatically sorts feedback comments from students of the online educational platform "JetBrains Academy". The algorithm returns only the insightful feedback on the educational content published on the company website, which can later be used to improve the theoretical section as well as practical assignments. The program utilizes one the the four supervised machine learning methods described in the theoretical chapter, along with an additional check for the presence of certain n-grams in the texts, indicating the relevance of the comment. The paper also contains the detailed description of the

tagged dataset cleaning, the preprocessing of comments in it, and the vectorization of textual data.

Keywords: Python, natural language, dataset, supervised learning, classification, vectorization, n-gramms, regular expressions

Оглавление

Введение

В современном деловом мире одной из важных задач является получение обратной связи от потребителя. Если таких комментариев получается слишком много, возникает задача их автоматической сортировки. В данной работе по заказу компании "JetBrains Academy" предпринята попытка автоматического выявления полезных комментариев к обучающим материалам по программированию.

Просматривать сообщения, оставленные пользователями необходимо, так как на их основе можно сделать вывод о качестве теоретического материала или понятности практических заданий. Проделывать эту операцию вручную затратно по ресурсам, поэтому и появляется необходимость автоматическим образом осуществить сортировку комментариев так, чтобы с высокой точностью отбрасывать тексты, не несущие полезной информации для создателей обучающего контента, и сохранять большую часть актуальной обратной связи, которая поможет усовершенствовать качество подаваемого материала.

Целью данной выпускной квалификационной работы является разработка алгоритма для автоматического выявления полезной обратной связи из комментариев к обучающим материалам платформы "JetBrains Academy".

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Изучить современные алгоритмы машинного обучения, выполняющие классификацию данных.
2. Выполнить предобработку корпуса комментариев, предоставленного компанией.
3. Преобразовать комментарии в векторный формат.

4. Провести сравнительный анализ качества работы разных алгоритмов машинного обучения и способов трансформации текстовых данных в вектора.

5. На основе проведенного эксперимента выявить лучший алгоритм и оформить его в API, которым в дальнейшем смогут пользоваться сотрудники "JetBrains Academy".

Данная исследовательская работа имеет высокую **практическую значимость**, поскольку она поможет улучшить качество работы платформы, на которой обучается большое количество людей со всего мира.

Актуальность работы определяется тем, что на сегодняшний день не существует универсального подхода к решению такой проблемы как классификации текстов: данные из разных источников всегда различны и необходимо проводить глубокий анализ для достижения хороших результатов. Таким образом, проведенный эксперимент поможет выявить наиболее подходящий вариант работы с такими типами текстовых данных, какие могут встретиться в сфере обучения программированию.

Глава 1. Обработка естественного языка

1.1 Язык как данные

Язык — это **неструктурированные** данные, которые используются людьми для общения между собой. Но компьютеру такой язык непонятен, он, в свою очередь, воспринимает только **структурированные** или **полуструктурированные** данные, включающие поля или разметку, которые позволяют компьютерной программе анализировать их. Неструктурированные данные, несмотря на отсутствие машиночитаемой структуры, не являются случайными. Напротив, они подчиняются лингвистическим правилам, которые делают эти данные понятными для людей. [1, стр. 31]

Естественные языки определяются контекстом использования, а не правилами, его приходится реконструировать для компьютерной обработки. Чаще всего мы сами определяем значения используемых слов, хотя и совместно с другими участниками беседы. Например, словом *crab* в английском языке может обозначаться морское животное или вечно недовольный, раздражительный человек или движение боком, но при этом оба – автор устного или письменного текста и его слушатель или читатель – должны согласиться с общим пониманием данного слова в ходе диалога. Именно поэтому язык обычно ограничивается обществом и территориальным местоположение – общаться и передавать смысл высказывания обычно намного проще людям, имеющим схожий жизненный опыт.

В отличие от формальных языков, в которых одинаковые сочетания знаков всегда имеют одинаковый смысл и которые всегда являются предметными, естественные языки намного более универсальны. Но из этого вытекает проблема избыточности естественных языков, необходимой для поддержания множества смыслов. Информационная избыточность является тем свойством любого текста, которое обеспечивает возможность успешного

восприятия речи. Также стоит отметить, что с этой точки зрения любой текст на естественном языке характеризуется информационной избыточностью, в противном случае он не может быть воспринят и понят адресатом, как, например, не будет ясен человеку текст программы, написанный на одном из языков программирования. [2, стр. 21]

Однако избыточность представляет собой серьезное препятствие, потому что мы не можем указать буквальный смысл для каждого символа, так как он по умолчанию является неоднозначным. Структурная и лексическая неоднозначность является основной характеристикой человеческого языка; данная особенность не только дает нам возможность генерировать новые идеи, но также позволяет общаться людям с разным опытом и культурой, хотя нередко возникают случайные недоразумения при таком взаимодействии.

В то же время качества, которые делают естественный язык таким богатым инструментом общения между людьми, затрудняют его анализ с применением детерминированных правил. Мы превосходим компьютер в моментальном понимании языка, поскольку люди обладают гибкостью интерпретации. Следовательно, в программной среде нужны столь же нечеткие и приспосабливающиеся к контексту вычислительные методы. И в этом методы машинного обучения могут поспособствовать – их добавление в сферу обработки естественного языка обеспечило определенную гибкость.

На самом деле приложениям, использующие приемы обработки естественного языка для анализа текстовых и аудиоданных, уже давно стали неотъемлемой частью нашей жизни, благодаря инновациям в области машинного обучения и эпохе переизбытка данных, когда сам Интернет представляет собой колоссальный граф знаний, который, среди всего прочего, содержит обширную гипертекстовую энциклопедию, специализированные базы данных о фильмах, музыке, спортивных результатах и о бесконечном количестве других вещей. [3, стр. 25] Эти продукты настолько распространены, что мы совершенно не замечаем

широкий спектр закулисных инструментов: от спам-фильтров, собирающих информацию о нашем почтовом трафике, до поисковых систем, которые выдают именно то, что мы искали, и виртуальных помощников, всегда готовых ответить на наш любой вопрос.

В качестве простого примера внедрения анализа естественного языка можно привести поддержку «тегов рекомендаций», реализованную в информационных продуктах таких компаний, как Amazon, Netflix, YouTube и других. В тегах хранится метаинформация о фрагментах контента, они определяют свойства описываемого ими содержания и могут использоваться для группировки схожих элементов. Данная метаинформация важна для рекомендаций и поиска, они играют большую роль в определении того, заинтересует ли контент конкретного пользователя.

И существует еще множество других интересных примеров, наверное, наиболее привычным для нас является функция в iMessage, технология обмена мгновенными сообщениями от компании Apple, которая пытается предсказать, что пользователь напечатает дальше, опираясь на недавно введенный текст, а функция автокоррекции исправляет орфографические ошибки. Также имеется ряд ответно-вопросные систем и голосовых виртуальных помощников: Alexa от Amazon, Алиса от Яндекса, – способных анализировать речь и давать более или менее осмысленные ответы.

Хотя это направление активно развивается в последнее десятилетие, все же многие проблемы еще остались нерешенными и далеко не все возможные перспективы реализованы. Основная сложность в науке о данных (*data science*), которую отмечают специалисты в этой области, связана с тем, что процесс исследования данных не всегда совместим с практикой разработки программного обеспечения. Данные могут быть непредсказуемыми, поэтому всегда сложно предугадать будет ли результат положительным. Хиллари Мейсон (*Hilary Mason*) однажды сказала о разработке приложений для обработки данных, «наука о данных не всегда отличается гибкостью». [4]

Более того, большинство публикаций о машинном обучении и обработке естественного языка носят исследовательский характер, что осложняет процесс разработки реальных приложений. Например, несмотря на наличие большого количества отличных инструментов для машинного обучения на текстовых данных, имеющиеся в открытом доступе источники информации – документации, руководства и статьи в Сети – как правило, опираются на искусственно подобранные массивы данных и исследовательские инструменты. Достаточно редко можно найти объяснение на конкретных примерах, в частности, того, как создать корпус, достаточно большой для поддержки программы, как управлять его структурой и размером или как трансформировать исходные документы в данные, пригодные для использования. Но именно эти аспекты являются ключевой частью практики создания масштабируемых приложений по обработке данных, основанных на анализе естественного языка.

Целью прикладного анализа текста является не что иное, как создание таких приложений – они принимают на входе текстовые данные, разбирают их на составные части, выполняют вычисления на основе этих частей и вновь собирают их, возвращая осмысленный результат. Таким образом, приложения по обработке данных извлекают ценные сведения из исходных текстовых материалов и, в свою очередь, генерируют новые данные. [5]

На данный момент базовая методология, которые используется при создании приложений, основанных на анализе естественного языка, – это машинное обучение с учителем или без. С его помощью можно разбить на значимые кластеры тексты по схожести или классифицировать тексты с применением конкретных меток.

Стандартный конвейер (*pipeline*) такого приложения реализует итеративный процесс, состоящий из двух этапов: **сборки** и **развертывания** – и является отражением конвейера машинного обучения. [6] На первом этапе сборки исходные данные преобразуются в форму, подходящую для передачи в модель машинного обучения и экспериментов с ними. На заключительном

этапе развертывания происходит окончательный выбор моделей, которые в дальнейшем будут использоваться для оценок и прогнозов, непосредственно касающиеся пользователя. На рис. 1.1 показан путь данных в приложении обработки естественного языка:

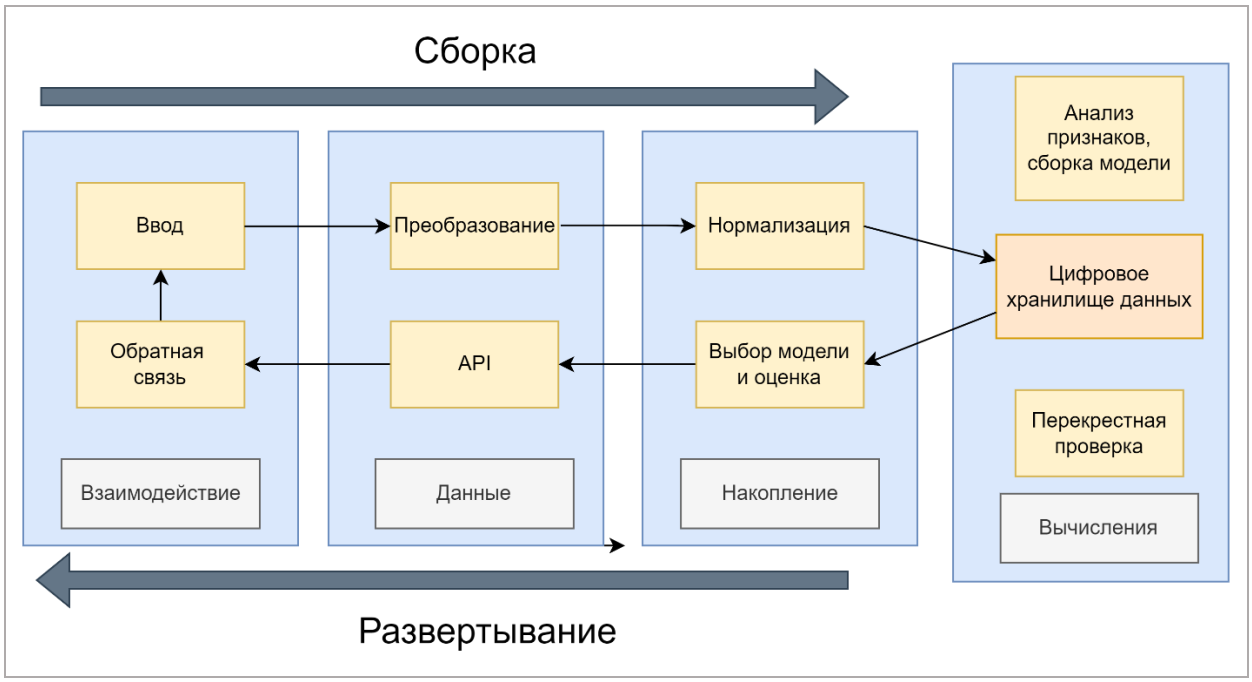

Рис. 1.1 – Конвейер приложения обработки естественного языка

В представленной выше диаграмме можно выделить для каждого из двух этапов четыре стадии: взаимодействие, данные, накопление и вычисления. Рассмотрим их подробнее:

1) В процессе **взаимодействия** необходима вспомогательная программа для ввода данных, а пользователю – прикладной интерфейс.

2) Под этапом **данных** понимаются внутренние компоненты, которые действуют как промежуточное звено перед переходом к стадии накопления. Данные преобразовываются, чтобы они были доступны для использования либо с помощью модели, либо с помощью какого-либо другого формата.

3) **Накопление** – функция хранения, которую обычно выполняет база данных. На этой стадии происходит сохранение больших объемов данных и параметров, необходимых для работы.

4) Наконец, стадия **вычислений** или обработки ответственна за обучение моделей и управление хранилищем вычислительных данных.

Стоит отметить, что этап сборки данных приложений требует особого внимания – тем более в случае анализа текста. Реализуя приложения, обрабатывающие естественный язык, мы создаем дополнительные лексические ресурсы (такие как словари, переводчики, регулярные выражения и т. д.), от которых зависит работа продукта.

На рис. 1.2 показано более развернутое представление стадии сборки для приложений машинного обучения, основанных на анализе естественного языка. Процесс перехода от исходных данных к развернутой модели, соответственно, состоит из последовательных преобразований данных. Сначала полученные материалы трансформируются во входной корпус, накапливаются и сохраняются в хранилище данных. Далее входные данные группируются, проходят очистку и нормализацию, затем преобразуются в векторы для последующей обработки. В конечном преобразовании одна или несколько моделей обучаются на векторизованном корпусе. В результате всех вышеописанных операций создается обобщенное представление оригинальных данных, которое потом используется готовым приложением.

Развертывание, помимо выбора и использования модели, не сильно отличается от более прямолинейной разработки программного обеспечения. Часто при создании таких приложения разрабатывается API (*Application Programming Interface – программный интерфейс приложения*), который используется другими программами или пользовательскими интерфейсами.

В конце пользователи дают обратную связь на выходные данные моделей, а их ответ, в свою очередь, снова подается на вход и используется для адаптации моделей.

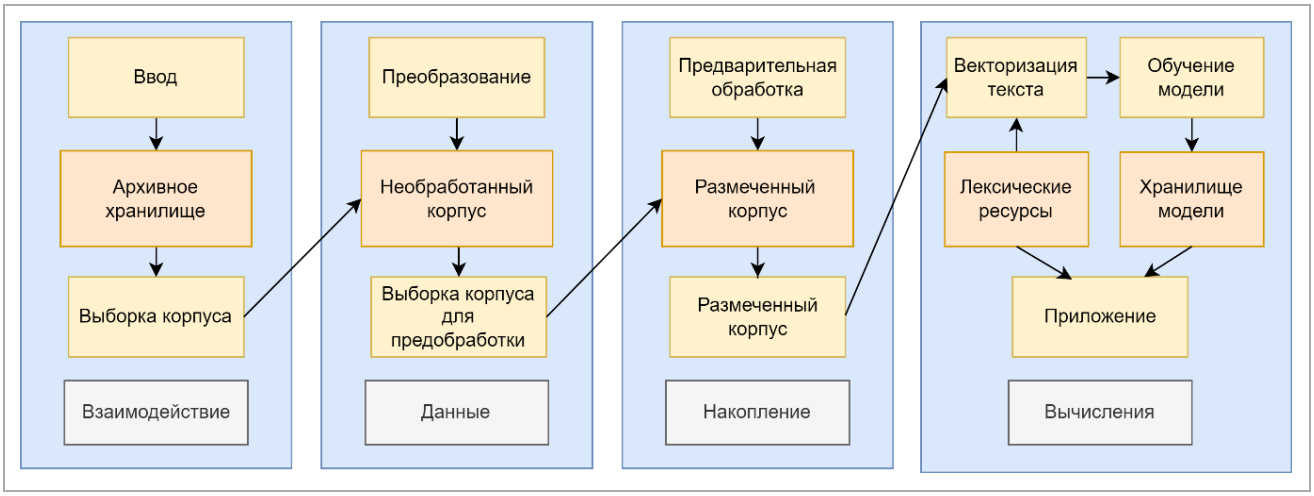

Рис. 1.2 – Этап сборки в приложениях обработки естественного языка

В заключении можно сказать, что неструктурированные данные трудно подогнать под конкретную модель данных, потому что их содержимое зависит от контекста или имеет переменный характер. Тексты на естественном языке составляют особую разновидность такого типа данных. Обработка такой информации достаточно сложна, потому что она требует знания как лингвистики, так и специальных методов data science. Есть существенный прогресс в направлении обработки данных на естественном языке, сообщество добилось успеха в сфере распознавания сущностей и тематических областей, обобщения, генерации текста и анализе тональности, но модели, адаптированные для одной предметной области, плохо обобщаются для других. Даже самые современные методы не смогут расшифровать смысл произвольного фрагмента текста. И этот факт не удивителен, поскольку у людей также возникают проблемы с восприятием естественного языка. Он неоднозначен по своей природе. Если два человека слушают один и тот же разговор, то вынесут ли они одинаковый смысл из него? Кроме того, смысл отдельных слов может меняться в зависимости от настроения говорящего. Следовательно, сама концепция смысла выглядит спорно. [7, стр. 22]

1.2 Машинное обучение как метод обработки данных

С давних времен человек выполнял различного рода работу в большом количестве, но не всегда есть возможность, время и ресурсы, чтобы выполнять определенные задачи вручную. Поэтому нельзя сказать, что машинное обучение (МО), или *machine learning*, является совершенно новой технологией, совсем наоборот, закладка фундамента в этой области началась в XVII веке с появлением первого механического калькулятора (1642) и разработкой Готфридом Лейбницем двоичной системы (1679).

Уже в XX столетие постепенно начала распространяться теория **когнитивного обучения**, разработанная швейцарским психологом Жаном Пиаже. Данная теория строится на основе предположения, что человек обладает способностью к обучению и может структурировать и сохранять накопленную информацию. Как мы видим по скорости развития современных технологий, когнитивное обучение было адаптирована и для компьютеров.

Одним из пионеров в этой области стал Артур Самуэль, создавший программу Checkers-playing (1956) – инновационная самообучающаяся программ в мире, она умела, как следует из названия, играть в шашки. Хоть правила игры относительно просты, но имеют определенную стратегию, поэтому было выгодно отработать на ней алгоритм, соответствующий проблемам искусственного интеллекта (ИИ). Компьютер обучался на гидах по игре, в которых были прописаны партии с хорошими и плохими ходами. В 1959 году Самуэль дал следующее определение машинному обучению: «Это процесс, в результате которого машина (компьютер) способна показывать поведение, которое в нее не было явно заложено (запрограммировано)». [8]

Также стоит отметить работу Фрэнка Розенблатта, в конце 1950-х в Корнелльском университете он построит систему Mark I Perceptron, которую условно можно признать первым нейрокомпьютером. Эта машина основана

на принципе персептрона, компьютерной модели восприятия информации мозгом. Перцептрон стал одной из первых моделей нейросетей.

Отталкиваясь от всего вышесказанного, можно сформулировать определение: **машинное обучение** – класс методов искусственного интеллекта (*artificial intelligence*), ориентированный на решение прикладных задач. Характерной чертой МО является обучение в процессе решения множества сходных задач, что подобно человеческому поведению в процессе овладения каким-либо ремеслом путем проб и ошибок. Для построения таких методов используются средства математической статистики, численных методов, теории вероятностей, теории графов и другие различные техники работы с данными в цифровой форме.

Цель МО заключается в предсказывании результатов по входным данным на основании некой математической модели. Чтобы реализовать концепцию МО, эксперты разрабатывают алгоритмы общего назначения, которые могут применяться к большим классам задач обучения. Если требуется решить определенную проблему, достаточно передать алгоритму более конкретные данные, здесь речь идет об обучении на примерах. В большинстве случаев компьютер использует данные как источник информации, сравнивает свой результат с желаемым, а затем вносит исправления. Чем больше данных или «опыта» накапливает компьютер, тем лучше он справляется со своей работой – как и человек. [7, стр. 82] Но нужно учитывать то, что данные должны быть предоставлены не только в большом количестве, но должны быть также разнообразны, иначе машине будет трудно найти закономерности, а это может привести к неточности результата.

У МО есть три основные составляющие: **данные** (*data*), **признаки** (*features*) и **алгоритм обучения** (*learning algorithm*). Как было сказано выше, материал, на котором будет проходить обучение, должен состоять хотя бы из нескольких тысяч. Их можно собирать вручную или с помощью парсера, который осуществляет автоматический сбор информации с заданного

ресурса. Иногда можно проводить исследования уже на готовых базах данных или, как их часто называют, **датасетах** (*data set*), но они являются большой редкостью.

Во-вторых, сложной задачей является выбор признаков, характеристик, на которые будет опираться машина в процессе обучения. Они могут совершенно разными в зависимости от цели и области исследования. Очень важно выделить нужные закономерности, чтобы они были присуще не только обучающей выборке, но и последующим прецедентам. Кроме того, в некоторых методах МО не используются заранее известные классы признаков.

Когда признаки попадают на входы системы МО, эта система пытается заметить закономерности между признаками, а на выходе генерируется результат этого процессам. Результаты работы принято называть **меткой** (*label*), так как система помечает входы определенным прогнозом, в какую категорию попадет выход после классификации. А саму систему, по которой элементу приписывается ответа, принято называть **целевой функцией**. [9, стр. 123]

Стоит также помнить, что для решения одной задачи чаще всего найдется не один метод. От выбранного алгоритма решения будет зависеть точность, скорость работы и размер готовой модели. Но все равно нельзя забывать, что качество все-таки в большей степени зависит от предоставленных программе данных. Существует большое количество различных алгоритмов, которые применяются как по отдельности, так и в совокупности.

Таким образом, МО можно определить последовательностью действий:

- **представление** классифицируемых элементов данных на формальном языке, который машина может интерпретировать;
- **оценка** – функция, которая позволяет определить полезность тех или иных классификаторов;
- **оптимизация** – поиск наилучших классификаторов.

По окончанию работы система должны выдать в идеале правильные ответы, а человек при этом затратит меньше рабочей силы на обработку большого количества данных. В таком случае можно будет считать, что машиной была подобрана корректная целевая функция.

Но выбор оптимальной модели – это сложный и итеративный процесс, состоящий из повторения циклов обучения, конструирования признаков, выбора модели и настройки гиперпараметров (так называют параметры алгоритмов МО, значения которых устанавливаются перед запуском процесса обучения). После каждой итерации выполняется оценка результатов с целью получить лучшую комбинацию модели, параметров и признаков. Этот процесс, изображенный на рис. 1.3, можно называть **тройкой выбора модели** (*Model Selection Triple* – MST). Выбор модели является итеративной и исследовательской операцией, поскольку пространство MST обычно бесконечно, и аналитики, как правило, не могут априори знать, какой MST даст удовлетворительную точность. Следовательно, цель данного процесса заключается в том, чтобы представить итерацию как основу науки машинного обучения, которая должна облегчаться, а не ограничиваться. [10]

Также Викхем (*Wickham*) с соавторами в своей статье, вышедшей в 2015 году, отлично устранили неоднозначность перегруженного термина «модель», описав три базовых случая его использования в машинном обучении: семейство модели, форма модели и обученная модель. Широкое понятие **семейства моделей** описывает отношения между переменными и поставленной целью (например, линейная модель предсказывает один ответ по формуле линейного уравнения и предполагает, что случайные ошибки имеют нормальное распределение). **Форма модели** представляет собой конкретный экземпляр MST: набор признаков, выбранный алгоритм и конкретные гиперпараметры. Наконец, **обученной моделью** считается та модели, которая обучилась на определенном наборе данных, и способная делать предсказания на новых, ранее не виденных данных. [11]

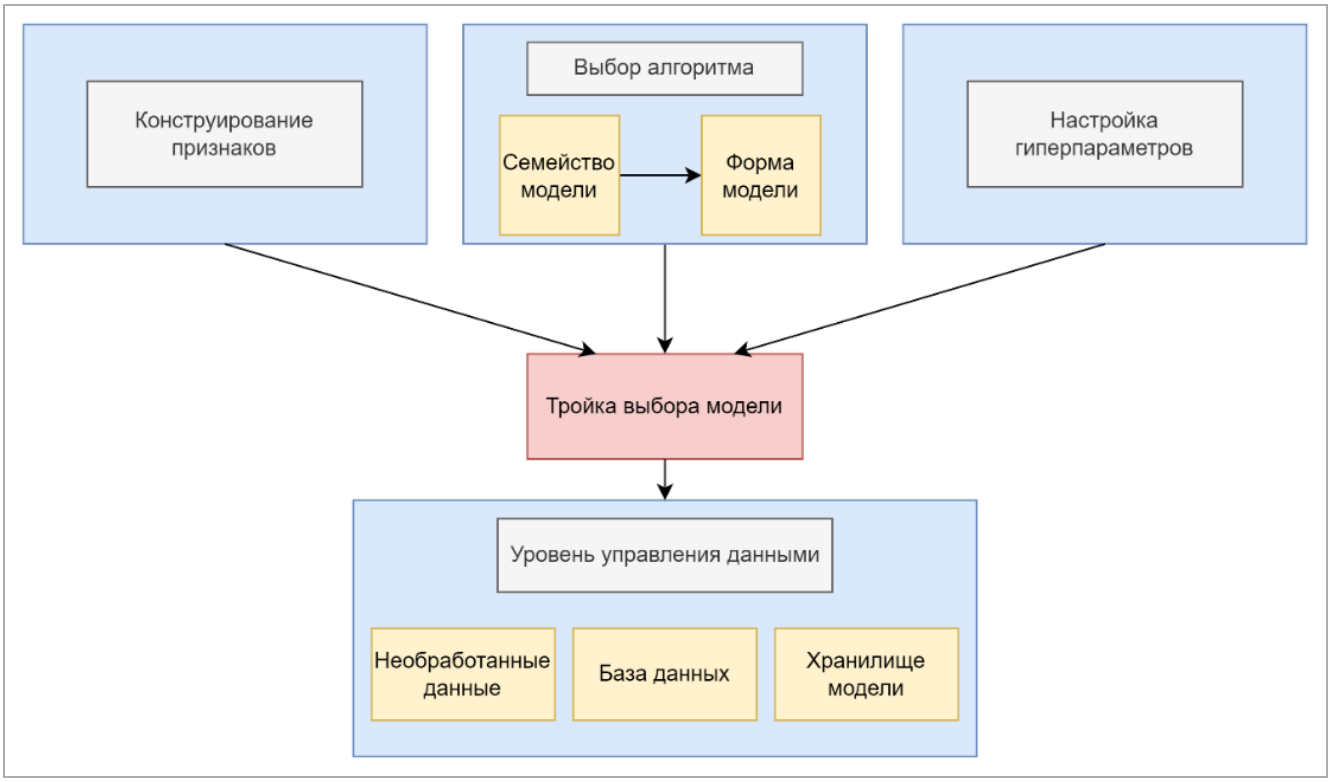

Рис. 1.3 – Тройка выбора модели

Если вернуться к вопросу о приложениях, обрабатывающих естественный язык, то основной задачей анализа текста является интерпретация происходящего в каждом из этих преобразований. При этом с каждым последующим преобразованием текст становится все менее значимым для нас, потому что все меньше напоминает человеческий язык.

1.3 Классификация в текстовом анализе

Мы живем в эпоху изобилия данных, структурированных и неструктурированных, и из них можно извлекать большую пользу для человека. Важность МО возрастает не только в исследованиях, имеющих отношение к компьютерным наукам; его роль в нашей повседневной жизни становится даже еще больше. МО позволяет нам пользоваться надежными фильтрами почтового спама, испытанными поисковыми механизмами, удобными приложениями по распознаванию текста и речи и, возможно, что в ближайшем будущем – безопасными беспилотными автомобилями.

Существует три основных типа МО: **обучение с учителем** (*supervised learning*), **обучение без учителя** (*unsupervised learning*), **обучение с подкреплением** (*reinforcement learning*). В данной выпускной квалификационной работе будет подробно описано обучение с учителем.

Обучение с учителем и без объединяет то, что это два направления **классического обучения**, первые алгоритмы которого имею статистическую природу. Их задача заключалась в поиске закономерности в цифрах, оценке близости точек в пространстве и вычисления направления векторов. Хоть данные алгоритмы зарождались в 1950-х годах, их актуальность не устаревает. Сегодня на классических алгоритмах держится множество привычных нам вещей, например, когда на сайте нам встречается блок «Рекомендованные статьи» или когда банковская карта может быть заблокирована из-за подозрительных операций. Кроме того, классическое обучение является более выгодным, так как не требует чрезвычайно больших наборов данных, для некоторых задач большие датасеты недоступны либо их приобретение окажется дорогостоящим и трудоемким.

Теперь основательно рассмотрим один из типов классического МО. Главная цель обучения с учителем – обучить модель на размеченной **обучающей выборке**, чтобы в дальнейшем модель смогла присваивать категории новым экземплярам, опираясь на шаблоны, выявленные в процессе обучения. Здесь понятие "с учителем" относится к набору образцов, где желаемые входные метки уже известны.

В качестве примера рассмотрим фильтрацию почтового спама, можно обучить модель с использованием алгоритма машинного обучения с учителем на выборке помеченных почтовых сообщений, они должны быть маркированы корректно – спам или не спам. Задача обучения с учителем с метками дискретных классов, такими, как в нашем примере фильтрации почтового спама, также называется **задачей классификации**.

Классификация является основной формой анализа текста, ее можно встретить в разделении веб-страниц и сайтов по тематическим каталогам, в анализе тональности, в определении языка текста и в показе наиболее релевантной рекламы.

Классификация – это подкатегория обучения с учителем, где целью является прогнозирование категориальных меток классов, к которым принадлежат новые образцы, на основе прошлых наблюдений. Такие метки классов представляют собой дискретные неупорядоченные значения, которые могут пониматься как принадлежность к группам образца. [12, стр. 30] Ранее упомянутый пример выявления почтового спама демонстрировал типичную задачу **двоичной классификации**, когда алгоритм МО изучал набор правил, чтобы проводить различие между двумя возможными классами: спам или не спам. Однако набор меток классов вовсе необязательно имеет двоичную природу, классификация такого типа будет называться **многоклассовой**, характерным примером такой задачи считается распознавание рукописных символов. В таком случае обучающий набор данных будет содержать множество рукописных примеров для каждой буквы алфавита. Теперь если пользователь с помощью устройства ввода предоставит новый рукописный символ, тогда прогнозирующая модель будет в состоянии с определенной точностью предсказать корректную букву алфавита для введенного рукописного символа. Но такая система МО не будет способна правильно распознать, например, цифры, если их изначально не было в обучающей выборке.

Самая сложная часть прикладного анализа текста – курирование и сбор предметно-ориентированного корпуса для построения моделей. Вторая по сложности часть – выработка аналитического решения конкретной прикладной задачи. [1, стр. 113]

В настоящее время разработан большое ряд моделей и механизмов классификации, которые математически более разнообразны, чем линейные модели, используемые в основном для задач регрессии, которые направлены

на прогнозирование непрерывных результатов, то есть вместо категорий предсказывается число. Приложения по анализу текстовых данных имеют выбор из широкого круга семейств моделей – от методов на основе экземпляров, использующих измерение расстояния между точками в пространстве и байесовские вероятности, до линейных и нелинейных аппроксимаций и нейронных сетей. Однако в основе всех семейств моделей классификации лежит один и тот же базовый процесс: можно одновременно тестировать несколько моделей для решения конкретной задачи и затем сравнивать их путем приемов перекрестной проверки для выбора наиболее эффективных.

Сам процесс классификации делится на два следующих этапа, представленных на рис. 1.4, – обучение и эксплуатация. На первом этапе корпус текстовых документов преобразуется в вектора признаков. Далее признаки документов вместе с их метками, распознаванию которых мы хотим обучить модель, передаются в алгоритм классификации, который определяет свое внутреннее состояние и выявленные шаблоны. После обучения можно векторизовать новый, ранее неизвестный документ в то же пространство признаков и передать результат алгоритму, который вернет предсказанную метку категории документа.

При выполнении данной дипломной работы стояла задача двоичной классификации, которая распознает только два класса, и оба считаются противоположными друг другу как, например, да и нет или, в нашем случае, полезная и бесполезная обратная связь. Если посмотреть на вопрос с точки зрения вероятности, то двоичный классификатор с классами A и B предполагает, что $P(B) = 1 - P(A)$.

Однако в реальности такая однозначная связь, разделение четко на два класса, встречается довольно редко, в частности, при анализе эмоциональной окраски, если документ не положительный, значит ли это, что он отрицательный? Какие-то документы могут носить и нейтральную окраску, как это часто бывает в действительности, добавление третьего класса в

классификатор может в значительной мере улучшить распознавание положительных и отрицательных текстов. В результате эта задача становится задачей многоклассовой классификации.

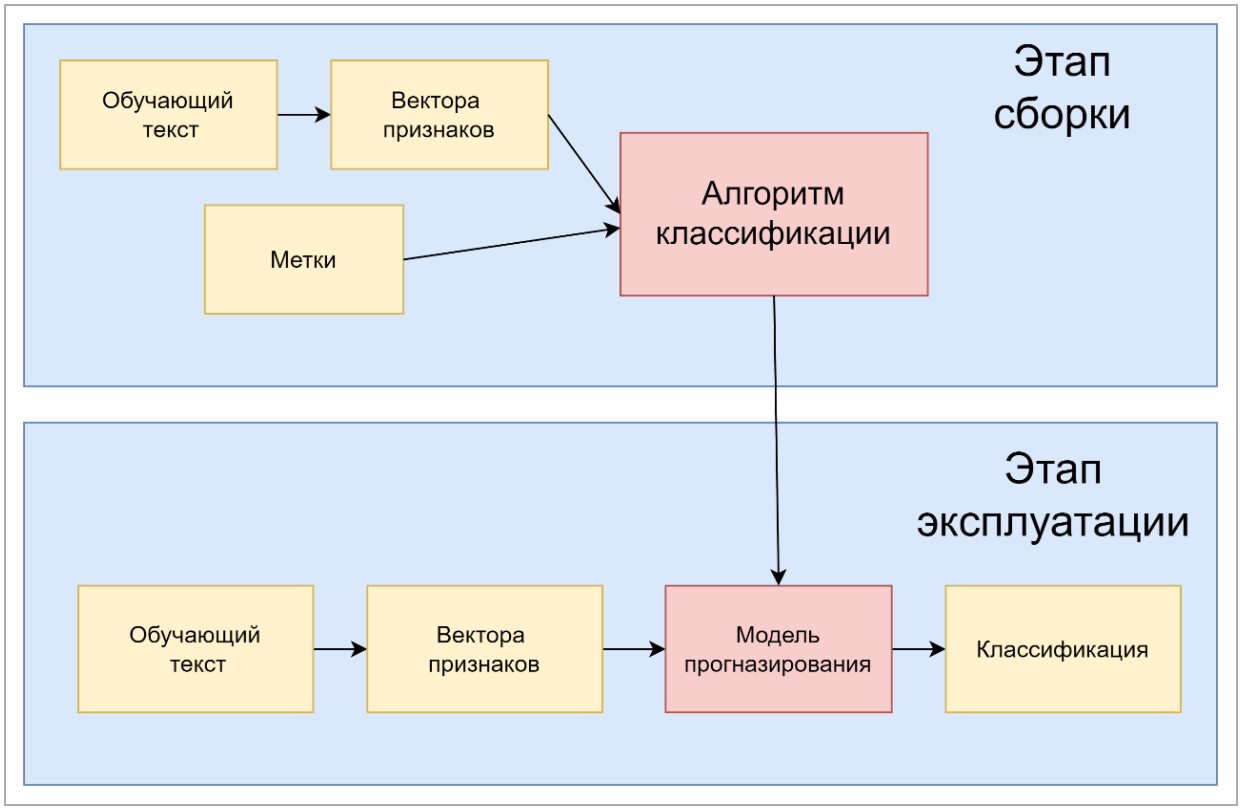

Рис. 1.4 – Процесс классификации

Как говорилось в предыдущем разделе, часто при обработке текстовых данных создаются дополнительные лексические ресурсы. Для улучшения бинарной классификации, выполняемой алгоритмом машинного обучения, можно использовать методы из ранее упомянутого анализа тональности, популярная дисциплина из широкой области обработки естественного языка. Данный подход не подразумевает извлечение фактов из информации, он занимается только степенью эмоциональной окраски, анализ тональности отслеживает именно эмоции авторов в отношении определенной темы.

Как и в других задачах прикладной лингвистики, основные подходы к автоматическому определению тональности текста можно разделить на две большие группы. Алгоритмы первой группы основаны на правилах (*rule-based*), а алгоритмы второй группы используют методы МО. [13]

Существует несколько способов извлечь мнение из большой коллекции текстов без использования МО, ниже перечислим основные из них и кратко обсудим принцип их работы.

1) **Построение правил**

Алгоритмы, создаваемые на основе правил, позволяют учитывать семантические, структурные особенности различных слов и самого языка. Заранее создаются шаблоны, по этим шаблонам из текста извлекаются n-компонентные цепочки (n-граммы), их тональность определяется как на основе правил, так и на основе словарей; например, правила могут строиться по модели «если …, то…». Но реализация такого метода сталкивается с рядом проблем:

- узкая сфера применения набора правил в связи с тем, что формат написания различных сообщений в сети Интернет достаточно сильно отличается от принятых норм русского языка в литературной форме;
- требуется формировать определённый корпус различных лингвистических правил, который обязан учитывать обширную часть различных конструктивных языковых особенностей;
- алгоритм не сможет быть перенесен на другой язык из-за связи с уникальной структурой определенного языка. [14]

2) **Словари оценочной лексики**

В словарях оценочной лексики хранятся слова и словосочетания, каждому из которых присвоен уровень эмоциональной оценки. В зависимости от словаря используются разные шкалы оценок: словам может быть приписана только одна, либо две тональных оценки. Также существуют наборы слов, которым приписываются разные эмоции (такие как «страх», «удивление», «вера», «гнев», «радость» и т.д.) и тональные оценки (положительная или отрицательная).

Далее в практической части выпускной квалификационной работы будет продемонстрировано, как комбинировать подход, основанный на правилах, а именно цепочки из n-грамм, с алгоритмом классификации.

1.4 Обзор классификаторов на основе машинного обучения

Выбор наиболее подходящего алгоритма классификации для решения определенной задачи требует практики и опыта. Каждый классификатор обладает специфическими особенностями и делает некоторые допущения. В действительности следует сравнивать показатели эффективности для нескольких разных алгоритмов машинного обучения для достижения наилучшего результата. На эффективность модели, т.е. на ее способность делать корректные прогнозы, сильно влияет лежащие в основе данные, которые передаются алгоритму классификации для обучения: в них может быть много шума, также достаточно часто классы экземпляров оказываются нелинейно разделимыми, что автоматически усложняет алгоритм.

В данном разделе будет рассмотрено четыре известных классификатора: **логистическая регрессия** (*Logistic Regression*), **метод k-ближайших соседей** (*k-nearest neighbors algorithm – KNN*), **случайный лес** (*Random Forest*) и **XGBoost**. Далее в главе, посвященной практической части дипломной квалификационной работы, будут представлены сравнительные результаты качества работы этих алгоритмов, которые обучались на комментариях, полученных с платформы "JetBrains Academy" по обучению программированию.

1.4.1 Логистическая регрессия

Линейные модели представляют собой класс моделей, которые широко используются на практике, они делают прогноз, используя линейную функцию. Данные модели применяются в задачах регрессии и классификации.

Основная идея линейного классификатора заключается в том, что признаковое пространство может быть разделено гиперплоскостью на два полупространства, в каждом из которых прогнозируется одно из двух значений целевого класса. [15] Если линейное разделение возможно

выполнить без ошибок, то обучающая выборка считается **линейно разделимой**.

Логистическая регрессия (*logistic regression*) очень широко используется в производственной среде для двоичной классификации. Несмотря на свое название, она является именно алгоритмом классификации, а не регрессии.

В основе логистической регрессии лежит вероятностная модель, для пояснения этой идеи введем понятие **коэффициента перевеса**, этот показатель определяет перевес в пользу определенного события. Данный коэффициент может быть записан следующим образом:

$$\frac{p}{(1-p)},$$

где p – это вероятность положительного события, т.е. событие является реальным. Например, если мы хотим спрогнозировать имеется ли у пациента какое-то заболевание или, как в случае этой дипломной работы, узнать относится ли комментарий к классу полезной обратной связи, то мы подразумеваем положительное событие, которому можно присвоить метку класса $y = 1$.

Далее мы определим **логит-функцию** (*logit*) равную натуральному логарифму коэффициента перевеса:

$$logit(p) = \log \frac{p}{(1-p)}$$

Входными значениями логит-функции являются числа в диапазоне от 0 до 1, которые затем трансформируются в значения по всему диапазону действительных чисел. Их можно использовать для выражения линейной зависимости между значениями признаков и логарифмом:

$$logit(p(y = 1 \mid x)) = w_0x_0 + w_1x_1 + \cdots + w_nx_n = \sum_{i=0}^{n} w_ix_i = w^Tx,$$

где $p(y = 1 \mid x)$ – условная вероятность того, что конкретный экземпляр принадлежит классу 1 при определенных признаках x, а w_0 означает член смещения при дополнительном входном значении x_0 равном 1.

Но на самом деле нас интересует предсказание вероятности того, что заданный образец принадлежит к указанному классу, – это, в свою очередь, обратная запись логит-функции, которая называется **логистической сигмоидальной функцией** или просто сигмоидальной, поскольку ее график имеет характерную форму в виде буквы S, как показано на рис. 1.5. Формула записывается таким образом:

$$sig(t) = \frac{1}{1 + e^{-t}}$$

В данном случае t является линейной комбинацией весов и признаков экземпляров: $t = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n = w^T x$.

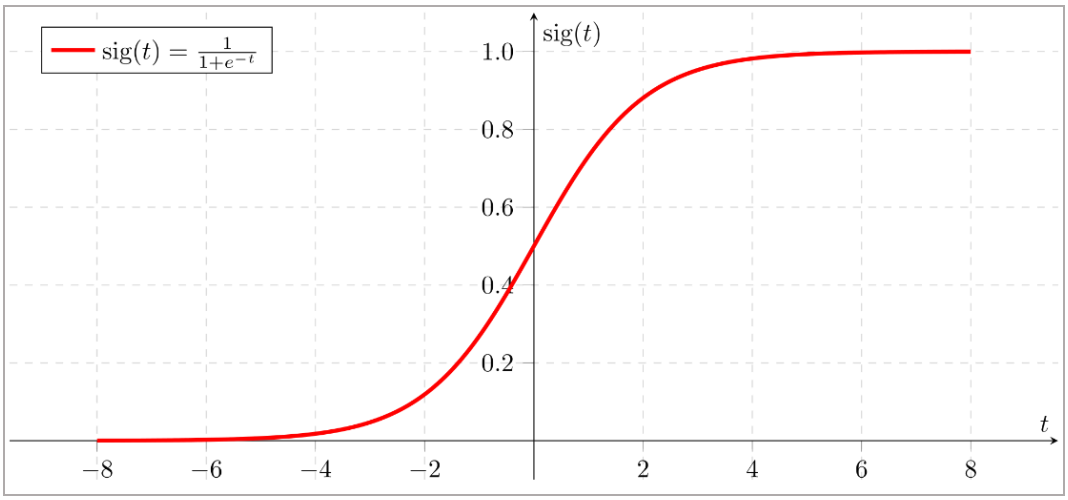

Рис. 1.5 – График сигмоидальной функции

На представленном выше графике видно, что функция $sig(t)$ стремится к 1, когда t стремится к бесконечности, потому что e^{-t} становится слишком малым для больших значений t. Аналогичным образом $sig(t)$ приближается к 0 в результате увеличивающегося знаменателя. Суммируя вышесказанное, сигмоидальной функции на вход подаются вещественные числа и преобразуются в значения в рамках диапазона [0, 1] с точкой пересечения $sig(t) = 0.5$.

Выход данной функции трактуется как вероятность отнесения определенного образца к положительному событию, т.е. к классу 1, при заданных признаках x и подобранных весах w ($sig(t) = P(y = 1 \mid x; w)$).

Значит, если для одного конкретного комментария мы получим $sig(t) = 0.7$, то возможность того, что этот текст является полезной обратной связью, составляет 70%. Соответственно, шанс того, что данный комментарий принадлежит классу 0, может быть вычислен так: $sig(t) = P(y = 0 \mid x; w) = 1 - P(y = 1 \mid x; w) = 0.3$ (или 30%). Предсказанную вероятность после можно трансформировать в бинарный результат с использованием пороговой функции:

$$\hat{y} = \begin{cases} 1, \text{если } sig(t) \geq 0.5 \\ 0 \text{ в обратном случае} \end{cases}$$

Прогнозирование не просто четкого ответа относится ли экземпляр к классу 1 или 0, а именно *вероятности* принадлежности к определенному классу является во многих задачах очень важным бизнес-требованием. Например, в задаче оценки кредитоспособности человека (*кредитный скоринг*), где чаще всего применяется логистическая регрессия, часто необходимо спрогнозировать вероятность невозврата кредита. Клиентов, обратившихся за получением кредита, сортируют по убыванию в зависимости от предсказанной вероятности, и получают рейтинг надежности клиентов. Следовательно, такое свойство алгоритма может быть полезно при работе со сложными данными, где может потребоваться повысить или понизить порог для принятия решения.

Однако при обучении модели логистической регрессии мы фокусируемся не на ошибках, а больше на максимизации вероятности соответствия предсказанных значений наблюдаемым значениям с использованием **оценки максимального правдоподобия** (*Maximum-Likelihood Estimation – MLE*). [16, стр. 319] Кратко рассмотрим, каким образом настраиваются параметры модели, веса w.

Сначала мы определяем функцию издержек – меру того, насколько ошибочна модель с точки зрения ее способности оценивать взаимосвязь между x и y. Обычно она выражается как разница или расстояние между прогнозируемым значением и фактическим значением. [17] Затем

минимизируем ее с целью найти оптимальные веса для классификационной модели. Прежде чем вывести данную функцию для логистической регрессии, определим правдоподобие L, которое желательно поднять до максимального уровня при построении модели с условием, что в наборе данных индивидуальные образцы друг от друга независимы. Формула правдоподобия выглядит так:

$$L(w) = P(y|\, x; w) = \prod_{i=1}^{n} P\left(y^i \middle|\, x^i; w\right) = \prod_{i=1}^{n} \left(sig\left(t^i\right)\right)^{y^i} \left(1 - sig\left(t^i\right)\right)^{1-y^i}$$

Намного легче на практике довести до максимума натуральный логарифм этого уравнения. Такой вариант называется **логарифмической функцией правдоподобия**:

$$\log L(w) = \sum_{i=1}^{n} \left[y^i \log\left(sig\left(t^i\right)\right) + \left(1 - y^i\right) \log\left(1 - sig\left(t^i\right)\right)\right]$$

Использовать логарифмическую функцию лучше, так как она снижает числовую потерю значимости, если величины правдоподобия слишком малы.

Также стоит отметить, что базовая реализации алгоритма в библиотеке Scikit-learn оптимизирует работу алгоритма с помощью **регуляризации**. Она помогает найти компромисс между **переобучением** и **недообучением**. Модель считается переобучившийся, когда она дает качественные результаты на обучающих данных, но плохо обобщается на тестовых данных, которые не встречались ранее. В противоположной ситуации модель будет слишком слаба, чтобы выявить структуру в обучающем наборе данных, и поэтому будет малоэффективной на неизвестных образцах данных.

Метод регуляризации полезен тем, что обрабатывает коллинеарность, сильную взаимосвязь между признаками, фильтрует шум в данных и, следовательно, предотвращает переобучение, когда модель просто запоминает переданные ей обучающие экземпляры. Идея, лежащая в основе данного метода, заключается в том, чтобы сократить сложность модели, штрафуя экстремальные значения весов.

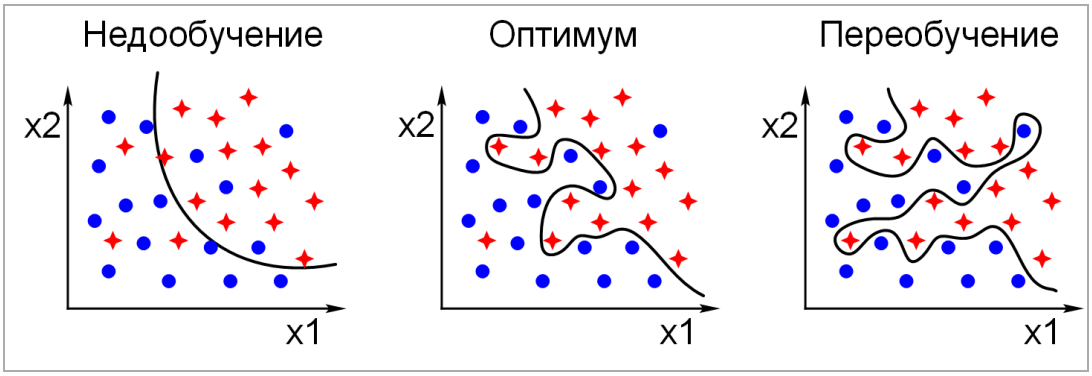

Рис. 1.6 – Проблема переобучения и недообучения

Первая форма регуляризации, которую мы рассмотрим, – это L2-регуляризация, она способствует появлению малых, но ненулевых, весовых коэффициентов модели, и записывается таким образом:

$$\lambda \|w\|_2^2 = \lambda \sum_{j=1}^{m} w_j^2$$

В формуле лямбда обозначает параметр регуляризации, с помощью этого параметра мы можем управлять тем, насколько качественно происходит подгонка к обучающим данным, при этом веса остаются низкими. Следовательно, когда λ увеличивается, повышается и сила регуляризации.

Еще один подход к сокращению сложности модели – регуляризация L1, которую можно представить в следующем виде:

$$\lambda \|w\|_1 = \lambda \sum_{j=1}^{m} |w_j|$$

L1-регуляризация отличается от L2 тем, что она обычно возвращает разряженные вектора признаков, т.е. многие веса станут равны 0 [18, стр. 478]. Таким образом, данную регуляризацию можно воспринимать как прием, позволяющий отобрать важные признаки, так как разреженность на практике может быть, когда мы оперируем данными с большим числом признаков, из которых не все являются релевантными.

Мы можем регуляризировать функцию издержек для логистической регрессии путем добавления, например, члена регуляризации L2, он будет сокращать веса модели в итеративном процессе обучения:

$$\log L(w) = \sum_{i=1}^{n} \left[y^i \log\left(sig\left(t^i\right)\right) + \left(1 - y^i\right) \log\left(1 - sig\left(t^i\right)\right) \right] + \lambda \|w\|^2$$

В результате, издержки приближаются к 0, если мы делаем корректный прогноз, что образец имеет метку того или иного класса. Однако в случае неправильного прогноза, издержки стремятся к бесконечности. Таким образом, при некорректных предсказаниях класса мы штрафуем модель более высокими издержками.

1.4.2 Метод k-ближайших соседей

Алгоритм МО с учителем **k-ближайших соседей** или **KNN** является типичным примером **ленивого ученика** [19]. Данный метод носит такое название не только из-за своей объективной простоты, но и по причине того, что он сохраняет информацию, переданную ему при обучении, без настройки целевой функции и использует ее на тестовых данных. Фактически, KNN не имеет фазы обучения, поскольку он только запоминают всю обучающую выборку, а на этапе классификации новых данных алгоритм напрямую ищет класс точек, которые ближе всего к целевому объекту.

Из этого следует, что алгоритмы МО могут быть разделены на **параметрические** и **непараметрические** модели [20]. Используя параметрические алгоритмы, мы оцениваем параметры на обучающих данных для вывода функции, которая будет способна предсказывать метки новых точек данных, в таком случае обучающий набор данных в дальнейшем не будет требоваться. Стандартным примером группы параметрических моделей является логистическая регрессия, описанная ранее.

Непараметрические алгоритмы напротив не могут быть описаны с помощью конкретного набора параметров, количество которых заранее известно. Их число будет расти вместе с обучающим набором данных. KNN

относится к **метрическим** алгоритмам классификации, подкатегории непараметрических моделей. Из вышесказанного следует, что непараметрические модели не делают явных допущений о глобальных законах, которым подчиняются данные. Единственное допущение, которое делают метрические методы, – это то, что свойства объектов можно узнать, имея представление о его соседях. [21]

Реализацию нашего алгоритма можно разбить на три этапа:

1. Выбрать число *k*, то количество экземпляров обучающей выборки, которые будет рассматривать при классификации нового объекта, и метрику расстояния.
2. Найти *k* ближайших соседей для целевого образца.
3. Назначить метку класса на основе мажоритарного голосования.

На рис. 1.6 показано, как новой точке данных (зеленому кругу) присваивается класс красного треугольника, так как из трех ближайших соседей треугольник набрал большинство голосов, а класс синего квадрата получил лишь один голос.

Таким образом, алгоритм KNN очень легко интерпретировать, кроме того, он быстро адаптируется к появлению новых образцов в обучающей выборке. Однако вычислительная сложность при классификации новых образцов будет линейно расти с увеличением количества обучающих данных, только если эти данные не имеют слишком значительное число признаков или измерений.

В стандартной реализации KNN используют **простое невзвешенное голосование**. При это предполагается, что все *k* экземпляров имеют одинаковое право голоса в независимости от расстояния до классифицируемого объекта. Тем не менее, чем дальше образец обучающего набора данных расположен от классифицируемого объекта в признаковом пространстве, тем ниже его значимость при определении класса. Поэтому для улучшения результатов работы модели можно ввести взвешивание примеров,

при котором вес значимости «соседа» будет уменьшаться при удаленности от целевого объекта. В таком случае используют **взвешенное голосование**. [22]

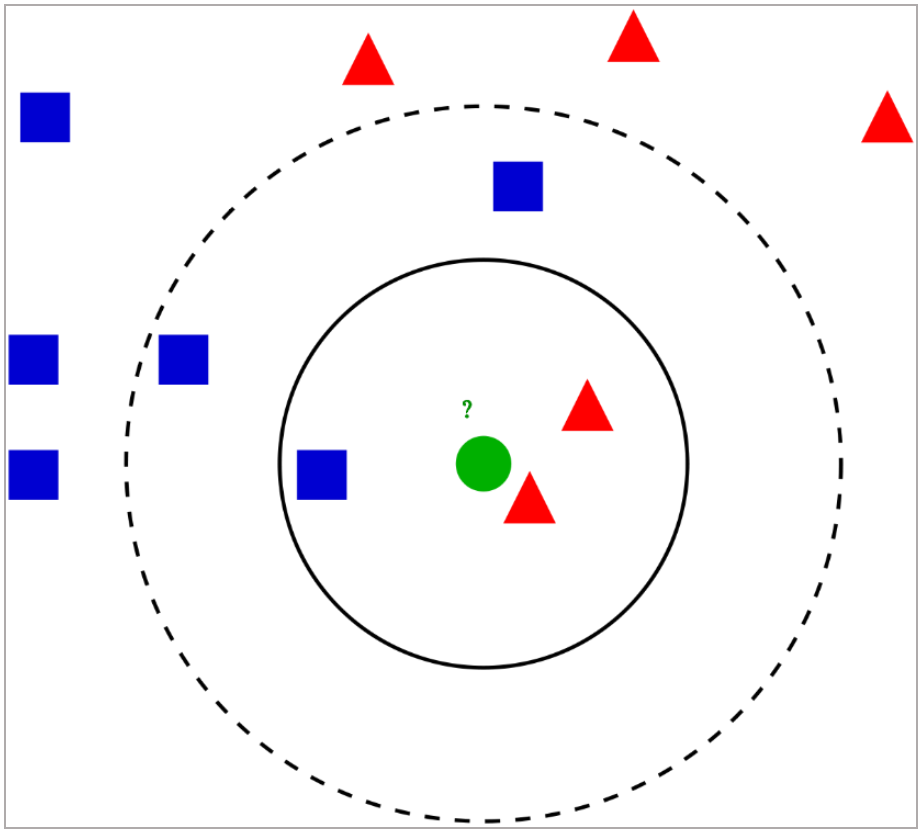

Рис. 1.6 – Назначение тестовому образцу класса треугольника по мажоритарному голосованию

Метод KNN находит в обучающих данных k примеров, наиболее похожие на классифицируемую точку, опираясь на выбранную метрику расстояния. Существует ряд способов, как вычислить расстояние между двумя точками в пространстве, в подавляющем большинстве случаев используется **евклидово расстояние**, вычисляемое по формуле:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Также достаточно часто применяется **манхэттенская метрика**, поскольку в высокомерном пространстве она более устойчива к выбросам. Допустим, существует два почти идентичных объекта в пространстве с тысячей признаков, но при этом они сильно отличаются по одному из

признаков. Следовательно, с большой вероятностью есть выброс в данном признаке, и скорее всего эти объекты все-таки близки. Если бы мы в такой ситуации воспользовались евклидовой метрикой, то различие в единственном признаке усилилось бы и сделало бы объекты дальше друг от друга из-за квадрата в формуле. В манхэттенском расстоянии наоборот это различие нивелируется, так как используется именно модуль:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Если не писать алгоритм самостоятельно с нуля, а использовать готовый метод из библиотеки Scikit-learn для машинного метода, то по умолчанию будет использоваться **метрика Минковского**, которая представляет собой обобщение евклидового (при этом параметр p будет равен 2) и манхэттенского расстояний ($p = 1$), в стандартной реализации $p = 2$:

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

Исходя из вышесказанного становится ясно, что у данного алгоритма есть ряд недостатков: он не слишком эффективен в задачах, где требуется большое количество данных, также чувствителен к неинформативным признакам. Кроме того, для качественной работы алгоритма требуется не только метрическая близость объектов, но и их семантическая близость. Однако у метода KNN есть множество применений в реальном мире, несмотря на его недостатки. Если правильно предобработать исходные данные, то алгоритм можно использовать для поиска похожих по семантике документов, в таком случае темы документов будут схожи при условии, что векторные представления близки друг к другу.

1.4.3 Обучение на основе деревьев решений

Алгоритмы классификации базирующиеся на **деревьях принятия решений**, как мы можем предположить из названия, разделяют данные, задавая определенные вопросы.

В качестве примера рассмотрим дерево на рис. 1.7, показывающее выживаемость пассажиров "Титаника". Цифры под листьями показывают вероятность выживания и процент наблюдений в листе. Посмотрев на диаграмму, можно сделать вывод, что шансы на выживание были высоки, если пассажир трансатлантического парохода был женщиной или маленьким мальчиком с несколькими родственниками на борту. Таким образом, в узлах дерева содержится правило или задаваемый вопрос, в листе, соответственно, находится подмножество объектов, которые удовлетворяют всем правилам ветви, заканчивающейся данным листом.

Используя следующий алгоритм, мы начинаем путь от корня дерева и разбиваем данные по конкретному признаку, который в результате дает наибольший **прирост информации**, далее это понятие будет раскрыто подробнее. Потом процедура рекурсивно повторяется: к каждому подмножеству вновь применяется правило, пока не будет достигнуто некоторое условие остановки работы алгоритма, т.е. мы заранее определяем максимальную глубину дерева, чтобы избежать переобучения. В последнем узле проверка и разбиение не производится, и он объявляется листом, он определяет решение для каждого попавшего в него примера. [23]

При формировании правила разбиения в узлах дерева принятия решений необходимо выбрать признак, по которому это будет сделано. Общее правило заключается в следующем: выбранный атрибут должен разбить множество наблюдений в узле так, чтобы подмножества в результате содержали примеры с одинаковыми метками, указывающими на принадлежность к классу, или были максимально приближены к этому. Для

этого существуют два основных критерия – **теоретико-информационный** и **статистический**.

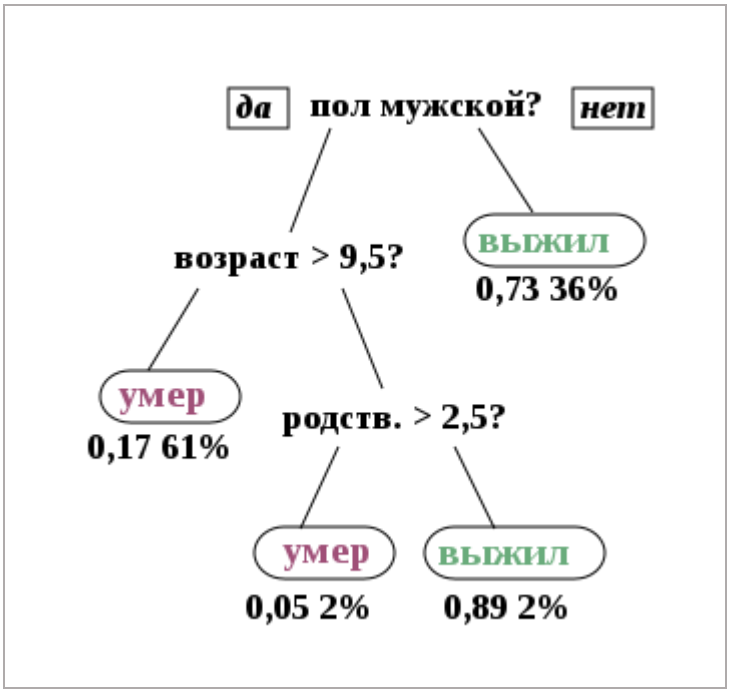

Рис. 1.7 – Выжившие после крушения Титаника

Первый критерий основан на понятиях теории информации, как следует из названия. Заимствованное понятие **информационная энтропия** в анализе данных и МО используется как мера классовой однородности подмножества наблюдений, которые получились при разделении алгоритмом обучающего набора данных на классы. [24] Ее можно написать в виде формулы:

$$H = -\sum_{i=1}^{n} \frac{N_i}{N} \log\left(\frac{N_i}{N}\right),$$

где n – общее число элементов исходного множества, N – количество примеров в подмножестве, и N_i – число примеров, принадлежащих классу i.

Следовательно, лучшим признаком для разбиения будет тот, который максимально снизит энтропию финального подмножества. Однако на практике говорят не об энтропии, а об информации, величине обратной энтропии. Таким образом, оптимальный атрибут разбиения обеспечивает наибольший прирост информации (*information gain*):

$$IG(A) = Info(S) - Info(S_A),$$

где $Info(S)$ указывает на информацию о подмножестве S до разбиения, а $Info(S_A)$ уже определяет информацию о том же подмножестве после разбиения его по атрибуту A.

Статистический подход опирается на использование индекса Джини, данный статистический показатель показывает, насколько часто пример, выбранный из обучающего множества случайным образом, будет распознан некорректно, при условии, что целевые значения этого множества были получены из некоторого статистического распределения. Из этого следует, что индекс Джини является показателем расстояния между двумя распределениями – целевых значений и предсказаний модели, – поэтому логично сделать вывод, что чем меньше это расстояние, тем выше качество работы модели. [25] Показатель можно представить в виде следующей формулы:

$$Gini(Q) = 1 - \sum_{i=1}^{n} p_i^2$$

Здесь Q – имеющееся в результате множество, n – количество классов в нем, p_i – вероятность класса i, выраженный как относительная частота примеров соответствующего класса. Данный индекс, очевидно, варьируется от 0 до 1. Он будет равен 0, если все примеры Q будут отнесены к одному классу, и равен 1 в случае, когда все классы обладают равной вероятностью и представлены в равных пропорциях.

Мы рассмотрели, как устроено одно решающее дерево, и его принцип работы может создать впечатление, что одного такого классификатора было бы достаточно для решения задачи, если бы была подобрана оптимальная глубина дерева. Но на сегодняшней день вместо одного дерева принятия решений широко применяется алгоритм **случайный лес**, поскольку он отличается эффективностью классификации и легкостью масштабируемости. Его можно рассматривать, как **ансамбль** одиночных деревьев. Цель ансамблевых методов заключается в том, чтобы объединить различные

классификаторы в один, который будет обладать лучшей эффективностью обобщения, чем каждый индивидуальный классификатор по отдельности.

Главная идея случайного леса – усреднить множество деревьев решений для построения более надежной модели, которая менее склонна к переобучению и обладает высокой эффективностью обобщения. Для оценки качества работы алгоритма существует разложение на три компонента – смещение (*bias*), разброс (*variance*) и шум (*noise*) – в литературе оно известно как **bias-variance decomposition**. [26] Каждая часть разложенной ошибки модели отвечает за отдельный фактор:

1. **Смещение** – способность модели приблизить лучшую среди всех возможных моделей, например, если пытаться решить абсолютно нелинейную задачу с помощью линейной модели, то обученная модель будет считаться смещенной, т.е. слишком слабой для этой задачи.

2. **Разброс** или **дисперсия** – устойчивость модели к изменениям в обучающей выборке, так деревья решений будут очень сильно меняться в зависимости от переданных данных, следовательно, у них большой разброс.

3. **Шум** – характеристика сложности и противоречивости данных. Если, например, в выборке есть два объекта с одинаковыми признаками, но разными метками, то тогда модель точно не сможет иметь нулевую ошибку.

Из этого следует, что при решении любой задачи методами МО перед нами стоит цель найти такую модель, которая является несмещенной, значит, достаточно сложной, чтобы приспособиться к этим данным, и которая имеет низкий разброс, что позволяет ей найти общие закономерности, а не зависеть от маленьких изменений в выборке. По этой причине одиночные деревья решений не будут наилучшим выбором алгоритма, у них слишком высокая дисперсия. Случайный лес, наоборот, помогает сократить разброс без увеличения смещения.

Алгоритм случайный лес представляет собой комбинацию **бэггинга** (*bagging, bootstrap aggregation*) и **метод случайных подпространств** при

выборе признаков для принятия решений деревьями. Технология бэггинга предполагает, что из обучающей выборки берется n примеров, они выбираются равновероятно, с повторениями. Таким образом, мы получим новую выборку, в которой некоторых элементов исходной выборки может не будет, при этом другие могут войти несколько раз. Затем, повторив процедуру k раз, мы получим k моделей, обученных на k выборках, предсказания которых усредним и получим окончательный ответ. [27] Следовательно, классификаторы не исправляют ошибки других моделей, а компенсируют их при голосовании. Процесс генерации подвыборок с помощью выбора экземпляров исходной выборки с возвращением или повторениями называется **бутстрепом** (*bootstrap*).

Представим алгоритм случайного леса в виде четырех этапов:

1. Сгенерировать бутстреп-выборку из n элементов.
2. Создать дерево принятия решений из полученной выборки, также в каждом узле нужно:
    a. случайным образом выбрать поднабор атрибутов без возвращения из всех признаков, метод случайных подпространств уменьшает корреляцию между деревьями, что позволяет избежать переобучения [28];
    b. разделить узел дерева, используя конкретный признак, обеспечивающий наилучшее разделение, т.е. доводящий прирост информации до максимума.
3. Повторить шаги 1 и 2 k раз.
4. Объединить прогнозы всех деревьев в случайном лесе и выбрать ответ по большинству голосов.

Стоит отметить, разброс модели получается снизить, используя процедуру бэггинга, но на смещение он не будет влиять. Поэтому можно сделать вывод, что смещение должно быть низким именно у деревьев, из которых строится ансамбль. Решение в данной ситуации – это использование

глубоких деревьев, так как они будут подробно запоминать подвыборку данных; из этого следует, что на тестовом наборе предсказания будут сильно варьироваться в зависимости от обучающей подвыборки, но при усреднении прогнозов результат с большой вероятностью будет близок к истине, дисперсия при этом будет высокой, а смещение низким. Неглубоких деревьев наоборот способны запомнить только верхнеуровневые зависимости в обучающей подвыборке, а они будут схожи у всех подгрупп данных, из-за этого целевая зависимость будет описана недостаточно подробно. Изменяя обучающие подвыборки, мы будем получать на тестовом объекте неточные прогнозы модели, но стабильные, следовательно, разброс низкий, но смещение высокое. В результате, данный алгоритм следует использовать с глубокими деревьями, так как ансамблевая модель устойчива к шуму от индивидуальных деревьев.

Единственный гиперпараметр алгоритма случайный лес, который необходимо подобрать – это количество деревьев принятия решений. Как правило, эффективность модели прямо пропорциональна количеству выбранных деревьев за счет увеличения вычислительных затрат.

Также можно оптимизировать другие гиперпараметры классификатора, например, размер бутстрэп-выборки и количество признаков, выбираемые случайным образом для каждого разделения. Выбирая размер бутстрэп-выборки, мы управляем компромиссом между смещением и дисперсией случайного леса. Если мы уменьшаем размер данной выборки, то это приведет к увеличению различий между одиночными деревьями, поскольку шанс того, что конкретный образец появится в бутстрэп-выборке, становится маленьким. Хотя такое решение может ослабить переобучение посредством увлечения произвольности случайного леса, но чаще всего это приводит к понижению общей эффективности модели. И обратная процедура, увеличения размера бутстрэп-выборки, наоборот с большой вероятностью может увеличить степень переобучения, так как индивидуальные деревья станут слишком похожи друг на друга.

Кроме того, мы можем управлять числом признаков, используемых при обучении одного дерева, и тем самым влиять на качество случайного леса. Чем больше признаков, тем меньше чувствуется эффект работы ансамбля, потому что увеличивается корреляция между одиночными деревьями. Соответственно, уменьшая число признаков, мы делаем сами деревья слабее. На практике для классификации рекомендуется брать корень из числа всех признаков для определения максимального количества признаков. [29]

1.4.4 XGBoost

В последнем параграфе первой главы мы рассмотрим самый актуальный алгоритм на сегодняшний день, который использует вспомогательный алгоритм **бустинга** (*boosting – усиление*). Идея бустинга заключается в том, чтобы при обучении модели последовательно минимизировали ошибки друг друга. По причине того, что предсказатели обучаются на ошибках, совершенных предыдущими, понадобится меньше времени для достижения правильного ответа. Данный подход, как и другие ансамблевые методы, преобразует нескольких слабых учеников в сильную модель и ищет компромисс между смещением и дисперсией. [30, с. 124]

Прежде чем перейти к основному алгоритму XGBoost, кратко рассмотрим несколько других алгоритмов бустинга и их особенности. Разновидность бустинга **AdaBoost** (*adaptive boosting – адаптивный бустинг*) концентрирует внимание на недообученности модели, из этого следует, что алгоритм при каждом новом предсказании будет фокусироваться на сложных подвыборках данных, чья классовая принадлежность неочевидна. Допустим, мы хотим построить классификатор AdaBoost, где решающее дерево является основным алгоритмом, тогда веса ошибочно классифицированных элементов, по принципу адаптивного бустинга, будут возрастать. На следующей итерации обучения алгоритм будет обучаться на обновленных коэффициентах. В таком случае общая схема работы адаптивного бустинга выглядит следующим образом: весовые коэффициенты увеличиваются

именно у неправильно классифицированных объектов обучающей выборки, чтобы последующая модель смогла сделать для них правильные предсказания. AdaBoost из-за своей техники последовательного обучения напоминает метод градиентного спуска, стремящийся найти минимум функции издержек, для этого он делает касательную линию к функции и итеративно пытается продвигаться в отрицательном направлении наклона, в сторону глобального минимума функции. [31] Адаптивный бустинг отличается тем, что он изменяет параметры или веса не только одного предиктора, а собирает модели в ансамбль, который постепенно улучшает. Но большой недостаток данного метода – это отсутствие возможности параллельного обучения моделей, так как каждый из предиктор начинает обучение только после того, как получил сведения от предыдущего, окончившего цикл обучения.

Другой наиболее общий алгоритм бустинга – **градиентный**, все современные методы бустинга, так или иначе, являются его модификациями или частными случаями. [32, стр. 5] Его принцип работы схож с описанным выше инструментом AdaBoost. Но градиентный бустинг отличается от адаптивного, который изменяет параметры модели при каждой итерации, тем, что он двигается к минимуму функции потерь, опираясь на остаточную ошибку прошлой модели. Таким образом, он минимизирует ошибку алгоритмом градиентного спуска, т.е. подбирает коэффициенты так, чтобы функция потерь (разность между истинным значением и прогнозом модели) стала минимальной и, следовательно, предсказание было максимально приближено к правильному. Градиентный бустинг реализуется таким образом:

1. сначала модель строится на обучающей выборке и делает предсказания для всего набора данных;
2. затем по истинным ответам и предсказанным вычисляются ошибки;

3. для новой модели эти ошибки будут целевой переменной, она будет делать новые прогноз ы, которые будут сочетаться с предсказаниями предыдущих моделей;

4. на следующем этапе снова вычисляются ошибки с опорой на правильные значения и предсказанные;

5. обучающий процесс закончится, если функция потерь не изменится или если будет достигнуто максимальное число предикторов. [33]

Алгоритм **XGBoost** (*extreme gradient boosting*) или **экстремальный градиентный бустинг**, в свою очередь, является продвинутой реализацией градиентного бустинга, используемого в сочетании с деревьями принятия решений, которые мы обсуждали в прошлом параграфе. [34]

Данный инструмент является исследовательским проектом Вашингтонского университета, он был представлен в 2016 году и произвел сильное влияние на сферу МО, поскольку он обладает высокой предсказательной способностью и намного быстрее других методов бустинга, так как нем есть возможность параллельного обучения.

Из всего вышесказанного следует, что экстремальный градиентный бустинг – это ансамблевый метод деревьев, использующий принцип бустинга слабых учеников (в нашем конкретном случае это бинарные деревья решений, в узлах которых ветвление может вестись только в двух направлениях, т.е. осуществляется выбор из двух альтернатив (полезный комментарий или нет) [35]) благодаря градиентному спуску, также он улучшает архитектуру Gradient Boosting Machines (*GBM – альтернативное название градиентного бустинга*) при помощи системы оптимизации.

Данный алгоритм был разработан для улучшения эффективности вычислительных ресурсов и сокращения затрат времени и памяти. Кроме параллелизации, XGBoost обеспечивает вычисления для очень больших наборов данных, которые не могут поместиться в память компьютера. Также он отсекает ветви деревьев, используя параметр максимальной глубины (`max_depth`). Экстремальный градиентный бустинг сначала строит деревья

размером с `max_depth`, что обеспечивает быстрое обучение, так как не требуется оценивать параметры регуляризации в каждом узле. После того, как каждое дерево будет увеличено до `max_depth`, алгоритм проходит рекурсивно от нижней части дерева до самого верха и проверяет, удовлетворяют ли разделения условиям, установленным в гиперпараметрах. Если разделение или узел не соответствует требованиям, то он удаляется из дерева. [36]

Исходя из перечисленных преимуществ современного алгоритма XGBoost, мы можем сделать предположение, что на практике он будет давать более хорошие результаты, чем его предшественник случайный лес.

Глава 2. Программа для классификации комментариев на платформе "JetBrains Academy"

Комментарии на платформе "JetBrains Academy"

Обучающая платформа по программированию "JetBrains Academy" предоставляет пользователям широкий ряд курсов, посвященных изучению таких языков программирования, как Python, Java и Kotlin. Но с приростом студентов, количество обратной связи в виде комментариев сильно увеличивается, обработка такого поток информации вручную является достаточно ресурсозатратным процессом.

Ответная реакция пользователей на теоретический материалы и задания на платформе очень важна, поскольку с их помощью можно выявить недоработки в содержимом сайта.

Компанией была предложена идея попробовать автоматизировать процесс сортировки обратной связи, для этого они подготовили датасет комментариев, размещенных под материалами о языке программирования Python. Был выбран только один раздел обучающих материалов, так как Python – это самый популярный язык для изучения на их сайте, они осветили 270 тем, связанных с ним. Кроме того, этот проект является тестовым и было решено, что сначала проверка алгоритма выполнится только для одного языка, ведь у каждого языка программирования свои синтаксические особенности, поэтому трудно сделать универсальный инструмент.

В представленном датасете кроме самих текстов есть дополнительная метаинформация, рассмотрим самые релевантные для нас аспекты:

- метка комментария (*Wont Fix* – бесполезная обратная связь, *Fixed* и *Pending* – полезные комментарии, на основе которых были выполнены исправления, либо они планируются, *New* – текст, у которого еще нет метки полезный/бесполезный);

- раздел, где был оставлена обратная связь, например, под задачей, где пользователь должен сам написать код, либо под теоретическим материалом;
- ссылка на страницу сайта, где был размещен комментарий;
- количество ответов других пользователей на данный комментарий;
- негативная реакция пользователей на комментарий, если он является неуместным, т.е. автор текста написал о том, что не имеет отношения к теме.

Таким образом, изучив набор данных, было принято решение сосредоточиться только на обратной связи, относящейся к разделу с кодовыми задачами, где чаще всего у пользователей возникают проблемы. С учетом этих критериев обучающая выборка составила 11 тысяч 299 размеченных комментариев, из которых 655 имеют метку *Fixed*.

Цель нашей работы заключалась в том, чтобы отсеивать комментарии, представляющие собой бесполезную обратную связь, так как на их основе нельзя внести поправки в обучающие материалы. Например, следующие комментарии будут иметь метку *Wont Fix*: «*You have made a mistake in the task a = 10*», «*Why only a few people liked this topic? Such good quality platform must have more, than 77 learners!*». Полезная обратная связь наоборот будет указывать на недостатки: «*strange behavior in test case #4,418723*», «*I think this problem should be formulated as «Select all the strings that follow the PEP8 guidelines» or somehow similarly.*».

В следующем разделе будет показано, как подготовить данные к последующей автоматической сортировке.

Предварительная обработка текстовых комментариев

Прежде чем перейти к алгоритмам МО, нужно выполнить предобработку датасета комментариев. Для выполнения следующей задачи

было релизовано несколько функций. Первая из них, `preprocessing`, принимает на вход датафрейм, двумерную структуру данных с строками и столбцами, чтобы удалить из него лишние тексты, несоответсвующие некоторым критериям.

Во-первых, для обучения мы оставляем комментарии только на английском языке, поскольку все материалы на платформе "JetBrains Academy" написаны именно на нем. Также будет выполнена дополнительная проверка на символы кириллицы, так как комментарии русских студентов могут содержать буквы двух алфавитов, из-за этого модель `fasttext` может ошибочно принять текст за англоязычный.

```
import fasttext

def preprocessing(df):
    # фильтруем по количеству ответов на комментарий
    df = df.loc[df['replies'] == 0].loc[df['abuses'] == 0]
    # фильтруем по длине
    for text in df['text']:
        text_split = set(text.split())
        if len(text_split) <= 5 and 'typo' not in text_split:
            df = df.drop(df.loc[df['text'] == text].index)
        # дополнительна проверка на кириллицу
        if len(re.findall(r'[а-яА-ЯёЁ]', text)):
            df = df.drop(df.loc[df['text'] == text].index)


    # оставляем отзывы только на анлийском языке
    model = fasttext.load_model('lid.176.ftz')
    delete_text = set()

    for text in df['text']:
        text = text.replace('\n', '').replace('\r', '')
        if model.predict(text, k=1)[0][0] != '__label__en':
            delete_text.add(text)

    for text in df['text']:
        if text in delete_text:
            df = df.drop(df.loc[df['text'] == text].index)

    return df
```

Рис. 2.1 – Реализация функции `preprocessing`

Во-вторых, необходимо отсеять комментарии, на которые отвечали другие пользователи, потому что в большинстве случаев это говорит о том, что учащиеся обсуждали задачу или кому-то требуется помощь в решении. Кроме того, удалим из исходного датасета комментарии, которые пользователи платформы посчитали спамом, нежелательными сообщениями. Также в функционале `preprocessing` есть проверка на длину комментария: если в тексте меньше пяти слов, то он является неактуальным для нас. Обычно в коротких комментариях выражают благодарность, либо говорят, что задание слишком сложное или, наоборот, простое (например, «*too easy*», «*Thanks for fixing it.*»). Но если в тексте встретится слово «*typo*», даже если по длине он будет меньше пяти слов, то такой комментарий не будет удален, потому что это слово свидетельствует об опечатке.

Вторая функция `comment_preprocessing` принимает на вход только один комментраий и будет обрабатывать ключевую особенность текстов, связанных с темой программирования. Такие данные совмещают в себе естественный язык и формальный язык программирования, что значительно осложняет обучение алгоритма МО. Более того, иногда синтаксис Python может быть схож с английским языком, что может привести к искажению семантического смысла комментария при их векторизации. Таким образом, все, что не имеет отношение к естественному языку будет закодировано специальными обозначениями *[code]* (если это пример кода на языке Python) и *[number]* (если это число). На рис. 2.2 изображены два списка со стоп-символами, в первом находятся ключевые слова в питоне и различные математические операторы, которые необходимо перезаписать в исходном комментарии как *[code]*; во втором были дополнительно прописаны эмотиконы, широко распространенные в Интеренете.

Но двух таких списоков будет недостаточно, поэтому в `comment_preprocessing` мы также воспользуемся популярным инструменом – регулярными выражениями (*regular expressions*). Модуль `re`

в Python служит стандартным интерфейсом для сопоставления текста, представленного в виде символьных строк, с шаблонами регулярных выражений [37, с. 263].

```
python_key_symbols = [
    'False', 'True', 'None', 'assert', 'range', 'break',
    'continue', 'enumerate', 'def', 'del', 'elif', 'else',
    'except', 'len', 'str', 'set', 'list', 'yield', 'class',
    '==', '!=', '<', '>', '<=', '>=', '|', '(', ')', '[',
    ']','{', '}', 'return', 'print', ':=', '=', '/', '//',
    '*', '**', '%', '+', '+=', '-=', '*=', '/=', '//=', '%=',
    '**=', 'LF', 'CRLF','x', 'y', '__main__', 'sep', 'ind', 'try:', 'err', '^'
]

emoticons = [
    '¯\_(ツ)_/¯', '^_^', ':)', '/ ',
    ';)', ':D', 'x_x', ':(', '=)'
]
```

Рис. 2.2 – Ключевые символы в Python и эмотиконы

С помощью метода `re.sub` можно найти в тексте комментария определенные шаблоны и заменить их, например, на *[code]*. В коде функции `comment_preprocessing` (см. Приложение 1) прописано большое количество шаблонов, рассмотрим несколько из них: шаблон `r'__\w+__'` способен находить такие стандартные конструкции в Python, как `__init__` или `__class__`; используя паттерн `r'\w+\.\w+\s'`, можно найти код, где обращаются к аттрибуту объекта класса (`obj.value`). Число с решеткой (`#4`) заменяется на *[number]*, если будет найдено совпадение по шаблону `r'#\s*(\d)+'`.

Результат работы данной функции будет следующий: если мы передадим ей строчку «*For those who has troubles with that task: just type random.seed(43) as it mentioned in the task before the needed random function.*», то на выходе она преобразуется в строку «*for those who has troubles with that task : just type [code] as it mentioned in the task before the needed random function .*», где слова приведены к нижнему регистру, а знаки препинания отделены от слов. Таким образом, применив две вышеописанные функции,

мы очистили нашу исходную базу данных и предобработали «сырые» тексты комментариев.

Векторизация текстовых комментариев

Алгоритмы машинного обучения могут использовать только числовые данные, поэтому при решении любой задачи из области обработки естественного языка, текстовые данные сначала необходимо преобразовать в вектор числовых признаков с помощью процесса, известного как **векторизация**.

Статистическая мера **tf-idf** – широко используемый способ векторизации при анализе текстов, основанный на частоте встречаемости слов. Используя этот показатель, можно оценить важность слов. При анализе текстовых данных мы всегда сталкиваемся со словами, которые неоднократно появлялись в противоположных множествах, в нашем случае, в комментариях обоих классов – полезные и бесполезные. Такие часто встречающиеся слова не содержат полезной или различительной информации, поэтому нам стоит применить меру под названием **частота терма – обратная частота документа** (*term frequency-inverse document frequency – tf-idf*), которая применяется для снижения веса признаков с высокой частотностью в векторах. Данная функция уже встроена в класс `TfidfVectorizer` подмодуля библиотеки Scikit-learn, ее не нужно реализовывать самостоятельно, но все же стоит сказать несколько слов о принципе работы данной меры.

Меру tf-idf можно определить как произведение частоты терма на обратную частоту документа:

$$tf - idf(t,d) = tf(t,d) \times idf(t,d)$$

Здесь $tf(t,d)$ или **сырые частоты термов** (*raw term frequency*) - сколько раз терм t встречается в документе d, а $idf(t,d)$ – обратная частота документа, вычисляемая по формуле:

$$idf(t,d) = log\frac{1 + n_d}{1 + df(d,\ t)}$$

В уравнении выше n_d – общее количество документов, а частота документа $df(t,d)$ – количество документов d, содержащих терм t. Константа 1 добавляется к знаменателю с целью присвоить ненулевые значения термам, но не является обязательной; log используется для того, чтобы гарантировать, что низким частотам документов не дается слишком большой вес. [12, стр. 320]

Еще одним способом представления текстовых данных в числовом формате является **эмбеддинги** – прием в языковом моделировании, когда одиночные слова или группы слов сопоставляются числовым векторам, сохраняющим семантическую связь.

В данной выпускной квалификационной работе используются вектора **GloVe** (*Global Vectors*), но стоит пояснить, почему была выбрана именно эта модель, а не **Word2Vec** или **W2V**. NLP-модели GloVe и W2V тесно связаны друг с другом – обе опираются на интерпретируемости векторов слов.

Эффективный метод W2V для извлечения векторных представлений слов, предложенный Томасом Миколовым в 2013 году, основывается на предположение о том, что слова обладают схожими значениями, если они часто находятся в одинаковых контекстах [38].

Главное отличие GloVe от W2V заключается в том, что данная модель учитывает совместную встречаемость слов, в то время как W2V использует совместную встречаемость для создания дополнительной обучающей выборки. Таким образом, в GloVe векторы слов группируются вместе с опорой на их глобальную схожесть. [39] Из этого следует, что NLP-модель GloVe предпочтительнее чем W2V, поскольку она учитывает, что значение слово может меняться в зависимости от контекста его использования. Поэтому в данной работе были применены уже предобученные вектора для английского языка, которые доступны на веб-сайте Стэнфордского университета.

Последний инструмент для векторизации текстов, который был применен в данной работе, – эмбеддинги **LaBSE** (*Language-Agnostic BERT Sentence*), отличительная особенность этого инновационного продукта от Google AI заключается в том, что мы получаем векторное представление не одного слова, а целого предложения. Следовательно, если мы будем использовать алгоритм векторизации LaBSE в решении нашей задачи, то у близких по смыслу комментариев будут схожие вектора. Модель создает не зависящие от языка эмбеддинги предложений для более чем 100 языков, она также обучена генерировать аналогичные вектора с сохранением семантики для двуязычных пар предложений, которые являются переводами друг друга. [40]

Решеточный поиск

В МО существует два вида параметров: первые определяются при обучении, например, подбор весов в логистической регрессии, а вторые параметры алгоритма оптимизируются отдельно. Последние мы кратко упоминали в первой главе, они также называются **гиперпараметрами**. В логистической регрессии одним из гиперпараметров является выбор регуляризации L1 или L2, в дереве решений это будет параметр глубины.

Решеточный поиск (*grid search*) – один из популярных приемов оптимизации гиперпараметров, его суть заключается в том, чтобы найти наилучший комбинацию значений гиперпараметров для повышения эффективности модели.

С помощью модуля `sklearn.model_selection` мы инициализируем объект `GridSearchCV`, которому в дальнейшем передадим список с рядом значений для различных гиперпараметров, затем компьютер будет проводить оценку эффективности всех возможных комбинаций. Рассмотрим пример для настройки алгоритма `RandomForestClassifier`, передадим объекту `GridSearchCV` необученную модель вместе со словарем `grid_params_rf`, где указаны гиперпараметры, которые мы

хотим подобрать, и их варианты значений. Далее проведем подбор оптимальных настроек на обучающих данных, например, на эмбеддингах LaBSE.

```
grid_params_rf = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': ['sqrt', 'log2', None],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000],
    'criterion': ['gini', 'entropy', 'log_loss']
}

gs_rf = GridSearchCV(
    RandomForestClassifier(), param_grid=grid_params_rf, verbose=1, cv=3, n_jobs=-1
).fit(torch.cat(X_train_labse), y_train)
```

Рис. 2.3 – Подбор гиперпараметров для алгоритма случайный лес

Когда выше проиллюстрированный код закончит выполняться, мы можем посмотреть наилучшую комбинацию через атрибут `best_params_` у нашей переменной `gs_rf`. Для данного примера наилучшая комбинация будет выглядеть так:

`{'bootstrap': True, 'max_depth': 100, 'max_features': 'sqrt', 'min_samples_leaf': 3, 'min_samples_split': 8, 'n_estimators': 100, 'criterion': 'entropy'}`.

Оценка работы моделей

На заключительном этапе нашей работы мы будем передавать алгоритму подготовленную набора данных для валидации, правильные метки которой не будут показываться системе. В свою очередь алгоритм для каждого элемента валидационной выборки будет определять, к какому конкретному классу относится комментарий.

Но этого будет недостаточно, оценивать работу нам нужно только по численным показателям. Для этой задачи мы можем использовать самые популярные метрики, применяемые в задачах классификации: **точность**

(*precision*), **полнота** (*recall*) и **F-мера** (*F-score*). Данные метрики с легкостью можно импортировать из модуля `sklearn.metrics`.

Точность работы системы в рамках одного класса показывает долю экземпляров, которые действительно принадлежат данному классу относительно всех документов, отнесенных классификатором к этой группе. Полнота – это процент найденных алгоритмом документов, принадлежащих классу от общей доли всех документов этой группы в выборке. Мы можем сгруппировать возможные ответы системы так:

- **TP** (*true positives*) или **истинно положительные** – корректные предсказания системой, т.е. получили на выходе те, что ожидали;
- **TN** (*true negatives*) или **истинно отрицательные** – категории, которые правильно отсутствуют в предсказании классификатором, поскольку их там быть не должно;
- **FP** (*false positives*) или **ложноположительные** – те категории, которые мы не ожидали на выходе, но анализатор их ошибочно вернул;
- **FN** (*false negatives*) или **ложноотрицательные** – ложно определенные анализатором категории, так как мы их не ожидали увидеть в предсказании.

Тогда, точность и полнота определяются следующим образом [41, стр. 1809]:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

F-мера основывается на точности и полноте, она представляет собой гармоническое среднее между точностью и полнотой. Если точность или полнота стремится к нулю (что является показателем ухудшения

эффективности модели), то данная мера будет, соответственно, тоже стремиться к нулю. По следующей формуле F-мера будет падать одинаково как при уменьшении, так и точности, так как полноте и точности придаются одинаковые веса:

$$F = 2 \times \frac{precision \times recall}{precision + recall}$$

Последняя метрика, которую стоит упомянуть, – это *accuracy*, самая простая оценка работы классификатора, поскольку она показывает только долю правильно классифицированных объектов. На самом деле, можно воспринимать эту меру как вероятность того, что класс будет предсказан корректно. Данная оценка вычисляется по формуле:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Но если оприорная вероятность у одного класса высокая, то метрика *accuracy* может искажать представление о результатах, поэтому когда есть дисбаланс в данных, необходимо применять `balanced_accuracy_score`, функцию из модуля `sklearn.metrics`.

Теперь, когда мы рассмотрели основные метрики качества бинарной классификации, можно изучить таблицы результатов работы каждого алгоритма. В каждой из четырех таблиц (см. Приложение 2-5) сравнивается эффективность одного алгоритма с тремя вариантами векторизации текстовых данных, насыщенным шрифтом выделена наилучшая комбинация. Изучив все таблицы, мы приходим к выводу, что наилучший вариант – это XGBoost классификатор в совокупности с LaBSE эмбеддингами:

accuracy = 0.53

precision = 0.72

recall = 0.78

F-score = 0.72

Все метрики были рассчитаны с учетом дисбаланса данных, потому что при обучении алгоритма было использовано 655 комментариев с меткой

Fixed и 2500 – с *Wont Fix*. Но даже самый эффективный вариант алгоритма из всех двенадцати комбинаций плохо справляется с правильной классификацией, как можно видеть по метрике *accuracy*. Это объясняется тем, что данные очень сложные по своей структуре: тексты достаточно короткие, в них часто неясен контекст и они совмещают в себя два типа языка – естественный и формальный. Предугадать все возможные варианты комментариев и подобрать для них шаблоны регулярных выражений, которые бы смогли отделить код на Python от обычно тексат, практически невозможно, поэтому было принято решение добавить проверку на наличие определенных n-грамм.

Словарь n-грамм

Ранее в первой главе говорилось о том, что можно использовать не только алгоритмы машинного обучения, но и n-граммы с определенной эмоциональной окраской. В нашем случае, мы предполагаем, что пользователи, если они чем-то недовольны, они будут выражать это в своих комментариях оценочными словами (например, «*confused*», «*ambiguous*», «*unclear*», «*senseless*»). Также из уже размеченных данных были извлечены следующие цепочки слов, которые указывают на то, что текст имеет метку *Fixed*: «*please improve task*», «*would be nice to*», «*improve the wording*», «*need fix it*», «*wrong task description*», «*can't be solved without*». В результате, мы собираем в один словарь n-граммы, которые с большой вероятностью могут появиться в полезной обратной связи, и, основываясь на них, делаем предсказания класса комментария с помощью функции `prediction`:

```
def prediction(string, ngramms):
    for target_string in ngramms:
        if target_string in string and 'you' not in string:
            return True
    return False
```

Рис. 2.4 – Реализация функция `prediction`

В функции делается проверка не только на совпадение n-грамм, но также дополнительно прописано условие, что в тексте не должно быть упомянуто местоимение «*you*» во избежание появления комментариев, которые являлись частью диалога между пользователями.

Данное дополнение к основному алгоритму МО позволяет немного повысить метрику *accuracy*, до 0.6.

Окончательный вариант алгоритма

В финальной версии алгоритма, в функции `comments_sorting` (см. Приложение 6), датафрейм с текстами и метаинформацией, который подается на вход, очищается при вызове `preprocessing`, затем комментариям присваиваются метки класса. Алгоритм объединяет два подхода к сортировке комментариев: классификатор XGBoost, в который передаются данные, векторизованные с использованием инструмента LaBSE, и словарь с n-граммами. Таким образом, у нас будет два списка с предсказаниями. Первый метки мы получаем при проверке наличия в комментарии определённой лексики из словаря – если есть совпадение, то текст считается полезной обратной связью. Второй список возвращает сам алгоритм XGBoost, который классифицировал обработанные с помощью функции `comment_preprocessing` комментарии. Затем, основываясь на метках из двух полученных списков с результатами, мы извлекаем только те тексты, которым был присвоен статус полезных, и возвращаем их в виде датафрейма вместе со ссылками на страницы сайта, где они были оставлены студентами.

Для проверки качества вышеописанного алгоритма возьмем случайным образом сорок комментариев с меткой *New* и пропустим их через нашу функцию. Она вернет четыре комментария, которые можно считать полезными:

1. «***It seems strange to give this problem before studying*** *for-cycle. Of course, as the list is predefined, the student can just check all possible indices explicitly, but it's certainly not beautiful*»

2. «***Can't be solved without*** *control structures, that is* ***not learned yet***.»

3. «*Oh, I think they meant "if b >= maximum:" It seems to be just a* ***typo***»

4. «*"from an operation ... with the lowest priority to an operation ... with the highest priority" that means the opposite order*»

Три комментария из перечисленных точно являются полезной обратной связью, поскольку в них есть определенные конструкции, которые указывают на это, они выделены насыщенным шрифтом. В последнем примере из четырех студент скорее всего указывал на то, как правильно выполнить задание, следовательно, такой текст не является для нас релевантным. Но если посмотреть на все остальные комментарии из сорока переданных алгоритму, то станет ясно, что все лишнее было отсеяно правильно. Например, функция посчитала следующие сообщения бесполезной обратной связью: «*x = 2 y = 2 print(f"{x} + {y} = {x + y}")*», «*@andioz yes it is in theory*», «*That is a character entity reference tag for HTML. Bunch of those errors on this website. "If you use the less than (<) or greater than (>) signs in your text, the browser might mix them with tags. Character entities are used to display reserved characters in HTML."*», «*You can try this examples in your computer, and see what you will get.*». Единственный комментарий, чья классовая принадлежность может вызвать сомнения, – «*I am getting the same as you. can't pass test #5 as well.*». Есть два возможных варианта для определения класса – неполадки в тестах (полезный) или невнимательность пользователей при решении задачи (бесполезный).

Таким образом, можно прийти к заключению о том, что алгоритм работает достаточно эффективно, он отсеивает большую часть неактульных комментариев и сохраняет важную обратную связь для сотрудников компании. В полученном результате также виден дисбаланс в данных, который мы наблюдали в обучающей выборке: полезные сообщения

пользователей составляет очень маленький процент от всех комментариев на платформе.

Заключение

В современном мире существует возможность автоматизировать многие аспекты нашей жизни и работы, поэтому в данной выпускной квалификационной работе был произведен эксперимент по сортировке комментариев на обучающей платформе по программированию "JetBrains Academy" с помощью алгоритмов машинного обучения с учителем.

В первой главе освещалась тема естественного языка и как тексты на нем можно обрабатывать и классифицировать для различных задач. Также были подробно рассмотрены четыре алгоритма классификации: логистическая регрессия, метод k-ближайших соседей, случайный лес и XGBoost. Некоторые из них являются достаточно простыми и базовыми инструментами в машинном обучении, другие применяют более сложные и современные технологии. Во второй главе было рассказано о сравнительных результатах каждого из них.

Кроме того, в данной работе рассказывается о проблеме сочетания в одном тексте естественного языка и формального, которым является любые языки программирования. Были предложены способы очистки и подготовки данных для обучения классификатора: в практической части работы были рассмотрены три способа векторизации текстов (tf-idf, предобученные эмбеддинги GloVe и эмбеддинги предложений LaBSE), также были показаны примеры, как удалить из комментариев фрагменты кода с помощью регулярных выражений, и провести дополнительную фильтрация по метаинформации, указанной в исходном наборе данных.

Финальный алгоритм показал хорошие результаты на неразмеченных данных, поскольку он использует не только автоматический классификатор, но и словарь с n-граммами.

На примере проведенного эксперимента было доказано, что при работе с такими сложными данными и достаточно субъективной разметкой, проведенной сотрудниками компании, необходимо основательно изучить и подготовить датасет с текстами, чтобы избежать переобучения алгоритма

машинного обучения, и, возможно, дополнить работу другими лексическими ресурсами.

Список источников

1. Бенгфорт, Б. Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка. [Текст]: учеб. пособие / Б. Бенгфорт, Р. Билбро, Т. Охеда – СПб.: Питер, 2019. – 368 с.
2. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика [Текст]: учеб. пособие / Е. И. Большакова, Э. С. Клышинский, Д. В. Ландэ и др. – М.: МИЭМ, 2011. – 272 с.
3. Грас, Д. Data Science. Наука о данных с нуля [Текст]: учеб. пособие / Д. Грас – СПб.: БХВ-Петербург, 2021. – 416 с.
4. Mason H. The Next Generation of Data Products [Электронный ресурс]. – URL: https://www.slideshare.net/continuumio/the-next-generation-of-data-products-anacondacon-2017 (дата обращения: 07.03.2022)
5. Loukides, M. What is data science? [Электронный ресурс]: статья. – URL: https://www.oreilly.com/radar/what-is-data-science/ (дата обращения: 10.03.2022)
6. Bengfort, B. The Age of the Data Product. [Электронный ресурс]: статья. –URL: https://www.districtdatalabs.com/the-age-of-the-data-product (дата обращения: 10.03.2022)
7. Силен, Д. Основы Data Science и Big Data. Python и наука о данных. [Текст]: учеб. пособие / Д. Силен, А. Мейсман, А. Мохамед – СПб.: Питер, 2017. – 336 с.
8. Кантор, В. Просто о сложном: как человек учит машину учиться [Электронный ресурс]: статья. – URL: https://theoryandpractice.ru/posts/12577-machine-learning (дата обращения: 15.03.2022)
9. Букия, Г. Т. Анализ тональности [Текст]: статья / Г. Т. Букия, Е. В. Протопопова // Прикладная и компьютерная лингвистика. – М.: ЛЕНАНД, 2017. – С. 123-139.
10. Kumar, A. Model Selection Management Systems: The Next Frontier of Advanced Analytics [Электронный ресурс]: статья / A. Kumar, R. McCann, J. Naughton, J.M. Patel. – URL: https://adalabucsd.github.io/papers/2015_MSMS_SIGMODRecord.pdf (дата обращения: 21.03.2022)
11. Wickham, H. Visualizing Statistical Models: Removing the Blindfold [Электронный ресурс]: статья / H. Wickham, D. Cook, H. Hofmann. – URL: http://had.co.nz/stat645/model-vis.pdf (дата обращения: 22.03.2022)
12. Рашка, С. Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow [Текст]: учеб. пособие / С. Рашка, В. Мирджалили – СПб.: Диалектика, 2019. – 656 с.

13. Хохлова, М. В. Анализ тональности [Текст]: статья // Прикладная и компьютерная лингвистика. – М.: ЛЕНАНД, 2017. – С. 245-258.

14. Гаршина, В. В. Разработка системы анализа тональности текстовой информации [Электронный ресурс]: статья / В. В. Гаршина, К. С. Калабухов, В. А. Степанцов, С. В. Смотров. – URL: http://www.vestnik.vsu.ru/pdf/analiz/2017/03/2017-03-21.pdf (дата обращения: 28.03.2022)

15. Кашницкий, Ю. Открытый курс машинного обучения. Тема 4. Линейные модели классификации и регрессии [Электронный ресурс]: статья. – URL: https://habr.com/ru/company/ods/blog/323890/#2-logisticheskaya-regressiya (дата обращения: 02.04.2022)

16. Sarkar, DJ. Text Analytics with Python. A Practitioner's Guide to Natural Language Processing. [Текст]: учеб. пособие / DJ. Sarkar – NY: Apress Media, 2019. – 674 с.

17. Conor, Mc. Machine learning fundamentals (I): Cost functions and gradient descent [Электронный ресурс]: статья. – URL: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220 (дата обращения: 04.04.2022)

18. Жерон, О. Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow [Текст]: учеб. пособие / О. Жерон – СПб: Диалектика, 2020. – 1040 с.

19. Asiri, S. Machine Learning Classifiers [Электронный ресурс]: статья. – URL: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623 (дата обращения: 04.04.2022)

20. Brownlee, J. Parametric and Nonparametric Machine Learning Algorithms [Электронный ресурс]: статья. – URL: https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/ (дата обращения: 06.04.2022)

21. Норкин, Д. Учебник по машинному обучению от Школы анализа данных. Метрические методы. [Электронный ресурс]: статья. – URL: https://ml-handbook.ru/chapters/metric_based/intro (дата обращения: 10.04.2022)

22. Орешков, В. Классификация данных методом k-ближайших соседей [Электронный ресурс]: статья. – URL: https://loginom.ru/blog/knn (дата обращения: 11.04.2022)

23. Шахиди, А. Деревья решений: общие принципы [Электронный ресурс]: статья. – URL: https://loginom.ru/blog/decision-tree-p1 (дата обращения: 17.04.2022)

24. Sam T. Entropy: How Decision Trees Make Decisions [Электронный ресурс]: статья. – URL: https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8 (дата обращения: 17.04.2022)

25. Петухов, Д. Коэффициент Джини. Из экономики в машинное обучение. [Электронный ресурс]: статья. – URL: https://habr.com/ru/company/ods/blog/350440/ (дата обращения: 20.04.2022)

26. Елистратова, Е. Учебник по машинному обучению от Школы анализа данных. Bias-variance decomposition. [Электронный ресурс]: статья. – URL: https://ml-handbook.ru/chapters/ensembles/intro (дата обращения: 26.04.2022)

27. Елистратова, Е. Учебник по машинному обучению от Школы анализа данных. Ансамбли в машинном обучении. [Электронный ресурс]: статья / Е. Елистратова, П. Губко. – URL: https://ml-handbook.ru/chapters/ensembles/intro (дата обращения: 27.04.2022)

28. Kashnitsky, Y. Bagging and Random Forest [Электронный ресурс]: статья. – URL: https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-5-ensembles-of-algorithms-and-random-forest-8e05246cbba7 (дата обращения: 01.05.2022)

29. Соколов, Е. А. Бэггинг, случайные леса и разложение ошибки на смещение и разброс [Электронный ресурс]: статья. – URL: https://github.com/esokolov/ml-course-hse/blob/master/2020-fall/lecture-notes/lecture08-ensembles.pdf (дата обращения: 02.05.2022)

30. Шалев-Шварц, Ш. Идеи машинного обучения: от теории к алгоритмам [Текст]: учеб. пособие / Ш. Шалев-Шварц, Ш. Бен-Давид. – М.: ДМК Пресс, 2019. – 436 с.

31. Aliyev, V. Machine Learning Algorithms from Start to Finish in Python: Logistic Regression [Электронный ресурс]: статья. – URL: https://towardsdatascience.com/machine-learning-algorithms-from-start-to-finish-in-python-logistic-regression-5a62e3495318 (дата обращения: 05.05.2022)

32. Фонарев, А. Ю. Обзор алгоритмов бустинга [Электронный ресурс]: статья. – URL: http://www.machinelearning.ru/wiki/images/9/9a/fonarev.overview_of_boosting_methods.pdf (дата обращения: 05.05.2022)

33. Цыпленков, Л. Бустинг с помощью AdaBoost и Gradient Boosting [Электронный ресурс]: статья. – URL: https://lambda-it.ru/post/busting-s-pomoshchiu-adaboost-i-gradient-boosting (дата обращения: 07.05.2022)

34. Chen, T. XGBoost: A Scalable Tree Boosting System [Электронный ресурс]: статья / T. Chen, C. Guestrin. – URL: https://arxiv.org/pdf/1603.02754.pdf (дата обращения: 08.05.2022)

35. Шитиков, В. К. Бинарные деревья решений [Электронный ресурс]: статья / В. К. Шитиков, С. Э. Мастицкий. – URL: https://ranalytics.github.io/data-mining/052-Binary-Decision-Trees.html (дата обращения: 08.05.2022)

36. Morde, V. XGBoost Algorithm: Long May She Reign! [Электронный ресурс]: статья. – URL: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d (дата обращения: 10.05.2022)

37. Лутц, М. Python. Карманный справочник. [Текст]: учеб. пособие / М. Лутц – М.: И.Д. Вильямс, 2015. – 320. (дата обращения: 12.05.2022)

38. Mikolov, T. Efficient Estimation of Word Representations in Vector Space [Электронный ресурс]: статья / T. Mikolov, G. Corrado, J. Dean. – URL: https://arxiv.org/pdf/1301.3781.pdf (дата обращения: 16.05.2022)

39. Pennington, J. GloVe: Global Vectors for Word Representation [Электронный ресурс]: статья / J. Pennington, R. Socher, C.D. Manning URL: https://nlp.stanford.edu/pubs/glove.pdf (16.05.2022)

40. Fangxiaoyu, F. Language-agnostic BERT Sentence Embedding [Электронный ресурс]: статья / F. Fangxiaoyu, Y. Yinfei, D. Cer, N. Arivazhagan, W. Wang. – URL: https://arxiv.org/pdf/2007.01852.pdf (дата обращения: 18.05.2022)

41. Ohsaki, M. Confusion-matrix-based kernel logistic regression for imbalanced data classification // IEEE Transactions on Knowledge and Data Engineering. – 2017. – Т. 29. – №. 9. – С. 1806-1819. (дата обращения: 15.05.2022)

Приложение 1

```
def comment_preprocessing(s):
    # удаляем эмотиконы
    s = ''.join(char for char in s if char not in emoji.UNICODE_EMOJI)

    for e in emoticons:
        if e in s:
            s = s.replace(e, '')

    s = re.sub(r'e\.g\.', 'for example', s)
    s = re.sub(r'i\.e\.', 'that is', s)
    s = re.sub(r'[A-z]+\.[A-z]+\([\w\s\'\",]*\)', '[code]', s)  # строки типа: random.seed(43)
    s = re.sub(r'[A-z]+\([A-z]+\([\w*]*\)\)', '[code]', s)  # строки типа: int(input(**kwargs))
    s = re.sub(r'[A-z]+\(\)', '[code]', s)  # строки типа: rfind()
    s = re.sub(r'\"([\w\s!?.\',]+)"', '[code]', s)  # где только двойные кавычки
    s = re.sub(r'["\']([\w\s!?.\'\"]+)["\']', '[code]', s)  # где оба типа кавычки
    s = re.sub(r'["\'][\w\s\-!?.\\*~^_\',\[\]()]+["\']', '[code]', s)
    s = re.sub(r'@\w+', 'user_name', s)  # имена пользователей через @
    s = re.sub(r'\w+\.\w+\s', '[code]', s)  # RockBand.genre
    s = re.sub(r'[a-zA-Z]+[0-9]+', '[code]', s)  # f1
    s = re.sub(r'\w+\.\w+[\W]+', '[code]', s)  # text.strip("")
    s = re.sub(r'\(self, [\w\s, ]+\):', '[code]', s)  # (self, hyp, leg_1, leg_2):
    s = re.sub(r'__\w+__', '[code]', s)  # __class__
    s = re.sub(r'class \w+:', '[code]', s)  # class RightTriangle:
    s = re.sub(r'line [0-9]+', '[code]', s)  # line 17
    s = re.sub(r'\w+Error:', '[code]', s)  # NameError:
```

```
  reader = io.StringIO(s)
  gen = tokenize.generate_tokens(reader.readline)

  clean_tokens = []
  try:
      for t in gen:
          # print(t)
          if t.type == 2:
              clean_tokens.append('[number]')  # числа тоже заменяем
          elif t.type in [61, 4, 0, 6, 5] or t.string == ' ':  # новая строка \n, NEWLINE, ENDMARKER, COMMENT
              continue
          elif len(t.string) == 1 and t.string.lower() not in {'a', 'i', 't', 's'} and t.type not in {54, 59}:
              clean_tokens.append('[code]')
          elif t.type == 59 and t.string != "'":
              continue
          elif t.type == 3:
              clean_tokens.append('[code]')
          elif t.string != 'user_name' and re.fullmatch(r'\w+_\w+', t.string):
              clean_tokens.append('[code]')
          elif t.string in set(python_key_symbols):  # синтаксис питона считаются кодом
              clean_tokens.append('[code]')
          else:
              clean_tokens.append(t.string)
  except (tokenize.TokenError, IndentationError):
      pass
```

```
for i, _ in enumerate(clean_tokens):
    if len(clean_tokens) > 1:
        if i == 0:
            if clean_tokens[i + 1] == '[code]':
                clean_tokens[i] = '[code]'
        elif i == (len(clean_tokens) - 1):
            if clean_tokens[i - 1] == '[code]':
                clean_tokens[i] = '[code]'
        else:
            if clean_tokens[i - 1] == '[code]' and clean_tokens[i + 1] == '[code]':
                clean_tokens[i] = '[code]'
```

```
tokens_final = []
for i, t in enumerate(clean_tokens):
    if tokens_final:
        if t == '[code]' and tokens_final[-1] == '[code]':
            continue
        else:
            tokens_final.append(t.lower())
    else:
        tokens_final.append(t.lower())

# часто бывают комментарии типа: Failed test #4 - заменяем просто на [number]
s = re.sub(r'#\s*(\d)+', '[number]', ' '.join(tokens_final))
return s
```

Приложение 2

Векторизация	Алгоритм классификации			
	Логистическая регрессия			
	accuracy	precision	recall	F-score
TF-IDF	**0.54**	**0.69**	**0.62**	**0.65**
GloVe	0.51	0.68	0.59	0.63
LaBSE	0.58	0.72	0.57	0.61

Приложение 3

Векторизация	Алгоритм классификации			
	k-ближайших соседей			
	accuracy	precision	recall	F-score
TF-IDF	0.51	0.72	0.79	0.71
GloVe	0.53	0.70	0.77	0.72

LaBSE	**0.53**	**0.71**	**0.77**	**0.73**

Приложение 4

Векторизация	Алгоритм классификации			
	Случайный лес			
	accuracy	precision	recall	F-score
TF-IDF	0.5	0.63	0.79	0.7
GloVe	**0.5**	**0.69**	**0.79**	**0.7**
LaBSE	0.5	0.63	0.79	0.7

Приложение 5

Векторизация	Алгоритм классификации			
	XGBoost			
	accuracy	precision	recall	F-score
TF-IDF	0.52	0.72	0.78	0.72
GloVe	0.51	0.75	0.79	0.71
LaBSE	**0.53**	**0.72**	**0.78**	**0.72**

Приложение 6

```python
def comments_sorting(df):
    df = preprocessing(df)

    # n-gramms
    target_gramms = []
    with open('n_gramms.txt') as f:
        for line in f:
            target_gramms.append(line.rstrip().lower())

    target_gramms = set(line for line in target_gramms if line)

    all_texts = []
    for comment in df['text']:
        # Normalize tabs and remove newlines
        no_tabs = comment.replace('\t', ' ').replace('\n', '')
        # Normalize spaces to 1
        multi_spaces = re.sub(" +", " ", no_tabs)
        # Strip trailing and leading spaces
        no_spaces = multi_spaces.strip()
        all_texts.append(no_spaces)

    labels = []
    for comment in df['text']:
        labels.append(prediction(comment.lower(), target_gramms))

    res_n_gramms = df.copy()
    res_n_gramms['labels'] = labels
    df_fix = res_n_gramms[res_n_gramms['labels'] == True][['text', 'content_link']]
```

```python
    # XGBoost + LaBSE
    preprocessed_comments = [comment_preprocessing(c) for c in all_texts]
    labse_embeddings = [labse_embedding(c) for c in preprocessed_comments]

    xg_predictions = xg_labse.predict(torch.cat(labse_embeddings))

    res_xg = df.copy()
    res_xg['labels'] = xg_predictions
    df_fix_2 = res_xg[res_xg['labels'] == 1][['text', 'content_link']]

    return pd.concat([df_fix, df_fix_2], ignore_index=True, axis=0).drop_duplicates(subset=['text'])
```