Федеральное государственное образовательное бюджетное учреждение
высшего образования
«Финансовый университет при Правительстве Российской Федерации»
(Финансовый университет)

Факультет информационных технологий и анализа больших данных

Департамент анализа данных и машинного обучения

Выпускная квалификационная работа

на тему: «Классификация транзакционных данных по расширенному описанию»

Направление подготовки 01.03.02 «Прикладная математика и информатика»

Профиль «Анализ данных и принятие решений в экономике и финансах»

Выполнил студент группы
ПМ17-2
Бахматов Андрей Вячеславович

Руководитель к.э.н., доцент
Макрушин Сергей Вячеславович

ВКР соответствует предъявляемым требованиям

Руководитель Департамента
д.э.н., профессор
В.И. Соловьев
«__» _____ 20__ г.

Москва 2021

Содержание

Введение

Машинное обучение является одной из важнейших частей в сфере информационных технологий: ее суть состоит в построении моделей с адаптирующимися параметрами, которые меняются в зависимости от того, какие данные модель получает для обучения. Адаптируясь к этим данным, качественная модель машинного обучения способна обобщать их и правильно реагировать на новые данные. Таким образом, вместо программирования определенного набора команд для выполнении задачи, эту задачу можно решить при помощи большого количества данных и алгоритма, который учится эту задачу выполнять.

Машинное обучение делят на три основных раздела:

1. обучение с учителем: на обучающей выборке имеется таргет - то, что модель должна предсказать (классификация - множество таргета категориально и заранее известно, пример - определение спама; регрессия - множество таргета в общем случае это подмножество $\mathbb{R}^n$, пример - прогноз цены ценной бумаги);

2. обучение без учителя: отсутствует таргер (кластеризация - требуется разделить данные на заранее неизвестное количество кластеров (групп); нахождение аномалий; уменьшение размерности данных);

3. обучение с подкреплением: обучение поведения интеллектуального агента на основе его взаимодействия с некоторой средой (пример - чатботы).

Огромный след машинное обучение оставило в NLP (Natural Language Processing), области лингвистики и искусственного интеллекта, занимающейся проблемами компьютерной обработки и анализа естественного языка. Благодаря синтезу этих областей появились сегодняшние чатботы, голосовые помощники.

В последнее время происходит бум в области информационных технологий: наблюдается использование методов машинного обучения в самых разнообразных сферах жизни: в интернет сервисах (чат-боты, поисковые системы, рекомендательные системы), в городских инфраструктурах (видеонаблюдение, штрафование автомобилистов), в финансовой сфере (определение/предотвращение мошенничества). Методы машинного обучения используются как для широкого потребителя, так и для решения специфических запросов, например, прогнозирования дефолтов компаний.

Данная работа является результатом одного из этапов разработки модели прогнозирования дефолтов, конкретно: создания классификатора транзакционных данных на основе их текстового описания, а также дополнительных признаков, опираясь на методы машинного обучения/NLP. В итоге было разработано 6 основных моделей: сверточная нейронная сеть, метод опорных векторов, BERT, случайный лес, RoBERTa, rule-based модель. Далее на их основе были созданы ансамбли: ансамбли одиночных моделей,

обученных на разделах обучающей выборки; ансамбли комбинации моделей с разными способами аггрегации предсказаний (голосования, использования precision каждой модели на валидационной выборке; stacking при помощи разных мета-моделей).

Использовавшийся для обучения и оценки моделей датасет состоит из $\sim$20000 транзакций.

Разработка осуществлялась на языке программирования Python 3.7[12]. Модели строились в основном на фреймворках sklearn[13], PyTorch[14], TensorFlow[15].

1 Теоретическая часть

1.1 Задачи обработки естественного языка и ее проблематика

Обработка естественного языка (NLP)[1] является довольно обширной областью, список ее основных задач:

1. классификация: определение класса текста на основе заранее известного списка классов (например, фильтр спама);

2. кластеризация: деление документов на кластеры (группы, заранее неизвестные), используется в рекомендательных системах, агрегации новостей;

3. разметка частей речи (part-of-speech/POS tagging) - процесс отнесения слов в тексте к определенным частям речи, учитывая само слово и его контекст. Используется в анализе тональности тексте (sentiment analysis), распознавании именных сущностей (named-entity recognition), а также в разработке поведения искусственного интеллекта;

4. синтаксический анализ естественных языков;

5. разметка семантических ролей - схожая с POS tagging задача, но вместо частей речи размечаются семантические роли (агент, цель, результат);

6. анализ тональности текста (sentiment analysis) - анализ эмоционального окраса текста, используется для анализа потребительских настроений относительного брэнда/продукта;

7. разрешение лексической многозначности - определение значения слов в тексте;

8. распознавание именных сущностей (named-entity recognition) - нахождение имен, организаций, мест и т.д. в тексте;

9. извлечение информации - извлечение структурированных данных из текста;

10. тематическое моделирование - определение тематики документов, используется для агрегации текстовых данных (например, новостных статей);

11. обобщение текста - преобразование текста, при котором остается лишь его краткое изложение;

12. исправление опечаток;

13. генерация естественного языка;
14. распознавание голоса;
15. синтез речи;
16. разработка вопросно-ответных систем;
17. машинный перевод;
18. разработка поисковых систем.

Для решения перечисленных задач могут быть использованы следующие процедуры:

1. токенизация - разбиение текста на составляющие части (обычно на слова, но иногда разбиение происходит по предложениям, слогам), используется для решения большинства задач в NLP. В большинстве языков решается относительно тривиально, но представляет большую проблему в языках, где слова не разделены пробелами (тайский, китайский языки). Обычно знаки препинания расцениваются как разделители между токенами, но в некоторых случаях имеет смысл относить и их к токенам (в социальных сетях и на форумах, где используются смайлики, например);
2. лемматизация - приведение слов в нормальную форму (имеешь -> имеет, сделала -> сделать);
3. стеммизация - нахождение у слов стеммы, то есть удаление окончания и суффиксов (имеешь -> име, сделала -> сдела);
4. очистка текста от незначащих слов (стоп-слов, не несущих смысловую нагрузки: такие слова как ‘а‘, ‘в‘, ‘или‘, ‘затем‘, ‘что‘, ‘который‘, ‘над‘, ‘под‘), от мусора (актуально при сборе информации из интернета, где данные веб-страниц хранятся в HTML)
5. представление текста - преобразование текста в иную форму, с которой легче работать. Популярные способы: ембеддинги (векторное представление), мешок слов (матрицы встречаемости слов в предложении), енкодинг метками (уникальным числами), TF-IDF (улучшенный мешок слов).
6. анализ схожести - создание метрик для сравнения слов, предложений, документов.

Работа в области NLP несет множество вызовов, связанных с проблематикой естественных языков:

1. контекст - в зависимости от него одни и те же слова/фразы имеют разное значение;

2. наличие иронии/сарказма;

3. различные неоднозначности - слов (одинаковая форма для разных частей речи слова), семантики (одно и то же предложение может иметь два разных смысла). В некоторых случаях даже для человека сложно определить, какой имеется ввиду смысл среди нескольких вариантов;

4. ошибки в тексте - опечатки, неправильное использование слов;

5. неформальные слова/слэнг;

6. научный текст/текст специфической области;

7. динамика значений слов - некоторые слова со временем меняют значение/обретают новые.

1.2 Модели классификации текста

В данной секции будут описаны модели классификации транзакций, а также методы, использующиеся при работе с ними.

1.2.1 Сверточная нейронная сеть

1.2.1.1 Искусственные нейронные сети

Искусственная нейронная сеть прямого распространения (или просто нейронная сеть) - это математическая модель, представляющая собой нелинейную функцию. Нейронные сети в общем случае характеризуются количеством слоев, количеством нейронов в каждом слое, функциями активации на каждом нейроне и весами на связях между нейронами. Входной слой: состоит из нейронов, хранящих и передающих информацию входных данных; выходной слой: имеет нейроны, получающие выходную информацию (ответ модели); скрытые слои: слои между входным и выходным слоем, передают вспомогательные признаки.

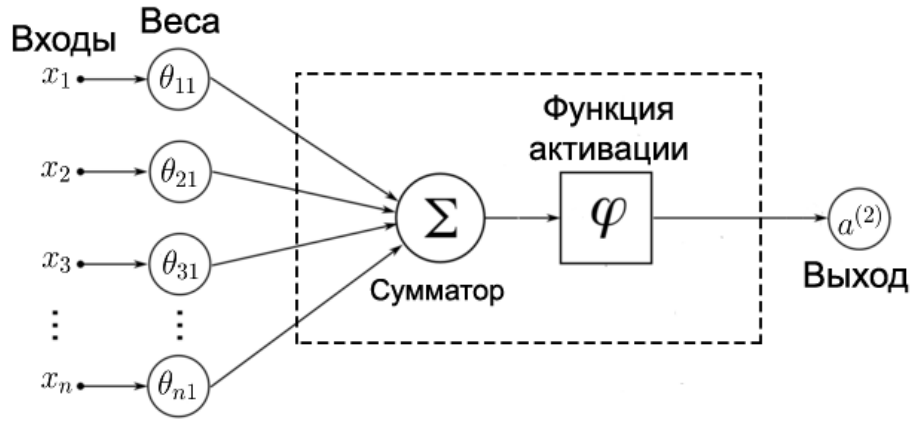

Рис. 1: Простая нейронная сеть прямого распространения (перцептрон)

На рисунке 1 показана нейронная сеть, имеющая входной слой из n нейронов (x_i), передающих информацию через связи с весами θ_{i1} (i - номер нейрона из первого слоя, 1 - номер нейрона из второго слоя) в выходной слой из 1 нейрона, который имеет функцию активации φ. На выходе получаем функцию $h_\theta(x)$:

$$h_\theta(x) = a^{(2)} = \varphi\left(\sum_{i=1}^{n} \theta_{i1} x_i\right),$$

где $a^{(2)}$ называют активацией нейрона на втором слое.

В данной модели парамерами являются веса $\theta = (\theta_{11}, \theta_{21}, ..., \theta_{n1})$: для их нахождения используется минимизация ошибки $J(\theta)$ (задающаяся, например, как

разница квадратов между значением $h_\theta(x)$ и целевым значением y). Базовый способ минимизации - градиентный спуск, при котором веса меняются по формуле:

$$\theta_{i1} = \theta_{i1} - \alpha \frac{\delta J(\theta)}{\delta \theta_{i1}},$$

где параметр α называют скоростью обучения.

В общем случае для всех весов будут определен набор матриц $\Theta = \{\Theta^{(j)}\}$, которые определяют веса между j-ым и j+1-ым слоями. Для прошлой модели этот набор будет иметь вид: $\Theta = \{\Theta^{(1)} = \theta\}$, если w - вектор-столбец. В качестве еще одного примера возьмем нейронную сеть с входным слоем из трех нейронов, одним скрытым слоем из 3 нейронов и выходным слоем из одного нейрона. Тогда выход модели $h_\Theta(x)$ будет вычисляться следующим образом:

$$a_1^{(2)} = \varphi_1 \left(\sum_{i=0}^{3} \Theta_{1i}^{(1)} x_i \right),$$

$$a_2^{(2)} = \varphi_1 \left(\sum_{i=0}^{3} \Theta_{2i}^{(1)} x_i \right),$$

$$a_3^{(2)} = \varphi_1 \left(\sum_{i=0}^{3} \Theta_{3i}^{(1)} x_i \right),$$

$$h_\Theta(x) = a_1^{(3)} = \varphi_2 \left(\sum_{i=0}^{3} \Theta_{1i}^{(2)} a_i^{(2)} \right),$$

где $a_i^{(j)}$ - активации j-ого слоя. x_0 и $a_0^{(j)}$ - свободные коэффициенты.

Новый вид градиентного спуска (обратное распространения ошибки):

$$\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha \frac{\delta J(\Theta)}{\delta \Theta_{ij}^{(l)}}.$$

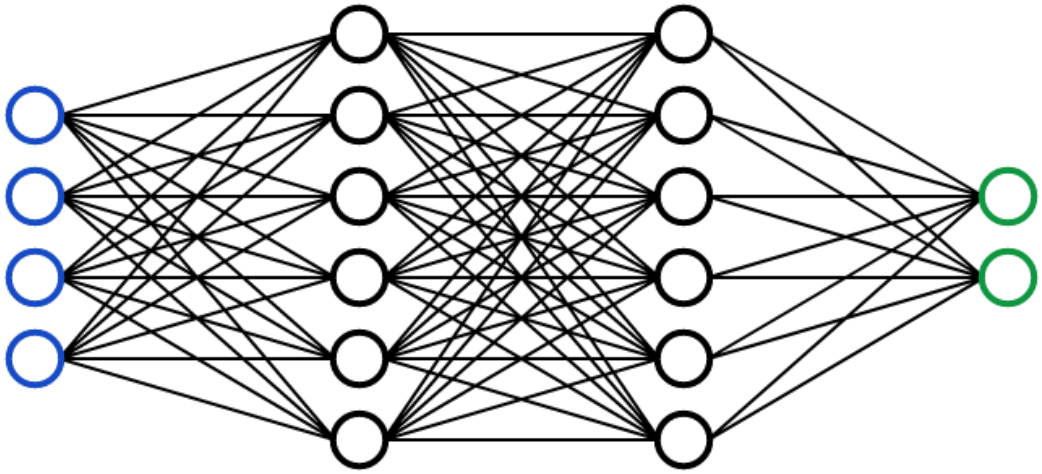

Рис. 2: Пример нейронной сети с несколькими скрытыми слоями

1.2.1.2 Архитектура сверточных слоев

Отличие сверточных нейронных сетей от обозначенной выше модели заключается в наличии сверточных слоев[2] - они изначально были разработаны для того работы нейронных сетей с фотографиями. Принцип их работы - умножение ядра слоя (матрицы-параметра) на фрагменты входного слоя, подающихся в виде скользящего окна. Таким образом формируются скрытые признаки, которые могут быть полезны при классификации. Из-за указанной методики эти признаки способны учитывать контекст близлежащих элементов (в нашем случае, слов).

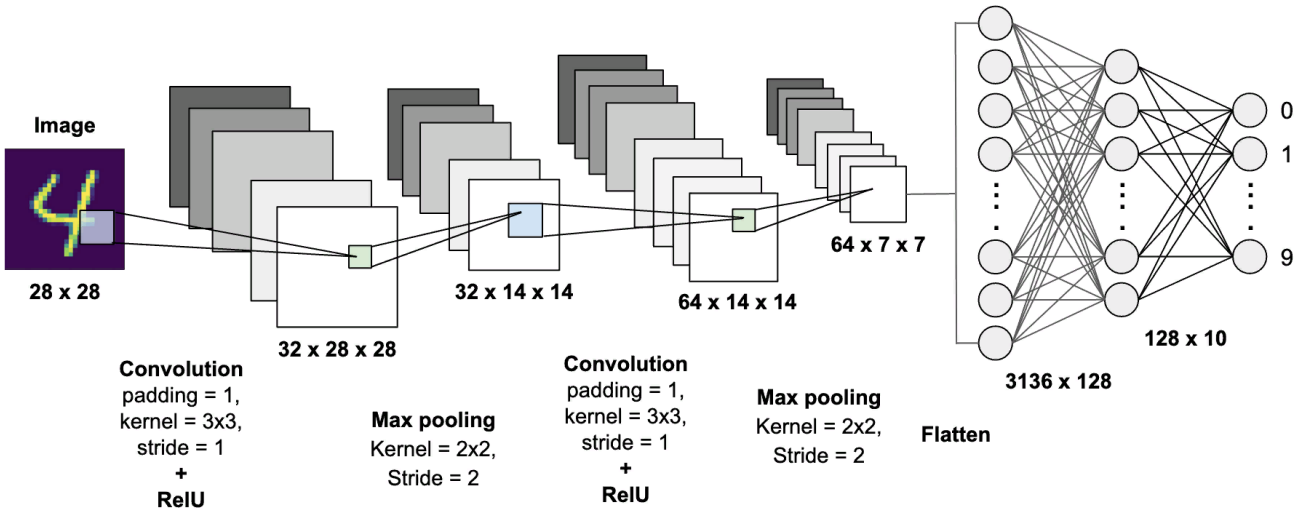

Рис. 3: Пример работы сверточной нейронной сети с изображением

Для снижения размерности полученных признаков используется пулинг: агрегация данных с помощью усреднения соседних элементов матрицы/выбора максимального значения из них. Несмотря на то, что сверточные нейронные сети были разработаны для классификации фотографий, они также хорошо работают с текстом, если его преобразововать в двухмерный вид (последовательность векторов, в контексте слов называемых ембеддингами).

1.2.1.3 Ембеддинги

Для того, чтобы сделать представление слов, при котором это слово можно будет математически сравнить с другими словами, можно представить его в виде ембеддинга - n-мерного вектора. Под сравнением имеется ввиду измерение расстояния между векторами - чем ближе ембеддинги двух слов, тем более схожи их значения. Качественные ембеддинги позволяют определять разные семантические и синтаксические отношения между словами при помощи простых алгебраических операций:

- мужчина-женщина (vec(мужчина) - vec(женщина) = vec(король) - vec(королева)),
- столица-страна (vec(Афины) - vec(Греция) = vec(Осло) - vec(Норвегия)),
- валюта-страна (vec(рубль) - vec(Россия) = vec(доллар) - vec(США)),
- название химического элемента (vec(медь) - vec(Cu) = vec(золото) - vec(Au)),
- имя-фамилия (vec(Путин) - vec(Владимир) = vec(Барак) - vec(Обама)),
- компания-продукт (vec(Microsoft) - vec(Windows) = vec(Apple) - vec(MacOS)),
- страна-национальность (vec(Россия) - vec(россиянин) = vec(Швеция) - vec(швед)),
- страна-еда (vec(Италия) - vec(пицца) = vec(Япония) - vec(суши)),
- прилагательное-наречие (vec(хороший) - vec(хорошо) = vec(веселый) - vec(весело)),
- антонимы (vec(возможно) - vec(невозможно) = vec(веселый) - vec(грустный)),
- формы сравнения (vec(хороший) - vec(лучше) = vec(веселый) - vec(веселее)),
- формы глагола - (vec(думать) - vec(думает) = vec(читать) - vec(читает)),

- разных частей речи (vec(думать) - vec(думающий) = vec(читать) - vec(читающий)),
- единственное-множественное число (vec(кот) - vec(коты) = vec(мешок) - vec(мешки)).

Ембеддинги, ставшие стандартом в машинном обучением - word2vec. В рамках статьи[3], описывающей word2vec ембеддинги, было описано два способа их обучения: непрерывный мешок слов и непрерывная скип-грамма. Непрерывный мешок слов учится, предсказывая текущее слово, основываясь на контексте соседних слов, а непрерывная скип-грамма наоборот, предсказывает соседние слова при заданном текущем слове.

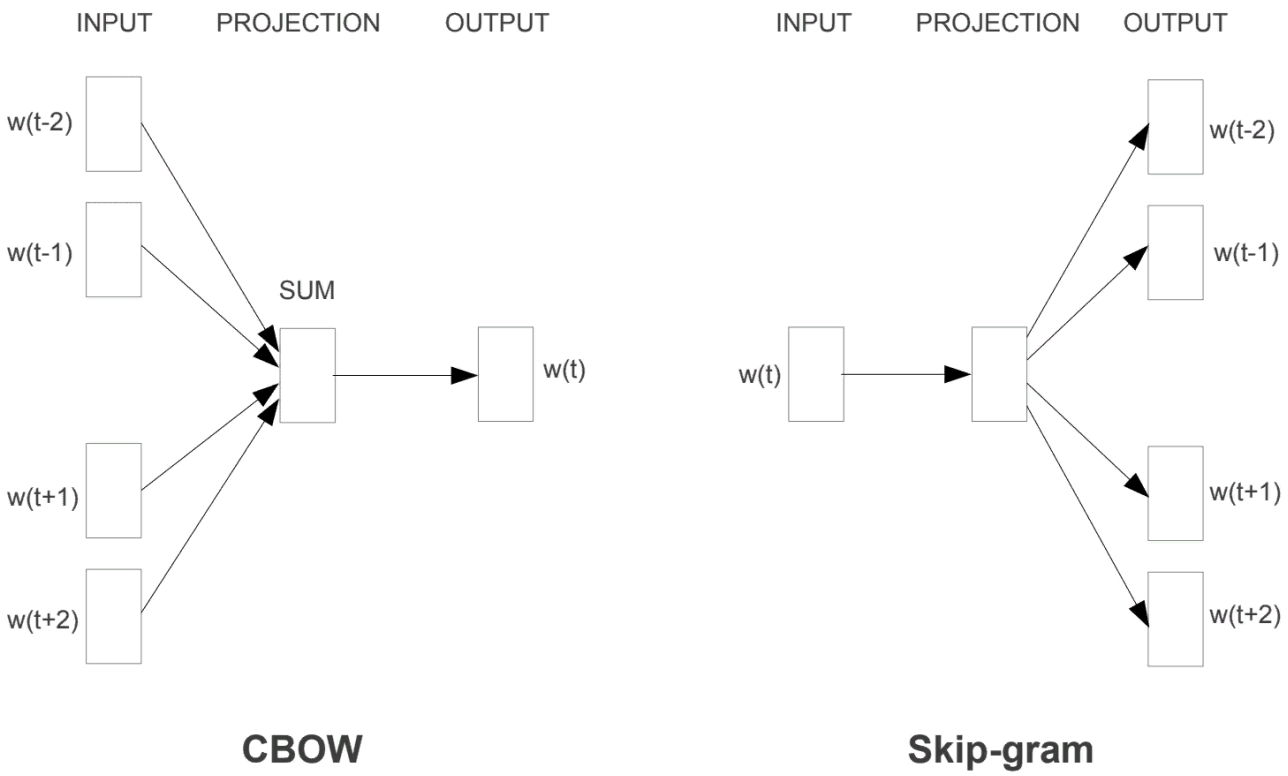

Рис. 4: Методы обучения word2vec ембеддингов

Развитием word2vec стал GloVe[4], сочетающий контекст-ориентированный word2vec с глобальными статистическими методами: вместо локального контекста ближайших слов используется глобальный, основанный на общей встречаемости слов друг с другом в корпусе.

При работе был протестирован один из вариантов GloVe - NaVec из проекта Natasha[6]: GloVe, обученный на большом количестве русскоязычных корпусов и квантизованный. Квантизация - сжатие размера ембеддингов. Оно достигается при помощи кодирования интервалов: например, интервал [-0.86,

-0.79) кодируется как 0, и если число в ембеддинге попадает в этот интервал, оно обозначается как 0. Таким образом, непрерывное пространство, на котором определены ембеддинги, становится дискретным. Далее происходит замена одного кода на три числа и кластеризация k-means, которая уменьшает размер еще больше. В итоге получаем просевшие на несколько процентов в качестве (на внутренних тестах), но занимающие меньшие в 12 раз ембеддинги. При этом за счет большого объема обучающего корпуса, они все равно в среднем сравнимы или лучше русских предобученных ембеддингов с RusVectores, являющихся по сути единственной альтернативой в области предобученных word2vec/GloVe ембеддингов на русском языке.

Одним из недостатков вышеупомянутых моделей является невозможность векторного представления слов, не встречавшихся при обучении векторов (out-of-vocabulary). Эту проблему решают ембеддинги fastText[5]. Используется непрерывная скип-грамм модель из word2vec, но кроме слов строится векторное представление нграмм букв в слове (для слова where и n=3 будет обучаться ембеддинг самого слова и следующих нграмм: wh, whe, her, ere, re; при этом нграмма "her" и слово "her" являются разными сущностями). Для представления out-of-vocabulary слов будет использоваться сумма ембедингов нграмм, из которых слово состоит.

1.2.1.4 Архитектура итоговой модели

Описав все релевантные термины, можно описать процедуру подачи данных модели и ее архитектуру:

1. предобработка/очистка текста;
2. преобразование текста в последовательность ембеддингов (при этом на ембеддинги во время обучения распространяется ошибка и они изменяются);
3. последовательность ембеддингов подается в сверточный слой, через дропаут (случайное обнуление нескольких входных нейронов для избежания переобучения) и пулинг, после чего попадает на выходной слой из нужного количества нейронов (равного количеству классов).

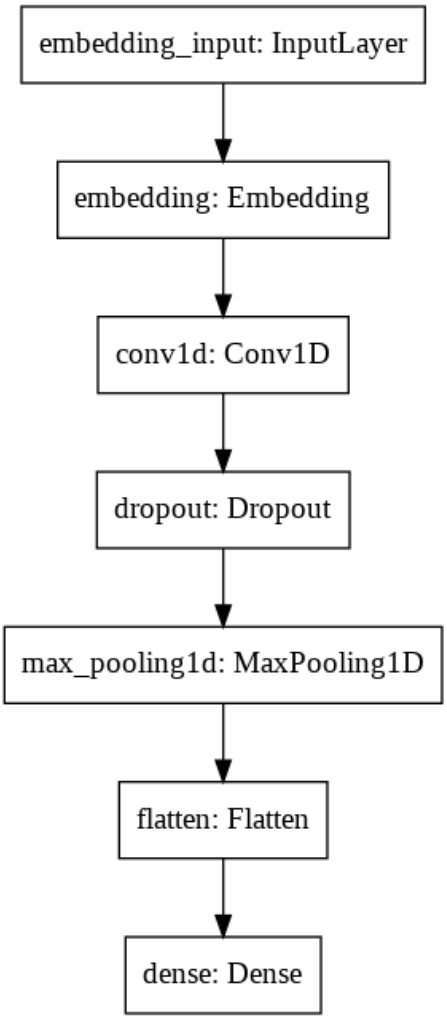

Рис. 5: Архитектура сверточной нейронной сети

1.2.2 Метод опорных векторов

Метод опорных векторов (Support Vector Machine) - модель, при классификации цель которой - определить гиперплоскость (границу решения), корректно разбивающее пространство входных признаков по принадлежности таргету (аналогично логистической регрессии).

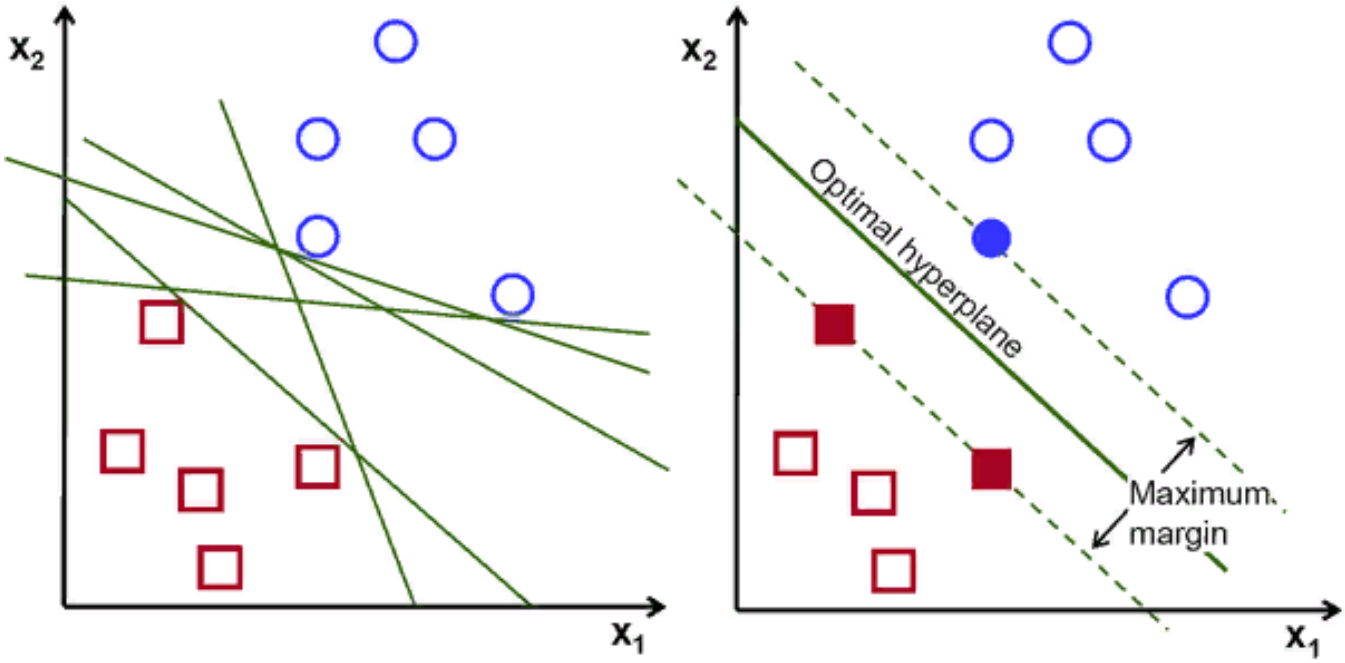

Рис. 6: Пример возможных разбивающих гиперплоскостей для задачи с двумя признаками x1 и x2: случайно выбранные слева и найденный при помощи SVM справа

Гиперплоскость задается уровнением: $\langle w, v\rangle + b = 0, w \in \mathbb{R}^n, v \in \mathbb{R}^n, b \in \mathbb{R}^n$, где w и b - параметры гиперплоскости, n - размерность взодных данных (принадлежность к классу определяется знаком выражения $\langle w, v\rangle + b$, то есть изначальная модель работает только с бинарной классификацией). Задача оптимизации, позволяющая найти w и b - максимизировать расстояние гиперплоскти от опорных векторов (самых близких к гиперплоскости представителей классов). Если расстояние гиперплоскости до вектора v определить как $d_H(v) = \frac{|\langle w,v\rangle + b|}{||w||_2}$, то формально задачу можно описать так:

$$w^* = arg_w max[min_{i=1,2...,m} d_H(v_i)],$$

где $v_i, i = 1, 2..., m$ - элементы обучающей выборки, $min_{i=1,2...,m} d_H(v_i)$ - расстояние до опорного вектора (минимальное расстояние до вектора из выборки).

Раскроем выражение:

$$w^* = arg_w max[min_{i=1,2...,m} d_H(v_i)] = arg_w max[min_{i=1,2...,m} \frac{|\langle w, v\rangle + b|}{||w||_2}] =$$

$$arg_w max[\frac{1}{||w||_2} min_{i=1,2...,m} |\langle w, v\rangle + b|].$$

Умножение w и b на константу не изменит результата, но при этом можно подобрать такую константу, что $|\langle cw, v\rangle + cb| = 1$. Если принять теперь cw за w, cb за b, то выражение приобретет вид:

$$\begin{cases} arg_w max[\frac{1}{||w||_2}], \\ c_i(\langle w, v_i\rangle + b) \geq 1,\ i = 1, 2, ..., m, \end{cases}$$

c_i - знак класса, к которому относится v_i. Второе выражение отвечает за упомянутое масштабирование: для опорного вектора v будет выполняться: $c(\langle w, v\rangle + b) = 1$.

Дальше задача оптимизации реализуется методом Лагранжа.

Таким образом мы находим линейное разделение пространства. Если в задаче данные линейно неразделимы, можно применить преобразование к входным данным при помощи функции ϕ, чтобы разделяющая гиперплоскость была нелинейной:

$$\begin{cases} arg_w max[\frac{1}{||w||_2}], \\ c_i(\langle w, \phi(v_i)\rangle + b) \geq 1,\ i = 1, 2, ..., m. \end{cases}$$

TF-IDF Для подачи методу опорных векторов текста он будет представлен в виде вектора с помощью метода TF-IDF: текст представляется как вектор-индикатор, показывающий для каждого слова из датасета, входит ли оно в этот текст. Далее каждое значение умножается TF-IDF слова - оно равно частоте появления слова в данном тексте (Term Frequency), деленной на величину, обратной появлению слова во всем датасете (Inverse Document Frequency). Таким образом получается сделать оценку 'полезности' слова в тексте - если оно встречается почти в каждом тексте (например, стоп-слова), то его значимость будет мала, и соответственно мал коэффициент TF-IDF.

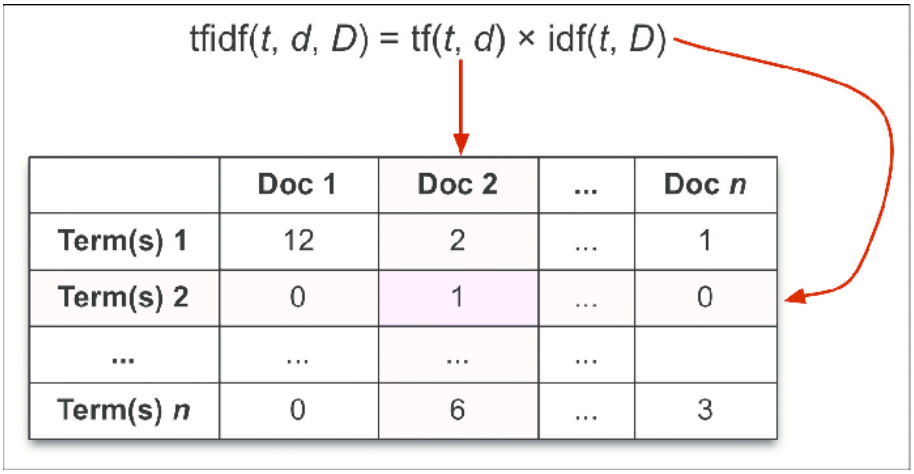

Рис. 7: text

1.2.3 BERT

BERT (Bidirectional Encoder Representations from Transformers)[7] является одним из способов построения векторного представления текста. BERT сначала обучается на нескольких NLP задачах, чтобы построить изначальное представление токенов (пре-обучение), а после во время обучения на реальной задаче корректирует параметры модели.

Пре-обучение состоит из следующих этапов:

1. наложение масок: в датасете случайным образом выбираются слова (вероятность выбора 0.15) и применяется одна из следующих операций: замена слова на служебный токен [MASK] (скрытие слова); замена слова на случайное; бездействие (вероятности этих событий соответственно равны 0.8, 0.1, 0.1);

2. модели на вход подается набор токенов с двумя последовательностями, разделенными токеном [SEP]: вторая последовательность является либо следующей в документе после первой, либо взятой из другого документа (вероятности обоих исходов равны);

3. модель учится предсказывать скрытые слова на основе контекста;

4. модель учится предсказывать следующее предложение (она должна ответить, следует ли вторая последовательность после первой на самом деле или является случайно взятой).

Токенизация текста осуществляется по методу WordPiece[8], который сначала рассматривает токен как один символ, и итеративно добавляет самые популярные в корпусе сочетания символов как возможные токены. В итоге токенизация разбивает текст в разрезе частей слов.

По архитектуре BERT является двухсторонним трансформером. Трансформер[9] - это модель, которая сначала преобразовывает входную последовательность в многомерный вектор (енкодер), а после преобразует вектор в новую последовательность (декодер). Для работы с последовательностями как входными данными часто используют рекуррентные нейронные сети (GRU, LSTM). В трансформерах отсутствует рекуррентность - для захвата контекста используется внимание: параметры, показывающие, насколько важен каждый элемент последовательности.

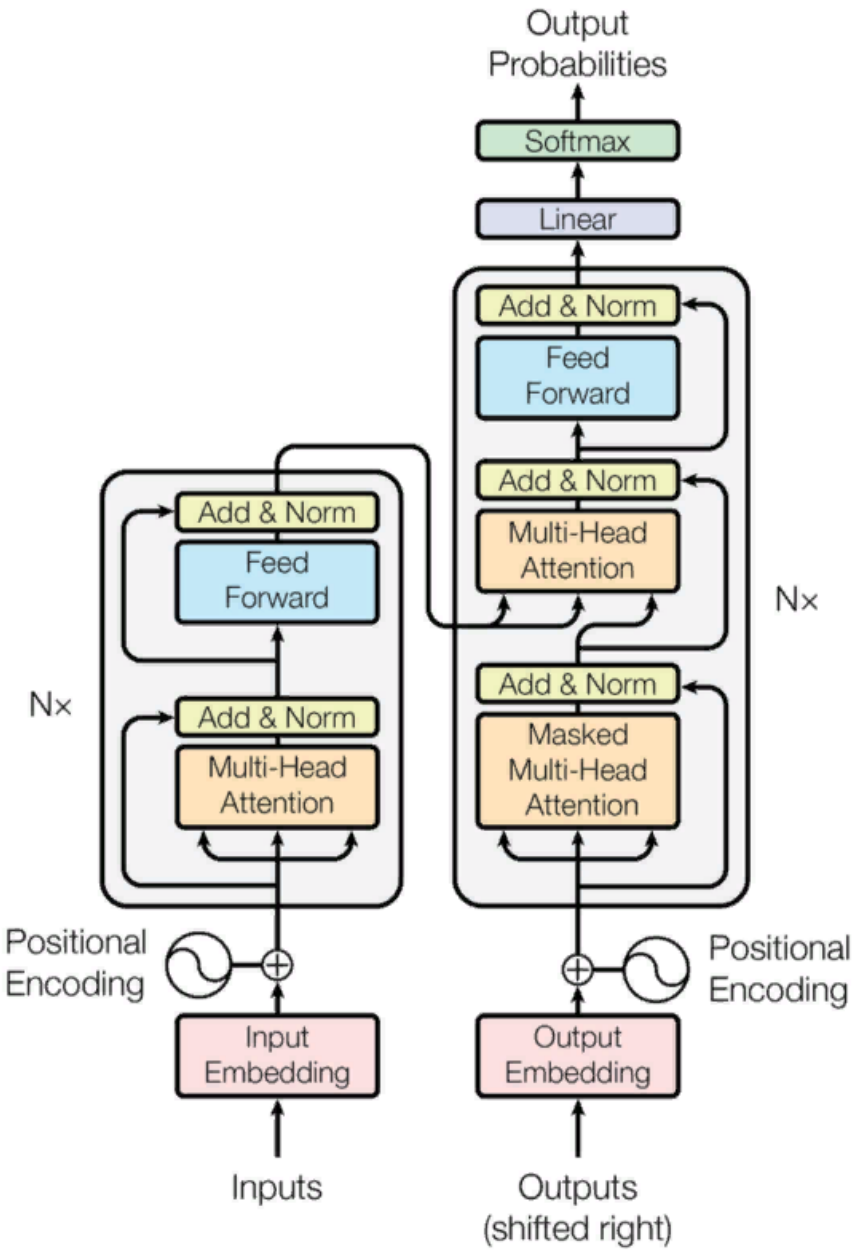

Рис. 8: Схематичное представление архитектуры трансформера

1.2.4 Случайный лес

Случайный лес[11] - это ансамблевый алгоритм, создающий множество деревьев решений.

Дерево решений - модель, принципом которой является разделение входных данных при помощи ветвления по их признакам.

Рис. 9: Дерево решений на примере классификации страхования

Дерево строится с помощью разделения датасета на подмножества на основании отдельных признаков (возраст автовладельца, место эксплуатации, стаж, тип автомобиля) неопределенное количество раз до того момента, как будет выполнен критерий остановки.

Алгоритм случайного леса:

1. m раз берется случайное подмножество датасета;

2. каждый раз на полученной выборке обучается дерево решений, при этом обучение будет происходит на случайном подмножестве признаков (для уменьшения корреляции между разными деревьями: если один признак сильно коррелирует с таргетом, то большая часть деревьев будет концентрироваться на одном и том же признаке);

3. в качестве предсказания случайного леса берется среднее предсказаний всех обученных деревьев решений или делается голосование.

1.2.5 RoBERTa

RoBERTa является результатом изменения этапа пре-обучения в BERT для улучшения качества.

Отличия пре-обучения в RoBERTa:

1. больше эпох;
2. больший размер батчей;
3. больший датасет (160 GB вместо 16);
4. динамические маски (каждую эпоху происходит повторое наложение масок);
5. отказ от задачи предсказания следующего предложения (NSP) и впоследствии увеличение длины входной последовательности: изначально считалось, что отказ от NSP ухудшает качество модели (основываясь на тестовых датасетах), однако при повторной реализации во время разработки RoBERTa было обнаружено обратное - отказ от NSP при увеличении входной последовательности в два раза (так как задача NSP требовала на один вход две последовательности) либо не меняет, либо немного улучшает метрики на тестовых датасетах, так как модель имеет возможность захватывать более долгий контекст в тексте;
6. ембеддинги байтов вместо юникодовых символов/последовательностей.

2 Построение классификационных моделей

2.1 Постановка задачи, датасет, начало построения моделей

Задачей данной работы, как было сказано выше, является классификация транзакционных данных. Перед предобработкой в датасете содержалось 19979 транзакции, после - 19421.

Транзакции имеют следующие признаки: documentid, to_acc, name_to, from_acc, name_from, purp, sum, payer_tax_num, payee_tax_num, payer_bank_bic_cd, payee_bank_bic_cd, inn_same, bic_same, class.

Из интересующих нас признаков имеются: purp - текстовое описание транзакции (например "За электроснабжение (окончательный расчет, за январь), по договору xxxxxxxx от 01.01.2018. С/ф N 1111111111111/12/00000 от 31.01.2019Сумма 98898-65В т.ч. НДС 20% 11111-11"); sum - сумма транзакции в рублях; name_to и name_from - бухгалтерские счета обоих сторон, участвующих в транзакции (to_acc и from_acc имеют тождественный смысл); class - класс транзакции (таргет классификации).

Изначально в датасете содержались следующие классы: аренда, валюта, вексель, взыскание с фл, возврат, вынос на просрочку, выплаты соцхарактера, выручка, гашение просрочки, депозит, дивиденды, докапитализация, займ, зарплата, инкассирование, комиссии банку, кредит, лизинг, налоги ндс, налоги ндфл, налоги прибыль, налоги прочие, обеспечение, оплата фл, перевод на фл, пожертвования и благотворительность, пополнение счета, проценты по кредиту, прочее, страховая премия, страховое возмещение, суды, штрафы государство, штрафы прочие, эквайринг.

В процессе предобработки следующие классы были собраны в отрицательный класс __прочее (из-за количества представителей в датасете меньше 20 элементов для каждого из этих классов): вексель, вынос на просрочку, гашение просрочки, депозит, докапитализация, лизинг, налоги ндс, пожертвования и благотворительность, штрафы государство.

Также в датасете имелся служебный класс _неоднозначность, который появился при разметке (изначально разметка осуществлялась так, что каждую транзакцию размечало 3 человека, и если все трое не сошлись в ответе, то транзакцию определяли этим классом). Перед подачей моделям транзакции с этим классом были убраны.

Для нахождения наилучшей классификационной модели будут обучены отдельные модели, а также на их основе созданы ансамбли моделей.

Начнем работу в Google Colaboratory: импортируем требуемые модули (в работе будут использоваться модули, специально созданные для наших задачи: preprocessing (различные функции для предобработки текста), ensemble (набор классов для создания ансамблей из одной конфигурации модели, например, из метода опорных векторов), embeddings (набор функций для

загрузки/создания ембеддингов word2vec, NaVec и fastText, использующихся в модели сверточной нейронной сети), rules (классификация, основанная на правилах), conv (модуль для инициализации и обучения сверточной нейронной сети, далее идут аналогичные модули для других моделей), bert, random_forest, roberta).

```
#текущая папка
base = 'drive/My Drive/GPB'

!pip install transformers
import numpy as np
import pandas as pd
import scipy.stats as st
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, precision_score, recall_score
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')
import importlib
import os
import pickle
import re

import seaborn as sns
from matplotlib import style
style.use('ggplot')
%config InlineBackend.figure_format = 'retina'

import sys
in_colab = 'google.colab' in sys.modules

if in_colab:
    sys.path.append(base)

!sudo pip3 install rutimeparser
import preprocessing
import ensemble

#чтобы не менять пути к файлам в зависимости от того, в колабе ноутбук или нет
```

```
def get_drive_path(path):
    in_colab = 'google.colab' in sys.modules
    return '/content/'+base+'/'+re.sub(r'\\','/',path) if in_colab else path
```

В нашем датасете классы енкодированы числами - загрузим словари для перехода от енкодингов к категориям и наоборот.

```
#кодировки классов (каждому классу соответсвует число)
code_names_path = get_drive_path('data/Классы_кодировка.csv')
code_to_class, class_to_code = preprocessing.get_code_names(code_names_path)
```

Загрузим датасет, убрав транзакции с классом _неоднозначность.

```
#загрузка датасета: в итоге датафреймы train_data и test_data должны иметь столбцы
↪  purp (текстовое описание транзакции), class, name_to, name_from

def load_data(path):
    data = pd.read_csv(path)
    data.rename({'Ответ 2 из 3:':target_name}, axis=1, inplace=True)
    data = data[data[target_name] != class_to_code['_неоднозначность']]
    return data

target_name = 'class'

train_path = get_drive_path('data\\train_data_19900_win.txt')
test_path = get_drive_path('data\\test_data_19900_win.txt')

train_data = load_data(train_path)
test_data = load_data(test_path)
all_data = pd.concat((train_data, test_data))
```

2.2 Отдельные модели

2.2.1 Сверточная нейронная сеть

2.2.1.1 Лучшая конфигурация

Инициализируем модель и входные параметры:

- загрузим все 19979 штук текстовых описаний (будем называть их корпусом) транзакций для обучения ембеддингов;

- cleaned_corpus_version и cleaned_corpus_path - какую версию очищенного корпуса загрузить, если она ранее создавалась (подробности в разделе "Оптимизации модели");
- path_to_shortenings_file - путь к файлу с имеющимися в корпусе сокращениям и их расшифровкам (комис - комиссия, зачисл - зачисление, ч/з - через), которые будут применяться на корпусе;
- fix_spelling - исправлять ли опечатки в корпусе при помощи API Яндекс.Спеллер;
- очищаем корпус при помощи функции из модуля embeddings;
- инициализируем объект класса conv.ConvPreprocessor, использующийся для удобной предобработки данных перед подачей их нейросети (use_other_features - использовать ли данных из других столбцов для классификации);
- делаем предобработку тренировочной выборки, создаем валидационную выборку;
- vocab_size - количество используемых слов, использующихся при енкодинге (переходе от предложений к последовательностям чисел), padding_len - длина самого длинного предложения в тренировочной выборке после енкодинга, output_number - размерность выхода нейросети, number_of_dimensions - размерность ембеддингов;
- обучаем или загружаем уже обученные ембеддинги fasttext при помощи embeddings.load_fast_text_trained.

```
import conv
import embeddings as eb
full_corpus = pd.read_excel(get_drive_path('data\\full_text_corpus.xlsx'))['purp']
cleaned_corpus_version = 5 #cleaned_corpus_versions.txt, зависящие переменные -
↪  path_to_shortenings_file, fix_spelling
cleaned_corpus_path =
↪  get_drive_path(f'data\\cleaned_corpus_{cleaned_corpus_version}.csv')

path_to_shortenings_file = get_drive_path('data\\Сокращения.xlsx')
fix_spelling = False
stop_words = stopwords.words('russian')

cleaned_corpus = eb.get_cleaned_corpus(cleaned_corpus_path,
```

```
                                        full_corpus,
                                        stop_words,

                                        ↪  path_to_shortenings_file=path_to_shortenings_fi
                                        fix_spelling=fix_spelling)

conv_processor = conv.ConvPreprocessor(use_other_features=False,

                                       ↪  path_to_shortenings_file=path_to_shortenings_file,
                                       fix_spelling=fix_spelling)

X_train, y_train, ids_train = conv_processor.get_data(train_data, all_data,
↪  train=True)
X_train, X_val, y_train, y_val, ids_train, ids_val = train_test_split(X_train,
↪  y_train, ids_train, test_size=0.2, stratify=y_train, random_state=42)

vocab_size = conv_processor.tokenizer.num_words if
↪  conv_processor.tokenizer.num_words <
↪  len(conv_processor.tokenizer.word_index)+1 else
↪  len(conv_processor.tokenizer.word_index)+1
padding_len = X_train.shape[1]
output_number = y_train.shape[1]

checkpoint_path = get_drive_path("saved models\\saved-model-conv.hdf5")

number_of_dimensions = 100

embedding_path =
↪  get_drive_path(f"data\\fasttext\\ft_corpus_cleaned_{number_of_dimensions}_{cleaned_cor
embedding_matrix = eb.load_fast_text_trained(embedding_path,
↪  conv_processor.tokenizer, number_of_dimensions, cleaned_corpus)
```

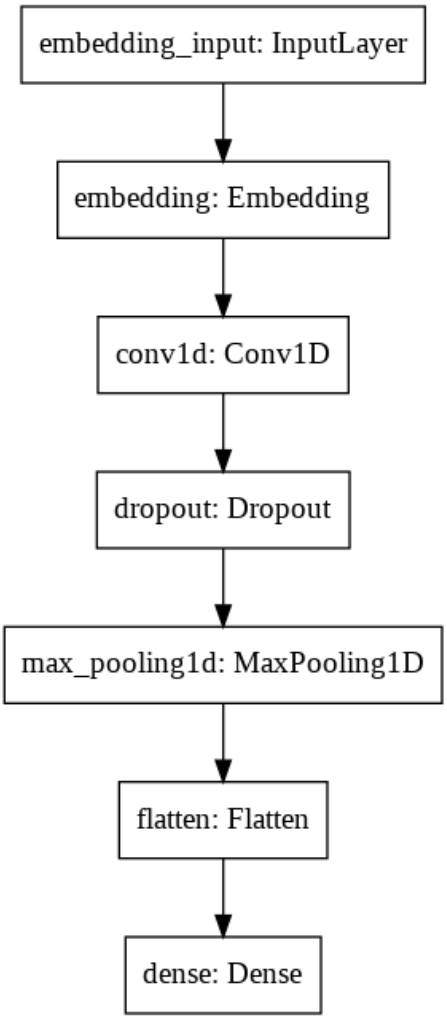

Рис. 10: Архитектура модели

Обучим модель:

```
conv1d_model,history = conv.train_conv1d_model(X_train, X_val, y_train, y_val,
                                               vocab_size, number_of_dimensions,
                                               ↪  embedding_matrix, padding_len,
                                               ↪  output_number, checkpoint_path)
```

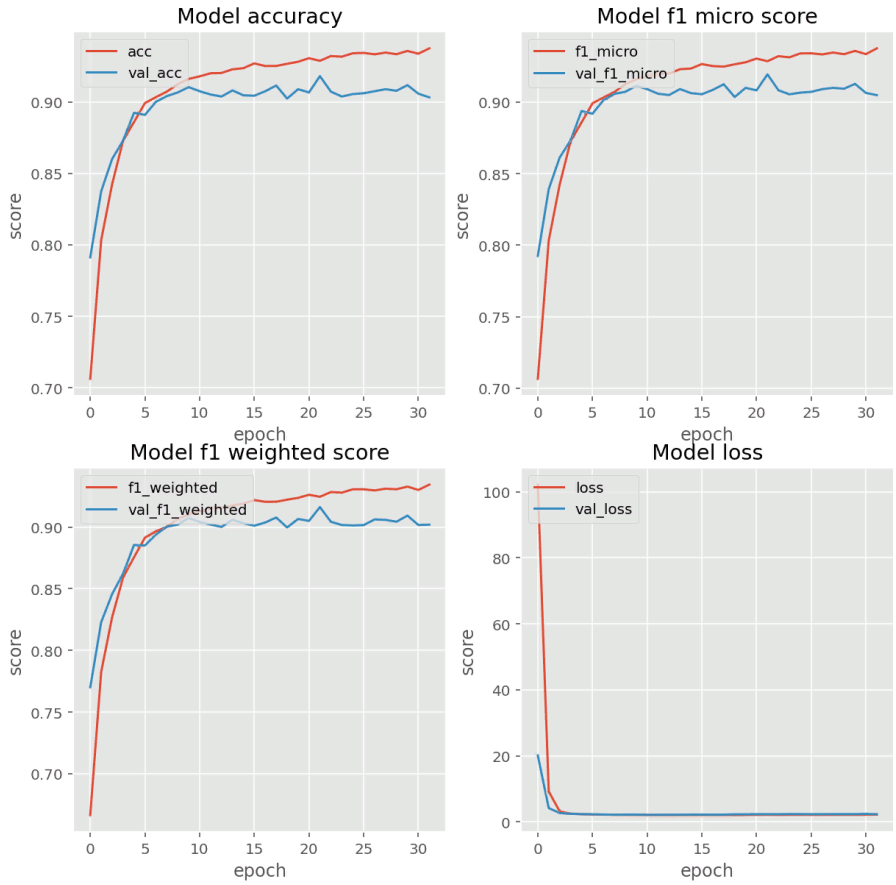

Рис. 11: Метрики модели во время обучения

2.2.1.2 Ансамбль моделей с одной и той же конфигурацией

Все ансамбли такого типа используют как основу абстрактный класс ensemble.EnsembleTemplate, который реализует процесс обучения модели на StratifiedKFold:

```
class EnsembleTemplate:

    def __init__(self, n_splits, model_folder_path, model_name):
        ...

    def train(self, data, retrain=False):
        '''
        Тренирует модель на KFold разделе выборки data

        Parameters:
        data: pandas.DataFrame
        retrain: bool - игнорировать ли сохраненные модели
```

```
        '''

        object_path = os.path.join(self.model_folder_path, self.model_name+'.pkl')
        ↪  #для сохранения/загрузки объекта
        self.models = []

        ...

        kf = StratifiedKFold(n_splits=self.n_splits, shuffle=True)

        ...

        for i, idxs in enumerate(kf.split(data, data['class']), 1):

            train_index, val_index = idxs

            ...

            try:
                self.models.append(self.fit(data.iloc[train_index],
                ↪  data.iloc[val_index]))

                 ...
```

Для выбора ответа ансамбля используется мода (голосование), но также был реализован stacking при помощи логистической регрессии (и попытка использовать нейронную сеть), но в итоге лучший результат всегда давала мода.

Код для обучения ансамбля conv:

```
conv_ensemble = ensemble.ConvEnsemble(full_corpus, cleaned_corpus_version,
↪  cleaned_corpus_path, path_to_shortenings_file, fix_spelling, stop_words,
↪  number_of_dimensions, embedding_path, checkpoint_path,
                                    n_splits=5,
                                    model_folder_path=get_drive_path('saved
                                    ↪  models/conv ensemble'),
                                    model_name='conv ensemble')

train_data_ens, val_data_ens = train_test_split(train_data, test_size=0.1,
↪  stratify=train_data['class'])
conv_ensemble.train(train_data_ens, retrain=True)
```

2.2.1.3 Оптимизации модели

В данной секции будут описаны различные предпринятые способы улучшения модели/нахождения альтернатив.

Ембеддинги: изначальным вариантом ембеддингов являлся Word2Vec (были перепробованы вектора, предобученные на корпусе Тайга (большая часть датасета - художественная литература, есть немного новостей и текста из социальных сетей, поэзия при обучении были исключена), Национальном корпусе русского языка). Также был опробован Navec (квантизированные вектора GloVe, обученные на данных из онлайн-библиотеки lib.rus.ec).

Из предобученных векторов fasttext использовались вектора, обученные на следующих корпусах: Википедия+Лента (новости), Тайга, Национальный корпус русского языка, GeoWAC (дамп русскоязычных документов).

Fasttext и Word2Vec также были обучены на разных вариантах нашего датасета; каждый вариант отличается тем, как текст из датасета был предобработан. Список версий:

1. убираются стоп-слова, нелексикографические символы и числа. Если после этого токен состоит из одного символа, он удаляется;
2. первая версия + добавляется исключение: если токен - это 20 чисел, он заменяется на токен 'карта';
3. вторая версия + удаление дат при помощи модуля rutimeparser;
4. убираются стоп-слова и токены, имеющие нелексикографические символы и числа + удаление дат при помощи модуля rutimeparser + исключение;
5. четвертая версия + сокращения заменяются на полную форму;
6. пятая версия + исправление опечаток при помощи API Яндекс.Спеллер.

Из-за долгого времени, требуемого на обработку данных Яндекс.Спеллером перед подачей модели, а также на отсутствие прироста производительности модели, в качестве лучшего предобработанного датасета используется пятая версия.

Альтернатива сверточному слою: вместо сверточной сети также рассматривалась сеть с LSTM слоем, но она требовала больше вычислений и уступала по метрике на 10% (что может объясняться тем, что LSTM пытается учитывать контекст всего текста, при том факте, что в нашем датасете этого не требуется).

```
from keras.layers import LSTM

loss = 'categorical_crossentropy'
activation = 'softmax'

hidden = 60

model = Sequential()
embedding_layer = Embedding(input_dim=vocab_size, output_dim=number_of_dimensions,
↪  weights=[embedding_matrix], input_length=max_len, trainable=True)
model.add(embedding_layer)
model.add(LSTM(hidden, activation='relu', kernel_regularizer=reg.l2(0.0005)))
model.add(Dropout(0.2))
model.add(Dense(output_number, activation=activation))

model.compile(optimizer=Adam(learning_rate=0.0005,clipnorm=5), loss=loss,
↪  metrics=['acc',f1_micro,f1_weighted])
model.summary()

filepath = "saved models\\saved-model-lstm.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_f1_micro', verbose=0,
↪  save_best_only=True, mode='max')
early_stop = EarlyStopping(monitor='val_f1_micro', mode='max', verbose=1,
↪  patience=10)

callbacks = [checkpoint, early_stop]
batch_size = 128
epochs = 100

history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
↪  callbacks=callbacks, validation_split=0.2)
```

Альтернативная архитектура: был добавлен дополнительный входной слой:

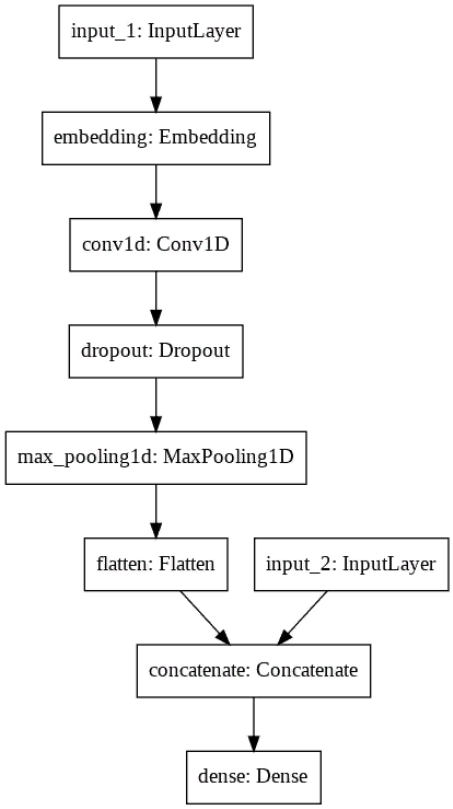

Рис. 12: Сверточная нейронная сеть с двумя входными слоями

В качестве содержимого второго слоя рассматривались следующие признаки:

1. Другие столбцы датасета.
   Реализация через модуль conv:

```
#доп признаки из других столбцов
conv_processor = conv.ConvPreprocessor(use_other_features=True,
↪  path_to_shortenings_file=path_to_shortenings_file,
↪  fix_spelling=fix_spelling)
X_train, y_train, other_features_train, ids_train =
↪  conv_processor.get_data(train_data, all_data, train=True)
X_train, X_val, y_train, y_val, other_features_train, other_features_val,
↪  ids_train, ids_val = train_test_split(X_train, y_train,
↪  other_features_train, ids_train, test_size=0.2, stratify=y_train,
↪  random_state=42)
```

   Опробованные признаки: нормализованный sum (через minmax), is_sum_int (является ли sum целым числом), to_acc, from_acc, последние два числа to_acc, последние два числа from_acc, склейка to_acc + from_acc, inn_same,

bic_same, payer_ОКВЭД_main, payee_ОКВЭД_main (коды ОКВЭД были найдены на основе ИИН). Отрывок из модуля:

```
features_dict = {}
        #not encoded features
        #features_dict['payer_ОКВЭД_main'] = data['payer_ОКВЭД_main']
        #features_dict['payee_ОКВЭД_main'] = data['payee_ОКВЭД_main']
        features_dict['is_sum_int'] = (data['sum'] % 1 == 0).astype('int')
        if not test:
            #encoded features
            encoder_dict = {}

            #features_dict['to_acc'], encoder_dict['to_acc'] =
            ↪  encode(data['to_acc'], LabelEncoder, all_data['to_acc'])
            #features_dict['from_acc'], encoder_dict['from_acc'] =
            ↪  encode(data['from_acc'], LabelEncoder, all_data['from_acc'])
            features_dict['sum'], encoder_dict['sum'] = encode(data['sum'],
            ↪  MinMaxScaler, all_data['sum'])
            #one-hot
            #features_dict['payer_ОКВЭД_main'],
            ↪  encoder_dict['payer_ОКВЭД_main'] =
            ↪  encode(data['payer_ОКВЭД_main'], OneHotEncoder,
            ↪  all_data['payer_ОКВЭД_main'])
            #features_dict['payee_ОКВЭД_main'],
            ↪  encoder_dict['payee_ОКВЭД_main'] =
            ↪  encode(data['payee_ОКВЭД_main'], OneHotEncoder,
            ↪  all_data['payee_ОКВЭД_main'])
            features_dict['to_acc'], encoder_dict['to_acc'] =
            ↪  encode(data['to_acc'], OneHotEncoder, all_data['to_acc'])
            features_dict['from_acc'], encoder_dict['from_acc'] =
            ↪  encode(data['from_acc'], OneHotEncoder, all_data['from_acc'])

            self.other_features_indices = [] #для перебора one-hot фич
```

Признаки to_acc, from_acc, их склейка, payer_ОКВЭД_main, payee_ОКВЭД_main были опробованы в двух вариантах: в виде label-енкодинга и в виде one-hot-енкодинга.
Код перебора признаков:

```
import itertools as it
combs_and_accs = []
```

```
for i in range(2, len(conv_processor.other_features_indices)+1):

    for comb in
     ↪  it.combinations(range(len(conv_processor.other_features_indices)), i):
        new_columns = list(it.chain(*[conv_processor.other_features_indices[i]
         ↪  for i in comb]))
        other_features_train_new = other_features_train.iloc[:, new_columns]
        other_features_val_new = other_features_val.iloc[:, new_columns]

        model, history =
         ↪  conv.conv_model_with_two_inputs(other_features_train_new,
         ↪  other_features_val_new)

        acc = max(history.history['val_f1_micro'])
        combs_and_accs.append([comb, acc])

res = pd.DataFrame(combs_and_accs, columns=['comb', 'acc'])
res.sort_values('acc', ascending=False)
```

	comb	acc
10	(0, 1, 2, 3)	0.908943
8	(0, 2, 3)	0.906711
6	(0, 1, 2)	0.901689
5	(2, 3)	0.901567
3	(1, 2)	0.900633
9	(1, 2, 3)	0.900572
1	(0, 2)	0.897564
0	(0, 1)	0.885445
4	(1, 3)	0.885227
7	(0, 1, 3)	0.884729
2	(0, 3)	0.883273

Рис. 13: Пример перебора признаков.

2. наличие коррелирующих с классом нграмм в тексте (нграмма - последовательности из n элементов, в данном случае - из n слов);

```
#доп признаки - нграммы
import utils        #модуль, созданный для нахождения нграмм

conv_processor = conv.ConvPreprocessor(use_other_features=False,
↪  path_to_shortenings_file=path_to_shortenings_file,
↪  fix_spelling=fix_spelling)
X_train, y_train, ids_train = conv_processor.get_data(train_data, all_data,
↪  train=True)
X_train, X_val, y_train, y_val, ids_train, ids_val = train_test_split(X_train,
↪  y_train, ids_train, test_size=0.2, stratify=y_train, random_state=42)

cleaned_train_data =
↪  train_data[['purp','class']].copy().rename({'class':'class_number'},
↪  axis=1)

cleaned_train_data['purp'] = [' '.join(sentence) for sentence in
↪  get_cleaned_features(train_data['purp'], stop_words)]
ngrams = utils.NGramsCounter(cleaned_train_data).choose_significant_ngrams(2,
↪  9, 0.7).columns
other_features_train = np.array([[int(ngram in ' '.join(sentence)) for ngram
↪  in ngrams] for sentence in get_cleaned_features(X_text_train,
↪  stop_words)])
other_features_val = np.array([[int(ngram in ' '.join(sentence)) for ngram in
↪  ngrams] for sentence in get_cleaned_features(X_text_val, stop_words)])
```

Несмотря на увеличение метрики при использовании нграмм во время начального этапа разметки (∼2000 размеченных транзакций), впоследствии при изменении методологии разметки коррелирующие с классами нграммы значительно сократились, и в итоге стали бесполезны.

3. соответствие правилам (подробнее о них написано в разделе ‘Классификация, основанная на правилах‘):

```
#доп признаки на основе правил
import rules

conv_processor = conv.ConvPreprocessor(use_other_features=False,
↪  path_to_shortenings_file=path_to_shortenings_file,
↪  fix_spelling=fix_spelling)
```

```
X_train, y_train, ids_train = conv_processor.get_data(train_data, all_data,
↪  train=True)
X_train, X_val, y_train, y_val, ids_train, ids_val = train_test_split(X_train,
↪  y_train, ids_train, test_size=0.2, stratify=y_train, random_state=42)

rule_predictor = rules.RulePredictor(all_data, code_names_path,
↪  sentences_features_path)

other_features_train =
↪  rule_predictor.get_one_hot_features(train_data.loc[ids_train])
other_features_val =
↪  rule_predictor.get_one_hot_features(train_data.loc[ids_val])
```

Оптимизация гиперпараметров и архитектуры: для этой цели использовался модуль autokeras (пример оптимизации модели без второго входного слоя):

```
import autokeras as ak
import tensorflow as tf

#для возможности взять готовую embedding_matrix (по сути просто оригинальный
↪  keras.layers.Embedding в обертке autokeras)
class CustomEmbedding(ak.Block):
    def __init__(self, input_dim, output_dim, weights, input_length, trainable,
    ↪  **kwargs):
        super().__init__(**kwargs)
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.weights = weights
        self.input_length = input_length
        self.trainable = trainable

    def build(self, hp, inputs=None):
        regularization = hp.Choice('reg_embedding', values = [5e-1, 1e-1, 5e-2,
        ↪  1e-2, 5e-3, 1e-3, 5e-4])
        input_node = tf.nest.flatten(inputs)[0]
        layer = tf.keras.layers.Embedding(input_dim=self.input_dim,
                                          output_dim=self.output_dim,
                                          weights=self.weights,
```

```
                                        input_length=self.input_length,
                                        trainable=self.trainable,

                                        ↪  embeddings_regularizer=reg.l1(regularization
        output_node = layer(input_node)
        return output_node

loss = 'categorical_crossentropy'

text_input = ak.Input()
output = CustomEmbedding(input_dim=vocab_size,
                         output_dim=number_of_dimensions,
                         weights=[embedding_matrix],
                         input_length=max_len,
                         trainable=True)(text_input)
output = ak.ConvBlock()(output)
output = ak.ClassificationHead()(output)

clf = ak.AutoModel(
    inputs=text_input,
    outputs=output,
    overwrite=True,
    max_trials=150,
    metrics='accuracy',
    loss=loss,
    objective='val_accuracy')

callbacks = []

clf.fit(X_train, y_train,
        epochs=30, callbacks=callbacks,
        validation_data=(X_val, y_val), verbose=1)

model = clf.export_model()
```

```
Trial 54 Complete [00h 02m 06s]
val_accuracy: 0.6198762655258179

Best val_accuracy So Far: 0.9010053873062134
Total elapsed time: 01h 59m 35s

Search: Running Trial #55

Hyperparameter    |Value             |Best Value So Far
custom_embeddin...|0.005             |0.005
conv_block_2/ke...|3                 |3
conv_block_2/se...|False             |False
conv_block_2/ma...|False             |False
conv_block_2/dr...|0                 |0
conv_block_2/nu...|3                 |2
conv_block_2/nu...|2                 |2
conv_block_2/fi...|64                |64
conv_block_2/fi...|32                |32
conv_block_2/fi...|16                |16
conv_block_2/fi...|32                |32
classification_...|global_max        |global_max
classification_...|0                 |0
optimizer         |adam              |adam
learning_rate     |0.001             |0.001
```

Рис. 14: Пример вывода autokeras

Пример выше перебирает не только гиперпараметры модели, но и разные архитектуры (разное количество слоев, разные сверточные блоки). Autokeras сверточной нейронной сети с двумя входными слоями:

```
loss = 'categorical_crossentropy'

text_input = ak.Input()
vector_input = ak.Input()

text_output = CustomEmbedding(input_dim=vocab_size,
                              output_dim=number_of_dimensions,
                              weights=[embedding_matrix],
                              input_length=max_len,
                              trainable=True)(text_input)
text_output = ak.ConvBlock()(text_output)

output = ak.Merge()([vector_input, text_output])
output = ak.ClassificationHead()(output)

clf = ak.AutoModel(
    inputs=[text_input, vector_input],
    outputs=output,
    overwrite=True,
    max_trials=45,
```

```
    loss=loss,
    metrics=['accuracy'],
    objective='val_accuracy')

callbacks = []

clf.fit([X_train, other_features_train], y_train,
        epochs=30, callbacks=callbacks,
        validation_data=([X_val,other_features_val], y_val), verbose=1)
```

2.2.2 Метод опорных векторов

2.2.2.1 Лучшая конфигурация

Инициализация и обучение модели:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.svm import LinearSVC

X_train, X_val, y_train, y_val = train_test_split(train_data['purp'],
↪  train_data[target_name], test_size=0.2, stratify=train_data[target_name],
↪  random_state=42)

vectorizer = TfidfVectorizer()
train_vectors = vectorizer.fit_transform(X_train)
test_vectors = vectorizer.transform(test_data['purp'])

classifier_svm = LinearSVC()
classifier_svm.fit(train_vectors, y_train)
```

2.2.2.2 Оптимизации модели

Перебор параметров:

```
from sklearn.model_selection import GridSearchCV

model = SVC()
params = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
          'C': [10, 5, 1, 0.5, 0.1, 0.05, 0.01],
          'max_iter': [-1],
          'break_ties': [True]}

model_grid = GridSearchCV(model,
                          params,
                          cv=5, n_jobs=-1, verbose=1, iid=False)

model_grid.fit(train_vectors, train_data[target_name])
```

Добавление нграмм: к матрице TF-IDF прикрепляется one-hot матрица нграмм:

```
ngrams_range = (6,12)
corr_rate = 0.5
dataset_date = '07-01-2021'

path_to_shortenings_file = get_drive_path('data\\Сокращения.xlsx') if True else
↪  None
fix_spelling = False

cleaned_train_data = train_data[['purp',target_name]].copy()
stop_words = stopwords.words('russian')

cleaned_train_data['purp'] = [' '.join(sentence) for sentence in
↪  get_cleaned_features(train_data['purp'], stop_words, path_to_shortenings_file,
↪  fix_spelling)]

ngrams_path =
↪  get_drive_path(f'data\\ngrams_{ngrams_range[0]}-{ngrams_range[1]-1}_{corr_rate}_{datas
try:
  ngrams = np.load(ngrams_path, allow_pickle=True)
  train_ngrams = np.array([[int(ngram in sentence) for ngram in ngrams] for
  ↪  sentence in cleaned_train_data['purp']])
except:
  train_ngrams =
  ↪  utils.NGramsCounter(cleaned_train_data.loc[:3000]).choose_significant_ngrams(*ngrams
  ↪  corr_rate)
  ngrams = train_ngrams.columns
  train_ngrams = np.array([[int(ngram in sentence) for ngram in ngrams] for
  ↪  sentence in cleaned_train_data['purp']])
  np.save(ngrams_path, ngrams.to_numpy(), allow_pickle=True)

vectorizer = TfidfVectorizer()
train_vectors = vectorizer.fit_transform(train_data['purp'])
train_vectors = np.hstack((train_vectors.toarray(),train_ngrams))

classifier_svm = LinearSVC()
classifier_svm.fit(train_vectors, train_data[target_name])
```

На начальных этапах разметки качество такой модели было лучше базовой, но со временем польза нграмм исчезла.

Правила: такой же принцип как и у нграмм:

```
rule_predictor = rules.RulePredictor(all_data,
↪  get_drive_path('data/Классы_кодировка.csv'),
↪  get_drive_path('data/Фичи_17-12-2020.xlsx'))
other_features_train = rule_predictor.get_one_hot_features(train_data)

vectorizer = TfidfVectorizer()
train_vectors = vectorizer.fit_transform(train_data['purp'])
train_vectors = np.hstack((train_vectors.toarray(),other_features_train))

classifier_svm = SVC()
classifier_svm.fit(train_vectors, train_data[target_name])
```

2.2.3 BERT

Инициализация и обучение модели:

```
import torch
import bert

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
num_classes = len(all_data[target_name].unique())

bert_processor = bert.BertProccessor()

X, other_features, y = bert_processor.get_data(train_data, all_data=all_data)
X_train, X_val, y_train, y_val, other_features_train, other_features_val =
↪  train_test_split(X, y, other_features, test_size=0.2, stratify=y,
↪  random_state=42)

bert_model = bert.BertForClassification(num_classes).to(device)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(bert_model.parameters(), lr=1e-5)
epochs = 6

bert_model.train_with_chunks(X_train, other_features_train, y_train,
                             X_val, other_features_val, y_val,
                             epochs=epochs, optimizer=optimizer, loss_fn=loss_fn,
                             ↪  device=device)
```

2.2.3.1 Ансамбль моделей с одной и той же конфигурацией

Обучение моделей:

```
bert_ensemble = ensemble.BertEnsemble(n_splits=5,
                            model_folder_path=get_drive_path('saved models/bert
                            ↪  ensemble'),
                            model_name='bert ensemble',
                            num_classes=num_classes,
                            all_data=all_data,
                            batch_size=9)

train_data_ens, val_data_ens = train_test_split(train_data, test_size=0.2,
↪  stratify=train_data['class'], random_state=42)
bert_ensemble.train(train_data_ens, retrain=True)
```

2.2.4 Случайный лес

Инициализация и обучение модели:

```
from sklearn.ensemble import RandomForestClassifier
import random_forest

rf_processor = random_forest.RandomForestProcessor()
X, y = rf_processor.get_data(train_data, all_data)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
↪  random_state=42)

best_params = {'bootstrap': False,
               'max_depth': 100,
               'max_features': 'auto',
               'min_samples_leaf': 2,
               'min_samples_split': 5,
               'n_estimators': 700}

clf = RandomForestClassifier(**best_params)
clf.fit(X_train, y_train)
```

2.2.4.1 Оптимизации модели

Перебор параметров:

```
params = {'bootstrap': [False],
          'max_depth': range(40,150,20),
          'max_features': ['auto'],
          'min_samples_leaf': range(2,13,5),
          'min_samples_split': range(5,46,20),
          'n_estimators': [700]}

clf = RandomForestClassifier()
model_grid = GridSearchCV(clf, params, cv=5, n_jobs=-1, verbose=1, iid=False)

model_grid.fit(X, y)
```

2.2.5 RoBERTa

Инициализация и обучение модели:

```
import roberta

num_classes = len(all_data[target_name].unique())
epochs = 4
checkpoint_path = get_drive_path("saved models\\saved-model-roberta.hdf5")

roberta_processor = roberta.RobertaProcessor()

tokens_train, attn_mask_train, y_train = roberta_processor.get_data(train_data,
↪  all_data)
tokens_train, tokens_val, attn_mask_train, attn_mask_val, y_train, y_val =
↪  train_test_split(tokens_train, attn_mask_train, y_train,




model = roberta.train_roberta_model(tokens_train, tokens_val, attn_mask_train,
↪  attn_mask_val, y_train, y_val,
                                    num_classes, checkpoint_path, epochs)
model.load_weights(checkpoint_path)
```

2.3 Ансамбли

После обучения каждой из моделей в словари сохранялись следующие данные:

1. предсказания на тестовой выборке (для получения аналогичного предсказания на ансамбле);
2. предсказания на валидационной выборке (для обучения stacking ансамблей);
3. precision всех классов на валидационной выборке (для ансамблей, использующих precision).

Пример для conv:

```
model_name = 'conv'
pred_dict[model_name] =
 ↪  conv_processor.y_encoder.inverse_transform(pred_conv_binary)
pred_val_dict[model_name] =
 ↪  conv_processor.y_encoder.inverse_transform(np.round(conv1d_model.predict(X_val)))
precision_dict[model_name] = precision_score(train_data.loc[ids_val][target_name],
 ↪  pred_val_dict[model_name], average=None)
```

Данные по ансамблю одной модели сохранялись вместо базовой, если метрика ансамбля оказалась лучше метрики базовой модели.

2.3.1 Ансамбли различных моделей с голосованием

Последовательность действий:

1. собрать все предсказания (и, если нужно, данные по precision) моделей в один объект;
2. итерироваться по всем возможным комбинациям моделей;
3. на каждой итерации получать предскания ансамбля на основе имеющейся комбинации моделей;
4. сохранять метрику для каждой итерации.

Таким образом, можно найти самую оптимальную комбинацию моделей для каждого вида ансамбля.

Реализация перебора ансамблей, где выбирается модель с максимальным precision по предсказанному классу:

```
#one-hot таргет тестовой выборки
_, y_test, _ = conv_processor.get_data(test_data, train=False)

preds = np.array([conv_processor.y_encoder.transform(pred.reshape(-1,1)).toarray()
↪  for pred in pred_dict.values()])
precisions = np.array(list(precision_dict.values()))

n = len(preds)
idx = range(n) #индексы моделей (каждому номеру соответствует одна из моделей в
↪  ансамбле)
names = np.array(list(pred_dict.keys()))
summary = []

#перебираем все возможные комбинации предсказаний
for r in range(1, n+1):
    for comb in combinations(idx, r):

        preds_comb = preds[list(comb)]
        precisions_comb = precisions[list(comb)]

        new_pred = ensemble_predict(preds_comb, precisions_comb, vote=False)

        summary.append([names[list(comb)],
```

```
                        f1_score(y_test, new_pred, average='micro'),
                        f1_score(y_test, new_pred, average='weighted')])

summary = pd.DataFrame(summary, columns=['Комбинация','f1 micro','f1 weighted'])
summary.sort_values('f1 micro', ascending=False)
```

Комбинация	f1 micro	f1 weighted
[bert]	0.952724	0.949619
[svm, bert]	0.951182	0.948050
[svm, bert, rf]	0.949640	0.946463
[bert, rf]	0.949126	0.944340
[bert, roberta]	0.948613	0.945352
[bert, conv]	0.948099	0.944833
[svm, bert, conv]	0.948099	0.944902
[svm, bert, roberta]	0.947585	0.944479
[svm, bert, rf, conv]	0.946557	0.943271
[svm, bert, rf, roberta]	0.946043	0.942779
[bert, conv, roberta]	0.945529	0.942133
[bert, rf, roberta]	0.945529	0.940655
[bert, rf, conv]	0.945529	0.940604
[svm, bert, conv, roberta]	0.945015	0.941725
[svm, bert, rf, conv, roberta]	0.943474	0.940094
[bert, rf, conv, roberta]	0.942446	0.937410
[svm]	0.936280	0.934278

Рис. 15: Пример перебора ансамблей

Ансамбли с голосованиями перебираются аналогично с другими параметрами в функции ensemble_predict.

2.3.2 Stacking-ансамбли различных моделей

Данные для ансамблей:

```
preds_val = np.array(

    ↪ [conv_processor.y_encoder.transform(pred_val_dict[model_name].reshape(-1,1)).toarr
    ↪ for model_name in pred_val_dict]
```

```
    )
preds = np.array(

    ↪  [conv_processor.y_encoder.transform(pred_dict[model_name].reshape(-1,1)).toarray()
    ↪  for model_name in pred_dict]
)
_, y_val = train_test_split(train_data[target_name],
↪  stratify=train_data[target_name], test_size=0.2, random_state=42)
y_test = test_data[target_name]
```

Перебор stacking-ансамблей, где мета модель является логистической регрессией:

```
from sklearn.linear_model import LogisticRegression

n = preds.shape[0]
idx = range(n) #какие предсказания используются
names = np.array(list(pred_dict.keys()))
summary = []

for r in range(1, n+1): #комбинации из i элементов
    for comb in combinations(idx, r):

        preds_val_comb = preds_val[list(comb)]
        preds_comb = preds[list(comb)]

        lr_model_ens = LogisticRegression(max_iter=1000, solver='lbfgs')

        preds_val_comb = np.hstack(preds_val_comb)
        preds_test_comb = np.hstack(preds_comb)

        lr_model_ens.fit(preds_val_comb, y_val)

        pred_ens = lr_model_ens.predict(preds_test_comb)

        summary.append([names[list(comb)],
                        f1_score(y_test, pred_ens, average='micro'),
                        f1_score(y_test, pred_ens, average='weighted')])

summary = pd.DataFrame(summary, columns=['Комбинация','f1 micro','f1 weighted'])
summary.sort_values('f1 micro', ascending=False)
```

Для svm stacking-ансамблей были оптимизированы параметры на основании ансамбля всех моделей:

```
svm_model_ens = SVC()

params = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
          'C': [5, 1, 0.5, 0.1, 0.05],
          'max_iter': [-1]}

model_grid = GridSearchCV(svm_model_ens,
                          params,
                          cv=5, n_jobs=-1, verbose=1)

model_grid.fit(np.hstack(preds_val), y_val)
```

2.4 Правила

Кроме методов машинного обучения также велась разработка правил, которые могли использоваться вместе с вышеперечисленными моделями. Все правила состоят регулярных выражений, проверяющих наличие/отсутствие в текстовом описании транзакции определенных слов или порядка слов. Некоторые имеет дополнительные условия, затрагивающие другие признаки транзакции: name_from, name_to, sum.

Главным недостатком rule-based подхода, кроме временных затрат, является необходимость их поддерживать: при каждой новом пересмотре методологии разметки часть правил становились нерелевантными. Из-за этого правила множество раз пересматривались и дополнялись. Но при этом из-за более детальной работы над методологией обнаруживались новые паттерны, которые впоследствии добавлялись как правила. На момент написания статьи имеется 126 правил.

№	name_from	name_to	purp	class
1			Есть: аренда Нет: возврат, лизинг	Аренда
2			Есть: Недостача Нет:	Возврат
3			Есть: Цессия Нет: концессия	Кредит
4		Физические лица	Есть: Дотация Нет: Алименты	Выплаты соцхарактера
5			Есть: Возврат Нет: Депозит, заём, безвозвратный	Возврат
6			Есть: Страховая выплата Нет:	Страховое возмещение

Рис. 16: Пример правил

Пример хранения правил на первой итерации:

```
self.cond_class = [
    [class_to_code['выплаты соцхарактера'], lambda x: ' дотация ' in
    ↪  spaces(x['purp']) and x['name_to'] == 'Физические лица'],
    [class_to_code['выплаты соцхарактера'], lambda x: ' помощь ' in
    ↪  spaces(x['purp'])],
    [class_to_code['возврат'], lambda x: ' недостача ' in spaces(x['purp'])],
    [class_to_code['зарплата'], lambda x:
    ↪  regex.match(r'(?<!ком.*)(?<!расходы.*)аванс(?!.*ком)(?!.*расходы)',
    ↪  x['purp']) and x['name_to'] == 'Физические лица' and x['name_from'] ==
    ↪  'Коммерческие организации'],
```

```
    [class_to_code['страховое возмещение'], lambda x: 'страх' in x['purp'] and
    ↪  x['name_to'] == 'Физические лица' and x['name_from'] == 'Финансовые
    ↪  организации'],
    ]
```

Хранение правил сейчас:

```
self.rules = RuleStorer()

self.rules.add([
    0, 'аренда', lambda x:
    ↪  regex.search(r'(?<!возврат.*)(?<!лизинг.*)аренда(?!.*лизинг)',
    ↪  x['purp']),
    1, 'возврат', lambda x: 'недостача' in x['purp'],
    2, 'кредит', lambda x: 'цессия' in x['purp'] and 'концессия' not in
    ↪  x['purp'],
    3, 'выплаты соцхарактера', lambda x:
    ↪  regex.search(r'(?<!алименты.*)дотация(?!.*алименты)', x['purp']) and
    ↪  x['name_to'] == 'Физические лица',
    4, 'возврат', lambda x:
    ↪  regex.search(r'(?<!депозит.*?)(?<!заём.*?)возврат(?!.*заём)(?!.*депозит)',
    ↪  x['purp']) and 'безвозвратный' not in x['purp'],
    ])
```

Использование правил на практике:

1. описание транзакции предобратывается: перевод в нижний регистр; токенизация при помощи модуля razdel, специализирующемся на русском тексте; лемматизация при помощи pymorphy2;

2. транзакция проверяется на все правила; если ни одно правило не соблюдается, то предсказание делает модель машинного обучения; если соблюдаются правила разных классов, происходит то же самое;

3. после классификации всех транзакций и сбора статистики по ее корректности можно отключить правила с неудовлетворимой точностью для повышения качества на тестовой выборке.

```
#предсказания и precision моделей из лучшего ансамбля
comb = ['conv', 'svm', 'bert', 'rf']
preds_comb =
↪  np.array([conv_processor.y_encoder.transform(pred_dict[model_name].reshape(-1,1)).toar
↪  for model_name in comb])
```

```
precisions_comb = np.array([precision_dict[model_name] for model_name in comb])

code_names_path, sentences_features_path =
↪  get_drive_path('data/Классы_кодировка.csv'),
↪  get_drive_path('data/Фичи_17-12-2020.xlsx')
rule_predictor = rules.RulePredictor(all_data, code_names_path,
↪  sentences_features_path)

remove_bad_rules = True #убрать правила с плохой производительностью (для этого
↪  запустить текущую и след ячейку, а потом еще раз текущую)
try:
    if not remove_bad_rules:
        raise NameError
    for i in bad_rules[::-1]:
        rule_predictor.cond_class.pop(i)
except NameError:
    pass

rule_preds_list = rule_predictor.classify_with_rules(test_data,
↪  use_key_sentences=False) #предсказание на основе правил и ключевых слов, если
↪  предсказания нет, то будет возвращено None
ensemble_rule_pred =
↪  conv_processor.y_encoder.inverse_transform(ensemble_predict(preds_comb,
↪  precisions_comb, vote=True))
for i,prediction in enumerate(rule_preds_list):
    if prediction is not None:
        ensemble_rule_pred[i] = prediction
```

Несмотря на трудоемкость и неспособность классифицировать любую транзакцию, можно добиться более высокого качества классификации: на последней версии разметки с удалением правил с точностью меньше 0.9 получаем f1 micro на уровне 0.97 на тестовой выборке при охвате 60% транзакций.

Использование ключевых слов: на втором этапе вместо передачи транзакции другой модели можно так же проверить, содержатся ли в тексте транзакции ключевые слова (нграммы), определяющие классы. Таким образом можно увеличить покрытие транзакций, но при этом точность определения класса ниже.

Классы	Однозначно	Неоднозначно	Комментарии
Картотека			
Вынос на просрочку			
Гашение просрочки			
Докапитализация			
Комиссии банку	Комиссия за взнос наличных;комиссия за расчетно-кассовое обслуживание;Комиссия за перевод денежных средств с банковского счета;Комиссия за перевод денежных средств в вал.РФ с банковского счета;Комиссия за прием и зачисл. наличных ден. ср-в;Комиссия за зачисление на Карточный счет;Комиссия за услуги по договору;Комиссия за зачисление выручки ч/з АДМ;Комиссия за взнос наличных на корпоративный карточный счет;Комиссия за выдачу гарантии;Комиссия Банка;Комиса прием и зачис на банк счет клиента налич денежных средств;Комиссия за исполнение функций агента валютного контроля;комиссия за	комиссия;комис;комисс;комиссионное вознаграждение	
Суды		суд.пристав, суд.приказ;выд. Арбитражный суд;суд.пристав-исполнитель;Взыскание согласно;Удержано с;Взыскание задолженности с;УПДУ_Удержано с;УОиПУ_Взыскание задолженности с	

Рис. 17: Пример ключевых слов

3 Оценка моделей

3.1 Выбор метрики

Для оценки классификационных моделей имеется большое количество метрик. Самая интуитивная из них - точность, но она и самая поверхностная, так как плохо учитывает прогнозы для всех классов в совокупности (типичный пример - классификация датасета, где один из классов превалирует). Для учета того, как хорошо модель предсказывает каждый класс, используются метрики precision и recall. Precision показывает, насколько модель подвержена ошибке первого рода (неверно предсказано, что входные данные не относятся к данному классу, или false negative):

$$precision = \frac{truepositives}{truepositives + falsepositives}$$

Recall показывает, насколько модель подвержена ошибке второго рода (неверно предсказано, что входные данные относятся к данному классу, или false positive):

$$recall = \frac{truepositives}{truepositives + falsenegatives}$$

В нашем случае нет предпочтения recall или precision, поэтому использоваться будет метрика f1, требующая как высокую precision, так и высокую recall:

$$f1 = \frac{2 * precision * recall}{precision + recall}$$

Так как f1 считается для каждого класса отдельно по принципу one-vs-rest (то есть один класс считается положительным, а все остальные - отрицательными), то требуется ее усреднить. Таким образом, основной метрикой была выбрана f1 с методом усреднения микро (считается глобального количество true positives, false negatives, false positives для всех классов, которые используется для расчета среднего f1). Также использовался взвешенный метод (среднее f1 - это взвешенная сумма f1 по всем классам, где веса - доля класса в датасете).

3.2 Оценка моделей на тестовой выборке

Лучшей моделью оказался ансамбль голосования из метода опорных векторов, BERT'а и RoBERTa с правилами, лучшей одиночной моделью - BERT и лучшим ансамблем из одной модели тоже BERT.

	f1 micro	f1 weighted
best ensemble + rules	0.9479	0.9456
vote ensemble	0.9474	0.9445
vote precision ensemble	0.9474	0.9442
logreg stacking	0.9439	0.9411
svm stacking	0.9424	0.9388
bert ensemble	0.9399	0.9385
bert	0.9394	0.9378
max precision ensemble	0.9394	0.9378
svm	0.9334	0.9299
conv ensemble	0.9139	0.9103
roberta	0.9139	0.9089
conv	0.9059	0.9015
random forest	0.8958	0.8884
svm ensemble	0.8913	0.8826
random forest ensemble	0.8853	0.8731

Рис. 18: Метрики моделей на тестовой выборке

Заключение

В рамках работы Финансового университета с Газпромбанком по классификации транзакционных данных было осуществлено следующее:

1. проведены различные способы предобработки текста: лемматизация, разные способы очистки от мусора, исправление опечаток, возврат сокращений в полную форму;

2. опробованы способы векторного представления текста: TF-IDF, word2vec, NaVec, FastText; для word2vec и FastText были использованы как предобученные на больших объемах данных ембеддинги, так и обученные на нашем датасете (также были попытки дообучить ембеддинги, скомбинировав таким образом оба способа);

3. построены модели классификации текста: сверточная нейронная сеть, метод опорных векторов, случайный лес, BERT, RoBERTa, rule-based;

4. для них предпринимались различные оптимизации, модификации и альтернативы: использование доп данных для сверточной нейронной сети, метода опорных векторов (другие признаки в датасете, one-hot коррелирующих нграм, one-hot соблюдения правил и наличия ключевых слов); попытка использовать LSTM; ансамбли моделей одной конфигурации на кросс-валидации; ансамбли всех моделей (в том числе одиночных ансамблей); разные способы составления ансамблей (голосование, ответ с максимальным precision, голосование precision, stacking на основе логистической регрессии, stacking на основе svm, stacking на основе нейронной сети); перебор возможных комбинаций для всех видов многомодельных ансамблей; оптимизации гипер-параметров и архитектур.

name_to	name_from	purp	sum	class	class prediction
Физические лица	Коммерческие организации	оплата больничного листа за февраль 2019г. на ...	27238.00	зарплата	зарплата
Физические лица	Коммерческие организации	[redacted] выплата подработки за март прик...	1500.00	оплата фл	перевод на фл
Физические лица	Физические лица	перевод в погашение по кд с/з клиента от 22/01...	17500.00	кредит	кредит
Коммерческие организации	Коммерческие организации	теплоэнергия,гвс по нормативам,по адресу:[redacted]	4392.67	выручка	выручка
Коммерческие организации	Коммерческие организации	перечисление по договору процентного займа б/н...	1000000.00	займ	займ
Коммерческие организации	Коммерческие организации	за эл энергию по договору п [redacted] от ...	6953.94	выручка	выручка
Коммерческие организации	Коммерческие организации	оплата по договору [redacted] от 30.07.2019 с...	1000261.60	выручка	выручка
Физические лица	Физические лица	пополн. счета банковск. карты [redacted] ...	500000.00	пополнение счета	пополнение счета
Незавершенные расчеты с операторами услуг плат...	Средства для выдачи и внесения наличных денег ...	карта:[redacted] операция: снят ие нал...	35460.00	прочее	прочее
Коммерческие организации	Коммерческие организации	оплата по дистрибьюторскому договору 1/дис от ...	349000.00	выручка	выручка

Рис. 19: Работа лучшей модели на тестовой выборке

Данная работа выполнена мною самостоятельно

01.06.2021

Список использованных источников

[1] Natural Language Processing Advancements By Deep Learning: A Survey [Электронный ресурс]. URL: https://arxiv.org/pdf/2003.01200.pdf (дата обращения 15.05.21)

[2] An Introduction to Convolutional Neural Networks [Электронный ресурс]. URL: https://arxiv.org/pdf/1511.08458.pdf (дата обращения 15.05.21)

[3] Efficient Estimation of Word Representations in Vector Space [Электронный ресурс]. URL: https://arxiv.org/pdf/1301.3781.pdf (дата обращения 01.05.21)

[4] GloVe: Global Vectors for Word Representation [Электронный ресурс]. URL: https://nlp.stanford.edu/pubs/glove.pdf (дата обращения 01.05.21)

[5] Enriching Word Vectors with Subword Information [Электронный ресурс]. URL: https://arxiv.org/pdf/1607.04606.pdf (дата обращения 01.05.21)

[6] Navec - компактные эмбеддинги для русского языка [Электронный ресурс]. URL: https://natasha.github.io/navec/ (дата обращения 01.05.21)

[7] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Электронный ресурс]. URL: https://arxiv.org/pdf/1810.04805.pdf (дата обращения 01.05.21)

[8] Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation [Электронный ресурс]. URL: https://arxiv.org/pdf/1609.08144v2.pdf (дата обращения 01.05.21)

[9] Attention Is All You Need [Электронный ресурс]. URL: https://arxiv.org/pdf/1706.03762.pdf (дата обращения 01.05.21)

[10] RoBERTa: A Robustly Optimized BERT Pretraining Approach [Электронный ресурс]. URL: https://arxiv.org/pdf/1907.11692.pdf (дата обращения 01.05.21)

[11] Understanding Random Forests: From Theory to Practice [Электронный ресурс]. URL: https://arxiv.org/pdf/1407.7502.pdf (дата обращения 15.05.21)

[12] Python 3.7 documentation [Электронный ресурс]. URL: https://docs.python.org/3.7/ (дата обращения 15.05.21)

[13] scikit-learn documentation [Электронный ресурс]. URL: https://scikit-learn.org/stable/ (дата обращения 15.05.21)

[14] PyTorch documentation [Электронный ресурс]. URL: https://pytorch.org/docs/stable/index.html (дата обращения 15.05.21)

[15] TensorFlow documentation [Электронный ресурс]. URL: https://www.tensorflow.org/api_docs (дата обращения 15.05.21)