МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«Южный федеральный университет»

Институт математики, механики
и компьютерных наук им. И. И. Воровича

Кафедра информатики и вычислительного эксперимента

Гурин Иван Сергеевич

ОЦЕНКА ВЛИЯНИЯ НОВОСТЕЙ В СОЦИАЛЬНЫХ СЕТЯХ НА ФИНАНСОВЫЕ ПОКАЗАТЕЛИ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению подготовки
02.03.02 – Фундаментальная информатика и информационные технологии

Научный руководитель –
Старший преподаватель Ячменева Наталья Николаевна

_____ _____
оценка (рейтинг) подпись руководителя

Ростов-на-Дону – 2023

Задание на выпускную квалификационную работу
студента Гурин И. С.

Направление подготовки: 02.03.02 - Фундаментальная информатика и информационные технологии.

Студент: Гурин И. С.

Научный руководитель: ст. преп. Н.Н. Ячменева

Год защиты: 2023

Тема работы: ОЦЕНКА ВЛИЯНИЯ НОВОСТЕЙ В СОЦИАЛЬНЫХ СЕТЯХ НА ФИНАНСОВЫЕ ПОКАЗАТЕЛИ

Цель работы: Разработка приложения для прогнозирования выбранных финансовых котировок с оценкой влияния новостей на прогноз.

Задачи работы:

- Определиться с выбором датасета, содержащего конкретные финансовые показатели, например, данные с бирж, котировки валют. Реализовать модель для предсказания значения финансовых данных.
- Организовать сбор новостей финансовой окраски в социальных сетях.
- Реализовать предобработку новостей для использования в методах машинного обучения.
- Разработать модель/модели для предсказания значения финансовых данных с использованием полученных новостей.
- Проанализировать полученные результаты.

Научный руководитель Н.Н. Ячменева

Студент Гурин И. С.

30 ноября 2022 г

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

Южный Федеральный Университет

о результатах проверки текстового документа
на наличие заимствований

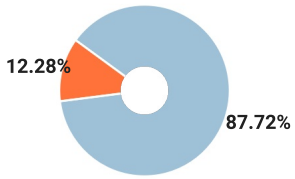
ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Гурин Иван Сергеевич
Самоцитирование рассчитано для: Гурин Иван Сергеевич
Название работы: ОЦЕНКА ВЛИЯНИЯ НОВОСТЕЙ В СОЦИАЛЬНЫХ СЕТЯХ НА ФИНАНСОВЫЕ ПОКАЗАТЕЛИ
Тип работы: Выпускная квалификационная работа
Подразделение: ИММиКН

РЕЗУЛЬТАТЫ

СОВПАДЕНИЯ	12.28%
ОРИГИНАЛЬНОСТЬ	87.72%
ЦИТИРОВАНИЯ	0%
САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 18.06.2023



Структура документа: Проверенные разделы: основная часть с.1, 3-32, содержание с.2, библиография с.33, приложение с.34-40
Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс*; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по коллекции Гарант: аналитика; Переводные заимствования по коллекции Интернет в английском сегменте; Переводные заимствования по Интернету (EnRu); Переводные заимствования по коллекции Интернет в русском сегменте; Переводные заимствования издательства Wiley ; eLIBRARY.RU; СПС ГАРАНТ: аналитика; СПС ГАРАНТ: нормативно-правовая документация; Медицина; Диссертации НББ; Коллекция НБУ; Перефразирования по eLIBRARY.RU; Перефразирования по СПС ГАРАНТ: аналитика; Перефразирования по Интернету; Перефразирования по Интернету (EN); Перефразированные заимствования по коллекции Интернет в английском сегменте; Перефразированные заимствования по коллекции Интернет в русском сегменте; Перефразирования по коллекции

Работу проверил: Ячменева Наталья Николаевна

ФИО проверяющего

Дата подписи:

Подпись проверяющего

Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.

Оглавление

Постановка задачи

Целью данной работы являлось создание программы, способной производить оценку влияния новостей в социальных сетях на финансовые показатели.

В связи с поставленной целью работы сформулированы следующие задачи:

1. Поиск и создание наборов данных для анализа влияния новостей из социальных сетей на финансовые показатели.
2. Алгоритмическая реализация применения методов предобработки к входным данным.
3. Применение классификатора с целью общего анализа сентимент-тональности новости.
4. Формирование и применение каскада регрессоров для анализа сентимент-тональности новостей. Исследование точности работы полученной модели.
5. Оценка влияния сентимент-тональности на финансовые показатели с помощью регрессивной модели. Исследование точности работы алгоритма.
6. Создание Telegram чат-бота для предоставления удобного интерфейса и гибкого взаимодействия с моделью.

Введение

В современном мире нередко какая-либо новость может, как и быть порождённой некоторым экономическим или социальным явлением, так и становиться причиной возникновения финансовых тенденций, или же быть фактором, влияющим на экономические показатели. Например, новость, опубликованная в социальных сетях от имени крупного финансового аналитика, о кризисе банковской системы, способна вызвать отток капиталов из банков, и, тем самым, привести к кризису банковской системы. Цель данной работы заключается в том, чтобы средствами библиотек для обработки естественного языка составить модель, способную по заданному новостному тексту из какой-либо социальной сети произвести общую оценку потенциального влияния новости на финансовые показатели, а также производить классификацию новостного текста на две категории - благоприятные и неблагоприятные. В качестве показателя, на который и оценивается влияние тех, или иных новостей, выступает процент, на который изменится стоимость ценных бумаг некоторой компании, фигурирующей в новости.

Также, в данной работе представлено общее описание построенной модели, её основных блоков и применяемых для построения модели инструментов и наборов данных. Помимо этого, в данной работе представлены примеры кода с использованием соответствующих функций, классов и методов.

В реализации некоторых применённых инструментов используются методы машинного обучения. В данной работе затронуты лишь основные моменты работы с данными инструментами. Без углубления в детали их технической реализации и/или детали математического доказательства состоятельности данных алгоритмов, так как это не предусмотрено тематикой данной работы.

1. Работа с наборами данных

1.1. Financial News Sentiment Dataset

Набор данных Financial News Sentiment Dataset (FiNeS) представляет из себя четко структурированный русскоязычный набор финансовых новостей о компаниях, производящих торги на Московской и СПБ биржах. Набор данных представлен в виде .tsv-файла. Также, он содержит в себе только финансовые новости, часть из которых относится к социальным сетям, что и подтверждает применимость данного набора данных в текущей работе. FiNeS содержит в себе свыше 500 публикаций и имеет следующую структуру:

- Title – колонка, содержащая в себе заголовок новости;
- Score – целевая переменная датасета. Является оценкой тональности новостного заголовка в виде вещественного числа в пределах от -1 до 1;
- Link – ссылка на источник новости;
- Summary – полный текст новостной публикации;
- Published – дата публикации новости;
- Tickers – краткие наименования компаний, упоминаемых в тексте новости.

В дальнейшем, при составлении модели понадобилось задействовать лишь новостные заголовки, полные тексты новости и значения целевой переменной. Столбцы title, summary, score, соответственно. Использование заголовков и полного текста в дальнейшей работе было сочтено целесообразным, ведь заголовки во многих случаях могут содержать краткую выжимку из статьи и могут быть эмоционально окрашены. А это и является важным аспектом, ведь «Score» и характеризует эмоциональный окрас новости, который потребовался в дальнейшем для анализа изменения курса акций компании, о которой и была составлена новость. В свою очередь, иногда, заголовки могут не содержать достаточного количества данных для

полноценного анализа текста.

Так как, разметка заголовков датасета проводилась его автором с помощью платформы “Яндекс Толока” методом Best-Worst Scalling, можно убедиться в надёжности разметки.

Целевая переменная подсчитана следующим образом: T = Npos / Ntotal - Nneg / Ntotal, где Ntotal - общее количество разметок заголовка, Npos - количество раз, когда заголовок был отмечен как более положительный, Nneg - количество раз, когда заголовок был отмечен как более отрицательный.

Согласно показателю Fleiss’ Kappa, приведённому в источнике и равному 0,2639, данный набор данных размечен согласованно и надёжно.

Данный датасет требуется скачать отдельно и разместить на своём жестком диске. Для наиболее удобного чтения из .tsv-файла было решено задействовать средства библиотеки pandas, а именно функцию read_csv(path, sep). В качестве параметра path требуется указать путь к файлу. В качестве сепаратора следует установить символ табуляции. После применения данной функции будет получен набор данных в виде переменной типа DataFrame.

Также стоит заметить, что значения колонки summary для некоторых записей могут быть пусты, в связи с чем в процессе предобработки данных можно столкнуться с различного рода исключениями. Дабы этого избежать, было принято решение продублировать для таких записей значение из колонки title в колонку summary.

Подключение и использование данного набора выглядит следующим образом:

Листинг 1. Пример работы с FiNeS

```
import pandas as pd

df = pd.read_csv("C:\dev/data.tsv", sep='\t')
pd.options.mode.chained_assignment = None
#Элемент предобработки. Заполняет Nan-ы
i = 0
while i < len(df['summary']):
```

```
if type(df['summary'][i]) == float:
        df['summary'][i] = df['title'][i]
i+=1
```

1.2. Дополнительный набор данных

Для того чтобы установить соответствие между сентимент-оценкой новости и изменением финансовых показателей потребовался вспомогательный набор данных. Вспомогательный набор содержит 527 записей. Некоторые данные записей были взяты из FiNeS и, в дальнейшем, были дополнены данными о стоимости ценных бумаг с Московской и СПБ бирж.

Вспомогательный набор данных имеет следующую структуру:

- Score – аналогичная одноименной из FiNeS;
- Published – дата публикации новости в формате YYYY-MM-DD;
- Tickers – тикер компании, о которой говорится в новости;
- Open – цена одной акции данной компании на момент открытия торгов;
- Close – цена одной акции данной компании на момент закрытия торгов.

В данный набор попали данные о стоимости ценных бумаг компаний, фигурировавших в новостях из FiNeS, с момента открытия и закрытия торгов. При этом собирались и анализировались данные вплоть до нескольких суток после выхода новости, что и позволило отследить общую динамику изменения курса на фоне вышедшей новости. Это и позволило, в дальнейшем, установить некоторые общие закономерности между изменением курса ценных бумаг и общей эмоциональной окраской новостей.

Также, потребовалось избежать неоднозначности и убрать из исходного набора данных, на данных которого и происходило формирование дополнительного набора данных, часть записей. А именно, были убраны

записи с несколькими тикерами компаний и записи, дату публикации которых, не удалось привести к формату YYYY-MM-DD.

Дополнительный набор данных представляет из себя .tsv-файл. Работа с ним производится аналогичным с FiNeS образом. Чтобы получить целевые данные достаточно лишь разность цены акции на момент завершения торгов и цены на момент открытия торгов поделить на цену с открытия торгов и перевести в проценты. Таким образом выглядит подключение дополнительного набора данных:

Листинг 2. Подключение дополнительного набора данных

```
import pandas as pd
from nltk.tokenize import RegexpTokenizer

df3 = pd.read_csv("C:\dev/data_fin.tsv", sep='\t')
i = 0
while i < len(df3['open']):
    print(i)
    open2 = float(df3['open'][i])
    close = float(df3['close'][i])
    df3['open'][i] = ( close - open2) / open2 * 100
    i+=1

X = df3[['score']].values
y = df3['open'].values
```

2. Примененные методы обработки текстовых данных

Этап предобработки текста — первый шаг, с которого начинается решение любой задачи в NLP. Текст без предварительной обработки — это зашумлённые данные, и работа с ними возможна, но может привести к снижению качества конечного результата. Однако, в целях повышения точности, было решено обучать модели на данных разной степени обработки, а именно – на данных предобработанных и на данных, не проходивших никакой обработки, помимо векторизации.

2.1. Токенизация

Токенизация – один из наиболее базовых методов первичной обработки текстовой информации. Токенизация подразумевает под собой разбиение всего текста на некоторые блоки – токены. В качестве токенов могут быть представлены, как и целые предложения из текста, так и отдельные слова. С первого взгляда может показаться, что сегментация текста на слова или предложения – тривиальная операция, которая может быть решена с помощью .split(‘. ’) или же какого-либо регулярного выражения, но некоторые блоки, схожие, с подстрокой “В т.ч. по причине...” способны доставить целый ряд проблем и сделать наивную реализацию алгоритма токенизации неэффективной. Также, в nltk, встроенный токенизатор способен отделять от слов частицу отрицания, что может помочь нам при дальнейшем анализе. Также он считает наименования цветов по типу “golden-red” за одно слово.

Токенизатор получает на вход строку и возвращает список (он же массив в языке Python) токенов, являющихся подстроками исходной строки.

Для осуществления предобработки данных следующим образом была применена токенизация по словам для данных из блоков summary и title:

Листинг 3. Применение токенизации

```
import nltk
from nltk.tokenize import RegexpTokenizer

          word_tokens = nltk.word_tokenize(df['summary'][i])
          word_tokens2 = nltk.word_tokenize(df['title'][i])
```

2.2. Удаление стоп-слов

Перед тем, как удалять стоп-слова следует разобраться в том, что же это такое. Стоп-слова, или же явно изобилующие в тексте слова и/или же слова, не несущие ни смысловой нагрузки, ни пользы при анализе текста. Их удаление является одним из базовых элементов первичной предобработки текста. Так для чего же их следует удалять? Это делается для повышения эффективности наших алгоритмов путём исключения лишних слов, которые не несут в себе никакой смысловой нагрузки и не могут являться чем-либо ценным при анализе текста. В качестве примера подобных слов можно указать различные слова-паразиты, связующие слова и слова-заполнители на подобие “Хммм”, “и” и тому подобные. Также, хорошим примеров подобных слов могут являться слова “Вроде”, “В общем”, а также некоторые бранные выражения, которые не несут и не могут нести в себе смысла при анализе текста. Однако, они могут являться носителями эмоционального окраса речи при сентимент-анализе текста.

Однако, решаемая задача не предполагает работы с подобными списками слов. Потому, достаточно удалять из текстовых данных стандартный перечень стоп-слов, уже встроенный в корпус библиотеки nltk.

В следующем фрагменте кода демонстрируется работа по удалению множества стоп-слов:

Листинг 4. Удаление стоп-слов

```
import nltk
from nltk.corpus import stopwords

     stop_words = set(stopwords.words('russian'))
```

```
i = 0
while i < len(df['summary']):
    word_tokens = nltk.word_tokenize(df['summary'][i])
    word_tokens2 = nltk.word_tokenize(df['title'][i])

    filtered_sentence = ''
    filtered_sentence2 = ''
    for w in word_tokens:
        if (w not in stop_words) and (w not in stop_list):
            filtered_sentence += ' ' + w
    df['summary'][i] = filtered_sentence
    for w in word_tokens2:
        if w not in stop_words:
            filtered_sentence2 += ' ' + w
    df['title'][i] = filtered_sentence2
    i+=1
```

Суть данного фрагмента кода в том, что алгоритм проходит по каждой записи в датафрейме и по каждому слову в каждой записи, перезаписывая в колонки title и summary те же строки, но не включая в эти строки стоп-слова.

2.3. Лемматизация слов с поищью pymorphy2

Также, одной из важных деталей предобработки текстовых данных может являться процесс лемматизации. Лемматизация применяется для взятия от слова его нормальной формы. Это также может помочь повысить точность работы модели, а также, в некоторых случаях уменьшить размерность словарей за счёт сокращения количества форм одного и того же слова.

В решении задачи по лемматизации слов во всех текстах можно задействовать библиотеку pymorphy2, которая способна обрабатывать слова на русском языке. Средствами данной библиотеки, для каждого слова

проводится морфологический анализ. Для этого в начале программы заводится один экземпляр класса MorphAnalyzer. Стоит обратить внимание на то, что каждый подобный экземпляр может занимать до 15Мб оперативной памяти, потому крайне желательно работать лишь с одним экземпляром.

В процессе предобработки текстовых данных было решено видоизменить код, представленный на прошлом листинге и добавить туда лемматизацию. Выглядит это следующим образом:

Листинг 5. Удаление стоп-слов и лемматизация

```
import nltk
from nltk.corpus import stopwords
import pymorphy2

    stop_words = set(stopwords.words('russian'))
    morph = pymorphy2.MorphAnalyzer()
    i = 0
    while i < len(df['summary']):
        word_tokens = nltk.word_tokenize(df['summary'][i])
        word_tokens2 = nltk.word_tokenize(df['title'][i])

        filtered_sentence = ''
        filtered_sentence2 = ''
        for w in word_tokens:
            if (w not in stop_words) and (w not in stop_list):
                p = morph.parse(w)[0]
                filtered_sentence += ' ' + p.normal_form
        df['summary'][i] = filtered_sentence
        for w in word_tokens2:
            if w not in stop_words:
                p = morph.parse(w)[0]
                filtered_sentence2 += ' ' + p.normal_form
        df['title'][i] = filtered_sentence2
        i+=1
```

2.4. Разбиение на тренировочную и тестовую выборки

В дальнейшем, для обучения первичных моделей потребовалось разбить исходные данные на 2 набора данных - тестовую и тренировочную выборки. Непосредственно обучение требует наличие тренировочной выборки данных. Набор обучающих данных должен быть заметно больше, нежели набор тестовых данных, которые нужны лишь для измерения точности обучения модели.

В решении данной задачи следует обратиться к библиотеке sklearn для языка программирования Python.

Листинг 6. Создание тренировочной и тестовой выборок

```
from sklearn.model_selection import train_test_split

    #Разбиение на тренировочный и тестовый наборы
    train, test = train_test_split(df, test_size=0.3, ran-
dom_state=42)
```

Среди указанных параметров можно заметить соответственно: датафрейм, который и требуется разбить; размер тестовой выборки; параметр, отвечающий за перемешивание данных в случайном порядке.

2.5. TF-IDF векторизация

Финальным же аккордом в предобработке текстовых данных можно считать представление их в виде, понятном для обучаемой модели, а именно - в числовом. И для решения этой задачи потребовалось прибегнуть к векторизации текстовых данных.

Самым простым способом представления текстовых данных в виде векторов является разреженная векторная модель, или, как её называют, «мешок слов». Её основная идея заключается в формировании матрицы, где столбцы соответствуют элементам словаря корпуса, а строки — документам.

Таким образом, каждый документ однозначно представлен некоторым вектором, что и является целью процесса извлечения признаков. Существует несколько модификаций данного подхода, определяющих метод расчёта значений векторов, и одним из наиболее применяемых является TF-IDF (Term Frequency – Inverted Document Frequency). В этой модели значения векторов зависят от информативности слова в рамках документа (TF) и специфичность слова в словаре корпуса (IDF).

Иными же словами, TF-IDF = TF(t, d) x IDF(t), где TF(t, d) - отношение числа вхождений некоторого слова к общему числу слов документа. Таким образом, оценивается важность слова t в пределах отдельного документа d. IDF(t) = инверсия частоты, с которой некоторое слово встречается в документах коллекции, равная логарифму отношения числа документов к числу документов из той же коллекции, в которых встречается слово t. Учёт IDF уменьшает вес широкоупотребительных слов.

В данной работе было решено создать 5 векторизаторов, на вход которым подаются различные данные. Первый векторизатор получает набор предобработанных данных из колонки Title датасета FiNeS. Второй обучается на обработанных данных из колонки Summary того же датасета. Третий и четвёртый, соответственно, получают необработанные данные из колонок Title и Summary. Пятый векторизатор обучается на объединении всех ранее перечисленных данных.

Листинг 7. Производимая векторизация текстовых данных

```
from sklearn.feature_extraction.text import TfidfVectorizer

content = []
for text in df['title']:
    content.append(text)
for text in df2['title']:
    content.append(text)
for text in df['summary'].values.astype('U'):
    content.append(text)
for text in df2['summary'].values.astype('U'):
    content.append(text)

#Разбиение на тренировочный и тестовый наборы
```

```
train, test = train_test_split(df, test_size=0.2, ran-
dom_state=42)
train2, test2 = train_test_split(df2, test_size=0.2, ran-
dom_state=42)

vectorizer = TfidfVectorizer()
vectorizer2 = TfidfVectorizer()
vectorizer3 = TfidfVectorizer()
vectorizer4 = TfidfVectorizer()
vectorizerU = TfidfVectorizer()

#Векторизация входных данных
vectorizer.fit(df['title'])
vectorizer2.fit_transform(df['summary'].values.astype('U'))

vectorizer3.fit(df2['title'])
vectorizer4.fit_transform(df2['summary'].values.astype('U'))
vectorizerU.fit_transform(content)
```

Полученные векторизованные данные хранятся в разреженной матрице SciPy, которая является эффективным способом хранения данных. Каждая строка в данной матрице соответствует документу, а каждый столбец каждому встреченному в коллекции документов слову, потому многие значения в матрице могут быть равны нулю.

3. Реализация системы прогнозирования

Одной из базовых задач в NLP является задача классификации текста. Текст возможно классифицировать по множеству разнообразных признаков и параметров на самые различные категории. В рамках темы данной работы, классификатор Байеса был применён для определения общего характера финансовой тенденции, заданной новостью. Тенденция может быть благоприятная или неблагоприятная.

Сам алгоритм является вероятностным и основан на теореме Байеса. Не смотря на то, что этот алгоритм является достаточно простым, он способен весьма эффективно обучаться. Для анализа влияния новостей из социальных сетей на финансовые показатели этого мало.

Финансовым показателем, оценка которого осуществляется в данной работе, был выбран процент, на который может уменьшиться или увеличиться стоимость ценной бумаги компании, фигурирующей в новости. Этапы решения основной цели представлены на следующем рисунке (рис. 1).

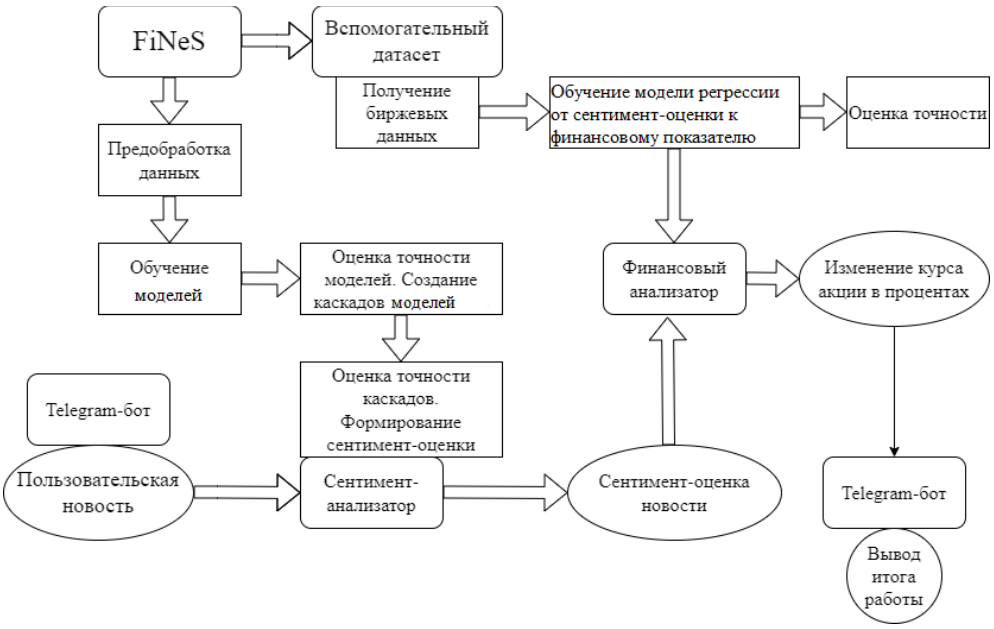


Рис. 1. Общая схема программы

Среди основных элементов схемы можно заметить следующие: FiNeS – основной набор данных; вспомогательный датасет – набор данных,

содержащий в себе биржевые данные; блок предобработки данных; блоки создания каскада регрессивных моделей для расчета сентимент-оценки новости; сентимент-анализатор; финансовый анализатор; Telegram-бот, являющийся удобным и простым интерфейсом для доступа к итоговой модели.

Данные элементы взаимодействуют следующим образом: данные из датасета FiNeS проходят необходимую предобработку. Далее, под различные обработанные данные формируются 12 регрессионных моделей, которые и были обучены на этих данных. Следующий шаг – оценка точности полученных моделей и формирование из 6 моделей с наибольшей точностью двух каскадов, из которых был сформирован третий. Далее, происходит оценка точности работы каскадов. Каскад, имеющий наименьший показатель ошибки, применён в качестве сентимент-анализатора. Также, на части данных из FiNeS был создан дополнительный набор данных. В него дополнительно включены финансовые данные с бирж. На его данных обучается финансовый анализатор. И, при получении от пользователя текстовых данных через Telegram-бота, данные последовательно проходят через сентимент-анализатор и через финансовый анализатор, после чего пользователь может запросить через того же бота итоговые данные.

3.1. Классификация по Байесу

Перед обучением классификатора Байеса необходимо осуществить отдельную предобработку данных. Так как классификатор Байеса оперирует вероятностями, нужно преобразовать весьма солидный массив данных в наборы слов с указанием частоты их возникновения в категориях благоприятных и неблагоприятных прогнозов.

Предобработка данных выглядит следующим образом:

Листинг 8. Пример предобработки данных для классификатора

```
import nltk

    i = 0
    all_words1 = []
```

```
    while i < len(df2['summary']):
        words =
nltk_tokenizer.tokenize(df2['summary'][i].lower())
        words2 =
nltk_tokenizer.tokenize(df2['title'][i].lower())
        for w in words:
            all_words1.append(w.lower())
        for w in words2:
            all_words1.append(w.lower())

        i+=1

    print('Слова посчитаны')

    # Получение частоты употребления для всех слов
    all_words1 = nltk.FreqDist(all_words1)

    all_words = {}

    for word in all_words1.keys():
        if not word in stop_words:
            p = morph.parse(word)[0]
            all_words[p.normal_form] = all_words1[word]

    print('Слова чет там')

    # взять первые 2000 наиболее значимых частых слов
word_features = list(all_words.keys())[:2000]
```

В данном коде сначала сохраняются все слова, встречающиеся в новостных текстах. Все заглавные буквы заменяются строчными, чтобы очистить словарь от одних и тех же вариантов одного и того же слова. После чего требуется получить частоту употребления всех слов с помощью функции `FreqDist()`. Далее, из полученного словаря удаляются стоп-слова, а сами же слова проходят через процесс лемматизации.

Далее требуется преобразовать слова в функции. Преобразование слов в функции выглядит следующим образом:

Листинг 9. Преобразование слов в функции

```
nltk_tokenizer = RegexpTokenizer(r'[а-яёа-z]+')
def find_features(document):
    words = []
    doc = nltk_tokenizer.tokenize(document.lower())
    for word in doc:
        if not word in stop_words:
           p = morph.parse(word)[0]
```

```
        words.append(p.normal_form)
    words = set(words)
    features = {}
    for word in word_features:
       features[word] = (word in words)
    return features
```

В дальнейшем, потребуется создать два набора данных. Первый набор является обучающей выборкой, а второй применяется при оценке точности обучения классификатора. Процесс создания двух наборов данных выглядит так:

Листинг 10. Создание тестовой и обучающей выборок

```
# взять первые 2000 наиболее значимых частых слов
word_features = list(all_words.keys())[:2000]

dataset1 = [] #title
dataset2 = [] #summary

i = 0
while i < len(df2['summary']):
    categ = 'pos'
    if df2['score'][i] < 0:
        categ = 'neg'

    text = df2['summary'][i]
    text2 = df2['title'][i]
    dataset2.append((find_features(text), categ))
    dataset1.append((find_features(text2), categ))

    i+=1
training_data = dataset1[:450]
test_data = dataset1[450:]
training_data2 = dataset2[:450]
test_data2 = dataset2[450:]
```

На данном этапе считается, что в наличии имеются все необходимые для обучения и для тестирования точности обучения классификатора данные.

Процесс обучения и применения классификатора имеет следующий вид:

Листинг 11. Обучение и применение классификатора Байеса

```
    classifier3 = nltk.NaiveBayesClassifier.train(training_data2
+ training_data)
```

```
      print((nltk.classify.accuracy(classifier, test_data +
test_data2))*100)

     def text_features(post):
         features = {}
         for word in nltk.word_tokenize(post):
             features[word.lower()] = True
          return features
      my_data = text_features("Илон Маск предсказал чудовищное
падение акций Google")
     print(classifier3.classify(my_data))
```

С этого момента данный классификатор считается обученным. Его точность на тестовой выборке составила около 67%.

Зачастую, процент точности классификатора Байеса при малом количестве категорий весьма высок и для крупной выборки данных способен составить от 69 до 87%, в зависимости от методов предобработки входных данных. Полученные результаты вполне закономерны на сравнительно небольшом исходном наборе данных.

3.2. Получение сентимент-оценки

Суть регрессии заключается в едином представлении данных для анализа и в установлении математической зависимости между входными независимыми параметрами и выходным зависимым значением. Регрессия может быть использована для прогнозирования значения выходного параметра при изменении входных параметров. Отталкиваясь от этого, применение регрессии для решения поставленной задачи целесообразно.

Линейная же регрессия является одной из самых простых и самых применяемых техник регрессии. И отличается простотой интерпретации результатов.

В целях повышения точности сентимент-анализа финансовых новостей из социальных сетей была реализована каскадная архитектура сентимент-анализатора. Благодаря представленным в предыдущем разделе текущей работы векторизаторам, удалось составить и обучить 12 различных моделей

регрессии. Данные модели обучены на различных данных с применением различных векторизаторов.

Общий список особенностей моделей представлен на схеме (рис. 2):

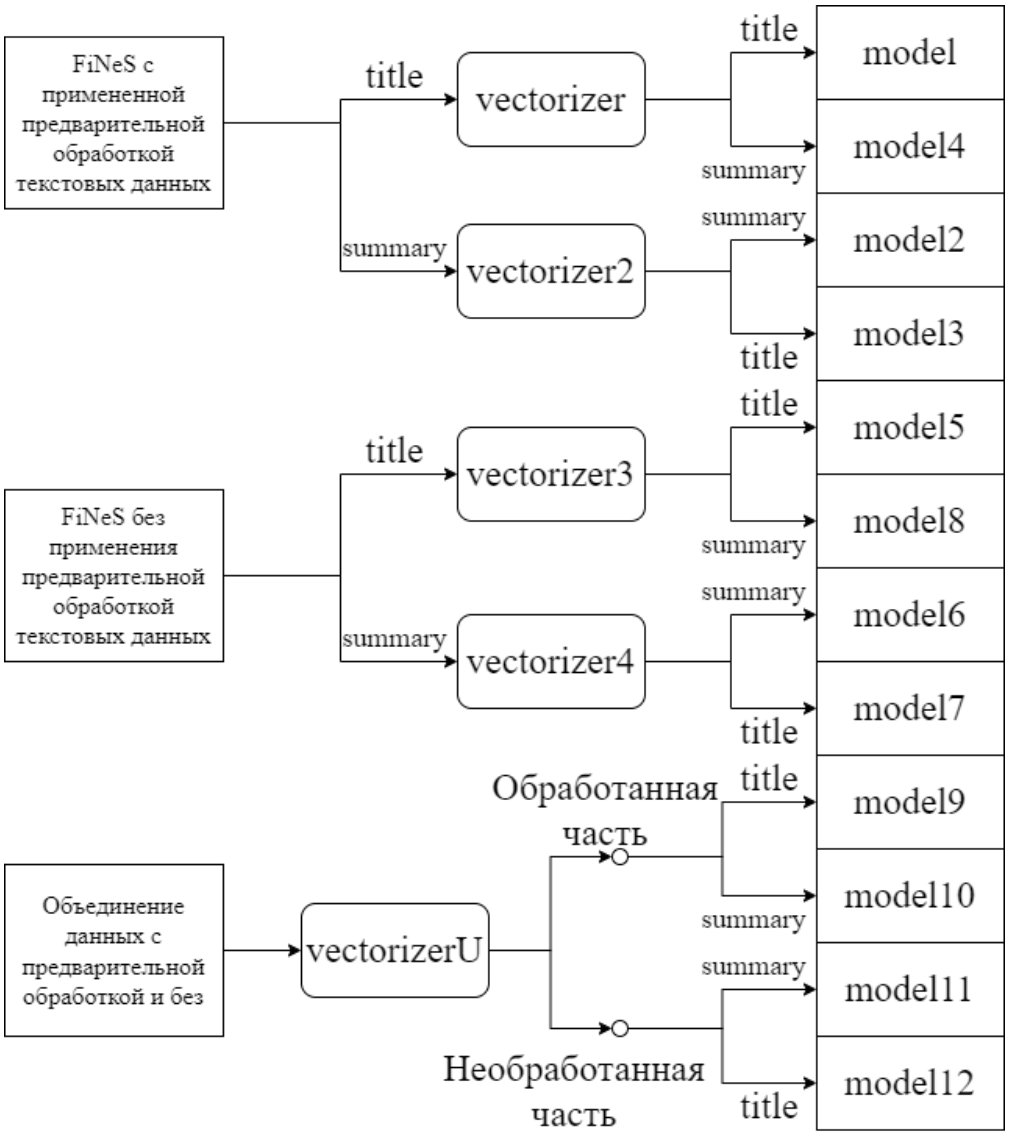


Рис. 2. Модели и их особенности

Применение данных моделей вслепую не является практичным решением, потому были реализованы методы оценки точности данных моделей, а также был реализован отбор тех моделей, которые отличились наименьшим значением ошибки. В качестве итоговой оценки точности работы модели была реализована и применена следующая оценка точности: $\frac{mse+mae}{2} * coef$. Она вычисляется как полу-сумма среднеквадратичной и средней абсолютной ошибок от анализа заголовков и полных текстов новостей из тестового множества, умноженная на коэффициент, который

обратно-пропорционален отношению количества верно подобранных по знаку оценок и количества записей в тестовой выборке. Данная оценка принята за итоговую, так как учитывает и количество верно спрогнозированных по знаку оценок, и отклонение от целевых значений. Похожая оценка точности была реализована и для каскадов сформированных моделей. Ознакомиться с функциями измерения точности моделей и каскадов можно в приложении 1.

Функция определения точности работы моделей способна рассчитывать, как и точность на данных из датасета FiNeS не прошедших предварительную обработку текста, так и точность на тех же данных с предварительной обработкой текстовых данных. На графике с рисунка 3 можно ознакомиться с визуализацией результатов подсчёта точности моделей (рис. 3).

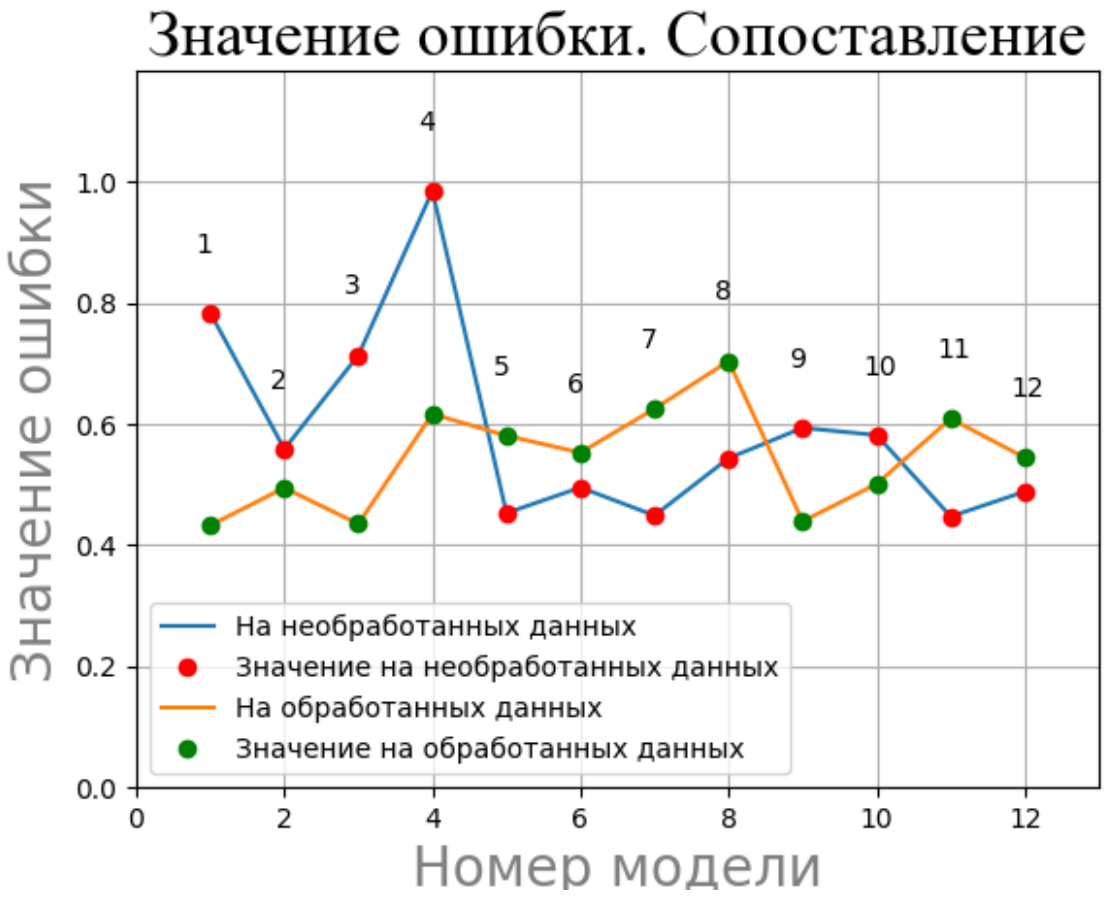


Рис. 3. Значения ошибки 12 моделей

Также, для удобства обращения к функции определения точности моделей было создано 12 функций-адаптеров следующего вида:

Листинг 12. Функция-адаптер для обращения к функции определения точности

```
def M1Acc():
    clear_pred_title =
model.predict(vectorizer.transform(test2['title']))
    proc_pred_title =
model.predict(vectorizer.transform(test['title']))
    clear_pred_sum =
model.predict(vectorizer.transform(test2['summary']))
    proc_pred_sum =
model.predict(vectorizer.transform(test['summary']))

        return   AccFinder(clear_pred_title,   proc_pred_title,
clear_pred_sum, proc_pred_sum)
```

Далее, в массив MetricsList были записаны значения для всех 12 моделей. После чего, было составлено и отсортировано по значению гибридной ошибки 2 отдельных массива кортежей. Каждый кортеж имел вид: (Гибридная ошибка, номер модели). Это потребовалось для формирования двух каскадов. Первый массив – это данные о работе моделей на необработанных данных. Второй – работа моделей на обработанных данных. Из каждого массива было взято и встроено в соответствующий каскад по 3 модели с наименьшей гибридной ошибкой. Данные действия потребовали реализации единого интерфейса обращения к моделям, который представлен на следующем листинге кода:

Листинг 13. Интерфейс для обращения к моделям

```
def GetModelNumRes(num, text, mode):
    txt = text
    if mode == 1:
        txt = Preproc_string(text)

    if num == 0:
        return model.predict(vectorizer.transform([txt]))
    elif num == 1:
        return model2.predict(vectorizer2.transform([txt]))
    elif num == 2:
        return model3.predict(vectorizer2.transform([txt]))
    elif num == 3:
```

```
        return model4.predict(vectorizer.transform([txt]))
    elif num == 4:
        return model5.predict(vectorizer3.transform([txt]))
    elif num == 5:
        return model6.predict(vectorizer4.transform([txt]))
    elif num == 6:
        return model7.predict(vectorizer4.transform([txt]))
    elif num == 7:
        return model8.predict(vectorizer3.transform([txt]))
    elif num == 8:
        return model9.predict(vectorizerU.transform([txt]))
    elif num == 9:
        return model10.predict(vectorizerU.transform([txt]))
    elif num == 10:
        return model11.predict(vectorizerU.transform([txt]))
    elif num == 11:
        return model12.predict(vectorizerU.transform([txt]))
    else:
        print('Ошибка. Интерфейс. Не тот номер')
         return 1
```

Данный интерфейс приспособлен для подачи, как и предварительно обработанных текстовых данных, так и необработанных.

Формирование каскадов выглядит следующим образом:

Листинг 14. Интерфейсы каскадов

```
#Значение по каскаду без обработок
def Get_Clear_Cascade(text):
    Acc, num = Clear_acc_list[0]
    res = GetModelNumRes(num, text, 0)
    Acc, num = Clear_acc_list[1]
    res += GetModelNumRes(num, text, 0)
    Acc, num = Clear_acc_list[2]
    res += GetModelNumRes(num, text, 0)
    return res/3

#Значение по каскаду с обработками
def Get_Proc_Cascade(text):
    Acc, num = Proc_acc_list[0]
    res = GetModelNumRes(num, text, 1)
    Acc, num = Proc_acc_list[1]
    res += GetModelNumRes(num, text, 1)
    Acc, num = Proc_acc_list[2]
    res += GetModelNumRes(num, text, 1)
    return res/3

#Получение значения по каскаду
def Get_Cascade(text):
         return             (Get_Clear_Cascade(text)             +
Get_Proc_Cascade(text))/2
```

Как можно заметить, в коде присутствует 3 каскада. Первый использует необработанные данные. Второй использует обработанные данные. Третий является объединением двух остальных.

Измерение точности работы каскадов моделей, в целом, аналогично измерению точности работы моделей. В коде это представлено следующим образом:

Графически, сравнение точности работы каскадов показано на следующем рисунке (рис. 4).

Согласно данным графика, наибольшую точность имеет третий каскад без дополнительной обработки данных, применённой к анализируемому тексту, однако это не говорит о бесполезности предварительной обработки данных, так как она была применена для подготовки обучающих данных некоторых моделей, вошедших в итоговый каскад.

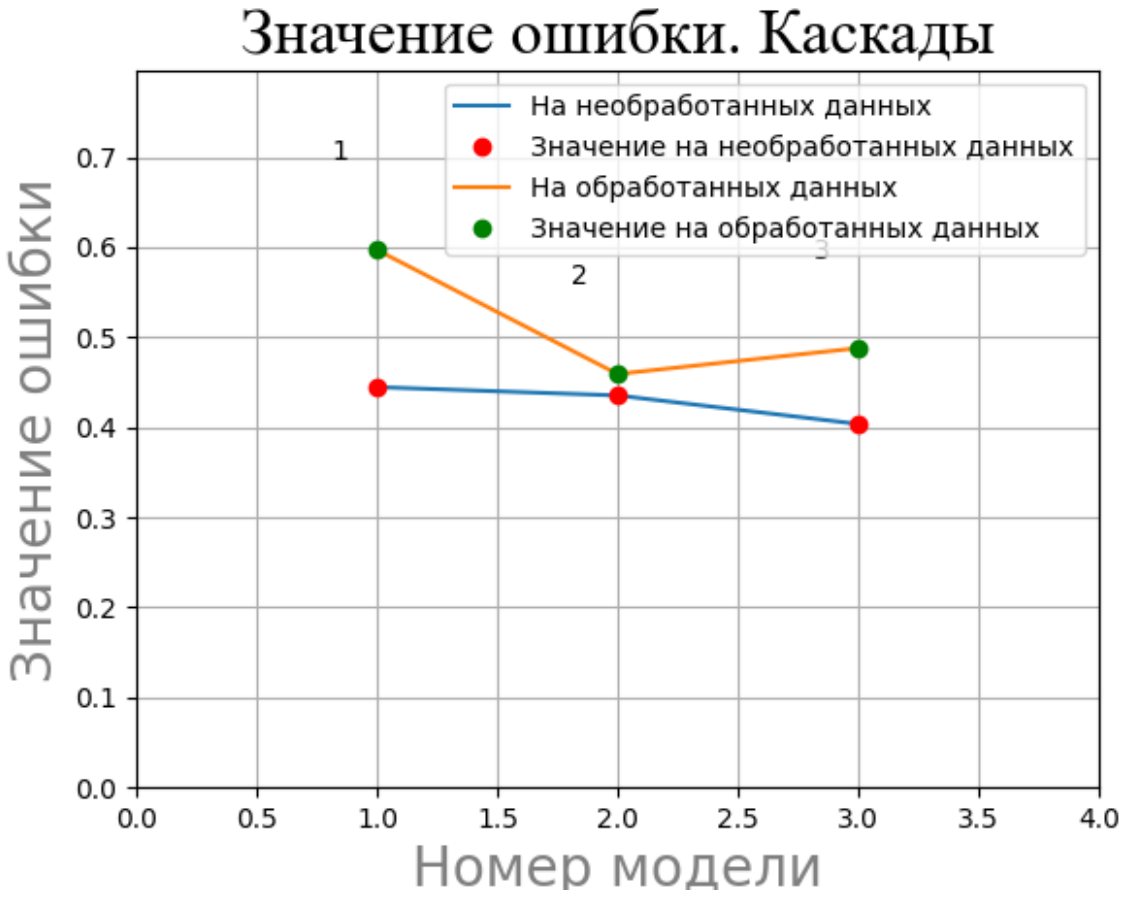


Рис. 4. Значения ошибки каскадов

Итоговое значение работы сентимент-анализатора определяется следующим образом:

Листинг 15. Интерфейс сентимент-анализатора

```
def GetBestSentiment(text):

    accdata, num = Cascade_accList[0]

    if num == 0:
        return Get_Clear_Cascade(text)
    elif num == 1:
        return Get_Clear_Cascade(Preproc_string(text))
    elif num == 2:
        return Get_Proc_Cascade(text)
    elif num == 3:
        return Get_Proc_Cascade(Preproc_string(text))
    elif num == 4:
        return Get_Cascade(text)
    elif num == 5:
        return Get_Cascade(Preproc_string(text))
    return 1
```

3.3. Прогнозирование изменения финансового показателя

В ходе сбора данных для дополнительного набора данных была осуществлена подготовка к реализации данного этапа. В данном этапе реализован финансовый анализатор, устанавливающий зависимость между сентимент-оценкой и процентом, на который изменится стоимость ценной бумаги некоторой компании, что может быть упомянута в новостях из социальных сетей.

Листинг 16. Финансовый анализатор

```
df3 = pd.read_csv("C:\dev/data_fin.tsv", sep='\t')
i = 0
while i < len(df3['open']):
    print(i)
    open2 = float(df3['open'][i])
    close = float(df3['close'][i])
    df3['open'][i] = ( close - open2) / open2 * 100
    i+=1

X = df3[['score']].values
y = df3['open'].values
```

```
modelSF = LinearRegression()
trX, teX, trY, teY = train_test_split(X, y, test_size=0.3,
random_state=42)

modelSF.fit(trX, trY)

def GetFin(text):
    return modelSF.predict([GetBestSentiment(text)])

def GetFinFromSent(sent):
    return modelSF.predict([sent])
```

Далее, была проведена оценка точности работы алгоритма с последующим графическим отображением. Ознакомиться с результатами можно на следующих графиках (рис. 5), (рис. 6), (рис. 7):

Рис. 5. Сравнение биржевых данных и данных, рассчитанных от заданного в датасете сентимента

Как можно заметить, алгоритм финансового анализатора оказался способен предугадывать общие экономические тенденции. Полученный алгоритм осуществляет оценку влияния новостей на финансовые показатели.

На следующем рисунке (рис. 5) можно ознакомиться со сравнением биржевых данных и тех данных, которые были получены благодаря последовательной тандемной работе сентимент-анализатора и финансового анализатора.

Рис. 6. Сравнение биржевых данных и данных, рассчитанных от рассчитанного в сентимент-анализаторе сентимента

На следующем рисунке (рис. 6) можно ознакомиться с финальным сравнением всех данных. А именно, биржевых, рассчитанных от заданного сентимента и рассчитанных от рассчитанного сентимента.

Рис. 7. Полное сопоставление всех рассчитанных данных и данных с бирж

Согласно всем полученным данным, можно сделать предварительный вывод о состоятельности реализованного алгоритма в области оценки влияния новостей на финансовые показатели.

4. Telegram чат-бот

Для предоставления удобного и простого интерфейса взаимодействия с полученными моделями был реализован Telegram чат-бот. Поскольку, со стороны пользователя предусматривается весьма простое взаимодействие с моделями, потребовалось изучение лишь общей структуры и функций бота и взаимодействие с BotFather-ботом Telegram. Для повышения удобства пользования было осуществлено добавление inline-кнопок.

После начала взаимодействия с ботом, бот предложит пользователю начать оценку новостного текста из социальной сети. После нажатия на кнопку “Начать” пользователь должен ввести свой новостной текст. Далее, бот предложит пользователю на выбор элементы своего функционала, среди которых есть:

- Классификация текста по Байесу по двум категориям: новость с благоприятным прогнозом и новость с неблагоприятным прогнозом;
- Схожая с предыдущей классификация по итогу работы сентимент-анализатора;
- Выдача сентимент-оценки;
- Общая важность данной новости на основе сентимент-оценки;
- Выдача прогноза изменения курса ценных бумаг в процентах, относительно текущего курса;
- Ввод новой новости для анализа.

Бот способен распознавать команды: /start, Привет, /help. При получении команды /start бот предложит начать анализ финансовой новости. С исходным кодом бота можно ознакомиться в приложении 2.

Блок расчета ответов бота выглядит следующим образом:

Листинг 17. Блок расчета ответов бота

```
def Result(text):
    res = []
    res.append(classifier2.classify(text_features(text)))
    gr = GetBestSentiment(text)[0]
    sent = 'pos'
```

```
if gr < 0: sent = 'neg'
res.append(sent) # По лин.регр
res.append(gr) # знач лин регр
gr2 = gr
gr3 = ''
if gr < 0: gr2 *=-1
if gr2 < 0.99: gr3 = 'Сверхзначительная'
if gr2 < 0.85: gr3 = 'Очень значительная'
if gr2 < 0.60: gr3 = 'Значительная'
if gr2 < 0.40: gr3 = 'Ощутимая'
if gr2 < 0.20: gr3 = 'Нормальная'
if gr2 < 0.10: gr3 = 'Низкая'
if gr2 < 0.05: gr3 = 'Малая'
if gr2 < 0.005: gr3 = 'Незначительная'
res.append(gr3)
gr2 = GetFin(text)[0]
res.append(gr2)
     return res
```

Заключение

В результате выполнения данной работы был реализован алгоритм, способный производить оценку влияния новостей в социальных сетях на финансовые показатели.

В ходе проделанной работы применён набор данных, подходящий для решения поставленной задачи. Был создан отдельный набор данных, в котором содержится информация с Московской и СПБ бирж.

Алгоритмически была реализована обработка входных данных для алгоритма. Была реализована простая классификация новостных текстов на две категории – финансово-благоприятные и финансово-неблагоприятные.

Была составлена, реализована, обучена и проверена модель, способная оценивать влияние новостей, как и из социальных сетей, так и из любого другого новостного источника, на финансовые показатели. В частности, на процентное изменение стоимости ценной бумаги той компании, о которой говорится в новостном источнике.

Было реализовано два основных компонента модели, а именно: сентимент-анализатор и финансовый анализатор. Первый – каскад из наиболее точных моделей, обученных на разных данных, цель которого заключается в даче наиболее точной сентимент-оценки. Второй – модель, которая находит зависимость между сентимент-оценкой и изменением в финансовых показателях.

Для обеспечения удобства пользования полученным алгоритмом, был создан Telegram чат-бот, имеющий весь необходимый функционал взаимодействия с моделью и с пользователем.

Литература

1. Вводное руководство по науке о данных Python nltk. – URL: https://russianblogs.com/article/87231561340/ (дата обращения 02.05.2022).
2. Репозиторий, являющийся источником для FiNeS. – URL: https://github.com/WebOfRussia/financial-news-sentiment/ (дата обращения 17.12.2022).
3. Статья “Основы Natural Language Processing для текста”. – URL: https://habr.com/ru/company/Voximplant/blog/446738/ (дата обращения 08.03.2022).
4. Статья по созданию Telegram-бота. – URL: https://habr.com/ru/articles/442800/ (дата обращения 14.03.2023).
5. Статья о модели линейной регрессии. – URL: https://habr.com/ru/articles/659415/ (дата обращения 14.03.2023).
6. Статья про инжиниринг признаков в машинном обучении. – URL: https://habr.com/ru/companies/otus/articles/681684/ (дата обращения 14.03.2023).
7. Техническая документация по библиотеке nltk. – URL: https://www.nltk.org (дата обращения 06.05.2022).

Приложение 1. Код функций, оценивающих точность моделей и точность каскадов моделей

```
#Подсчёт верно угаданных знаков
def CheckSigner(pred_list):
    i = 0
    cntr = 0
    for scr in test['score']:
        if (pred_list[i] * scr) > 0:
            cntr+=1
        i+=1

    return cntr


#Измерение точности моделей
def AccFinder(clear_pred_t, proc_pred_t, clear_pred_s,
proc_pred_s):

    test_len = len(test['title'])
    cs = (CheckSigner(clear_pred_t) +
CheckSigner(clear_pred_s))/2
#среднее число угаданных тональностей без предобраб
    cs2 = (CheckSigner(proc_pred_t) +
CheckSigner(proc_pred_s))/2
#среднее число угаданных тональностей с предобраб

    coef_clear = test_len / cs
    coef_proc = test_len / cs2

    clear_mse_err =(mean_squared_error(test2['score'],
clear_pred_t ) + mean_squared_error(test2['score'],
 clear_pred_s ))/2
    proc_mse_err =(mean_squared_error(test['score'],
proc_pred_t ) + mean_squared_error(test['score'],
proc_pred_s ))/2

    clear_mae_err =(mean_absolute_error(test2['score'],
clear_pred_t ) + mean_absolute_error(test2['score'],
clear_pred_s ))/2
    proc_mae_err =(mean_absolute_error(test['score'],
proc_pred_t ) + mean_absolute_error(test['score'],
proc_pred_s ))/2

    Avg_clear_err = (clear_mse_err + clear_mae_err)/2 *
coef_clear
    Avg_proc_err = (proc_mse_err + proc_mae_err)/2 *
coef_proc
```

```
    acc_res = [Avg_clear_err, Avg_proc_err, clear_mse_err,
clear_mae_err, proc_mse_err, proc_mae_err, 1/coef_clear,
1/coef_proc]
    # 0 - средняя ошибка без предобр, 1 - средн ошибка с
предобр. 2 - mse_clear, 3 - mae_clear, 4 - mse_proc, 5 -
mae_proc, 6 - coef_clear, 7 - coef_proc

    return acc_res



#Оценка точности каскада. Принимает функцию каскада и режим
оценки. Режимы - сырые данные и предобработанные
def CascadeAccFinder(Cascade, mode):
    texts1 = test2['title']
    texts2 = test2['summary']
    if mode == 1:
        texts1 = test['title']
        texts2 = test['summary']
    pred1 = []
    pred2 = []
    test_len = len(test['title'])
    for text in texts1:
        pred1.append(Cascade(text))
    for text in texts2:
        pred2.append(Cascade(text))

    cs = (CheckSigner(pred1) + CheckSigner(pred2))/2 #среднее
число угаданных тональностей
    coef = test_len / cs
    mse_err =(mean_squared_error(test2['score'], pred1 ) +
mean_squared_error(test2['score'], pred2 ))/2
    mae_err =(mean_absolute_error(test2['score'], pred1 ) +
mean_absolute_error(test2['score'], pred2 ))/2
    Avg_err = (mse_err + mae_err)/2 * coef

    acc_res = [Avg_err, mse_err, mae_err, 1/coef]
         return acc_res
```

Приложение 2. Код бота для мессенджера Telegram

Данное приложение содержит исходный код бота, используемого в итоговом проекте. Токен, необходимый для функционирования бота был изменён из соображений безопасности итогового проекта.

```
import telebot;
from telebot import types
bot = telebot.TeleBot('Token');


users = {}
user_text = {}
# 1 - гл. меню, 2 - набор новости, 3 - набор полной новости

@bot.message_handler(content_types=['text'])
def get_text_messages(message):
  try:
    if message.text == "Привет":
        users[message.from_user.id] = 1
        bot.send_message(message.from_user.id,
"Привет, чем я могу тебе помочь?")
    elif message.text == "/start":
        users[message.from_user.id] = 1
        bot.send_message(message.from_user.id,
"Здравстуйте, я могу провести анализ текста экономической новости из соцсети.")
        keyboard = types.InlineKeyboardMarkup();
#наша клавиатура
        key_yes = types.InlineKeyboardButton(text='Начать', callback_data='start'); #кнопка «Да»
        keyboard.add(key_yes);
#добавляем кнопку в клавиатуру
        bot.send_message(message.from_user.id, text='Начнём?', reply_markup=keyboard)
        print(message.from_user.id)
    elif message.text == "/help":
        bot.send_message(message.from_user.id,
"Напишите /start")
    elif users[message.from_user.id] == 3:
        #bot.send_message(message.from_user.id,
"Напишите /start")

        msg = message.text.split('\n')
        msg2 = ''
        i = 0
        while i < len(msg):
            if (msg[i] != ' ') and (msg[i] != ''):
```

```
                    msg2 += msg[i] + ' '
                i+=1

            regr_v = get_res(user_text[message.from_user.id])
            regr_v2 = get_res(msg2)

            with open('C:\dev/data2.tsv', newline='') as f:
                f.readline()
                reader = csv.reader(f, delimiter='\t')
                data = list(reader)

            output = []

            for line in data:
                if len(line) > 0:
                    title, summary, score = line
                    output.append([title, summary, score])

            with open('C:\dev/data2.tsv', 'w', newline='') as f:
                headers = ['title', 'summary', 'score']
                writer = csv.writer(f, delimiter='\t')
                writer.writerow(headers)
                writer.writerows(output)
                writer.writerow(
     [user_text[message.from_user.id], msg2,
      str((regr_v + regr_v2)/2)])

            keyboard = types.InlineKeyboardMarkup();
     #наша клавиатура
            key_yes =  types.InlineKeyboardButton(text='Начать',
callback_data='start'); #кнопка «Да»
            keyboard.add(key_yes);
     #добавляем кнопку в клавиатуру
            bot.send_message(message.from_user.id,
text='Начнём?', reply_markup=keyboard)


        elif users[message.from_user.id] == 2:
            user_text[message.from_user.id] = message.text

            keyboard = types.InlineKeyboardMarkup();
     #наша клавиатура
            key_yes = types.InlineKeyboardButton(
     text='По Байесу', callback_data='Beyes'); #кнопка «Да»
            keyboard.add(key_yes);
     #добавляем кнопку в клавиатуру
            key_yes = types.InlineKeyboardButton(
     text='По регрессии', callback_data='Reg'); #кнопка «Да»
            keyboard.add(key_yes);
     #добавляем кнопку в клавиатуру
            key_yes = types.InlineKeyboardButton(
     text='Значение регрессии', callback_data='Reg_val'); #кнопка
«Да»
```

```
            keyboard.add(key_yes);
     #добавляем кнопку в клавиатуру
            key_yes                                                    =
types.InlineKeyboardButton(text='Важность',
callback_data='Weight'); #кнопка «Да»
            keyboard.add(key_yes);
     #добавляем кнопку в клавиатуру
            key_yes                                                    =
types.InlineKeyboardButton(text='Показатель',
callback_data='Pok'); #кнопка «Да»
            keyboard.add(key_yes);
     #добавляем кнопку в клавиатуру

            key_yes = types.InlineKeyboardButton(
     text='Новый текст', callback_data='start'); #кнопка «Да»
            keyboard.add(key_yes);
     #добавляем кнопку в клавиатуру
            key_yes = types.InlineKeyboardButton(
     text='Ввести  полный  текст',  callback_data='New');  #кнопка
«Да»
            keyboard.add(key_yes);
     #добавляем кнопку в клавиатуру

            bot.send_message(message.from_user.id,
     text='Можно получить такие общие данные по тексту:',
      reply_markup=keyboard)
            #bot.send_message(message.from_user.id,
str(len(message.text)))
         else:
            bot.send_message(message.from_user.id,
      "Я тебя не понимаю. Напиши /help.")
       except Exception as e:
                print(e)
                time.sleep(4)


     @bot.callback_query_handler(func=lambda call: True)
     def callback_worker(call):
         if call.data == "start":
            try:
                print(call.message.chat.id)
                users[call.message.chat.id] = 2
                bot.send_message(call.message.chat.id,
      'Введите текст новости для анализа:');
            except Exception as e:
                print(e)
                time.sleep(4)
        elif call.data == "New":
            try:
             users[call.message.chat.id] = 3
             bot.send_message(call.message.chat.id,
      "Введите текст:");
            except Exception as e:
```

```
            print(e)
            time.sleep(4)
    elif call.data == "Beyes":
        try:
         rs = Result(user_text[call.message.chat.id])
         if rs[0] == 'pos':
            bot.send_message(call.message.chat.id,
"Прогноз благоприятный");
         else:
            bot.send_message(call.message.chat.id,
"Прогноз неблагоприятный");
        except Exception as e:
            print(e)
            time.sleep(4)
    elif call.data == "Reg":
        try:
         rs = Result(user_text[call.message.chat.id])
         if rs[1] == 'pos':
            bot.send_message(call.message.chat.id,
"Прогноз благоприятный");
         else:
            bot.send_message(call.message.chat.id,
"Прогноз неблагоприятный");
        except Exception as e:
            print(e)
            time.sleep(4)
    elif call.data == "Reg_val":
        try:
         rs = Result(user_text[call.message.chat.id])
         bot.send_message(call.message.chat.id,
"Значение регрессии: "+str(rs[2]));
        except Exception as e:
            print(e)
            time.sleep(4)
    elif call.data == "Weight":
        try:
         rs = Result(user_text[call.message.chat.id])
         bot.send_message(call.message.chat.id,
'Значимость новости: ' + str(rs[3]));
        except Exception as e:
            print(e)
            time.sleep(4)
    elif call.data == "Pok":
        try:
         rs = Result(user_text[call.message.chat.id])
         bot.send_message(call.message.chat.id,
'Ожидается падение акций на ' + str(rs[4]) + '%.');
        except Exception as e:
            print(e)
            time.sleep(4)


while True:
```

```
try:
    bot.polling(none_stop=True)

except Exception as e:
    print(e)

    time.sleep(4)
```