Федеральное государственное образовательное бюджетное учреждение высшего образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
Факультет «Информационные технологии и анализ больших данных»
Департамент анализа данных и машинного обучения

КУРСОВАЯ РАБОТА

на тему:

«Машинное обучение в задачах классификации текстов»

Выполнил:
студент гр. ПМ19-5
Борисов Дмитрий Сергеевич
Научный руководитель:
ассистент Шаталова А. Ю.

Москва – 2022

Содержание

ВВЕДЕНИЕ

С популяризацией компьютеров вопрос автоматизации ручной работы становится очень актуальным. На данный момент человечество сделало огромный прорыв к науке о компьютерах, так как теперь мы можем не просто разрабатывать алгоритмические программы. У нас есть возможность научить машину делать какие-либо предсказания. Такие открытия помогают делать методы машинного обучения, которые в последние десятилетия научились решать огромный пул задач. Одно из самых популярных направлений - анализ текстовой информации. В такой парадигме имеем ряд очень перспективных направлений: генерация текста, саммаризация текста, классификация текста. Последнее имеет множество вариаций, одна из них - это анализ тональности текста (sentiment analysis).

Анализ тональности - набор методов матчинга исходных объектов (текстов), которые предназначаются для автоматического определения эмоциональной окраски текстов. Обычно на язык математики это сводится к следующему - необходимо решить задачу классификации с тремя классами: negative, neutral, positive. Данный алгоритм применяется во многих сферах: анализ отзывов на объект, анализ фраз клиента при разговоре с чат-ботом, анализ комментариев в социальных сетях, антифрод задачи, определение спама, и многое другое.

Цели работы. Целью данной работы является построение модели классификации эмоциональной окраски текстов. Для хорошего конечного результата нашей модели, следует выдвинуть как можно больше гипотез, поэкспериментировать с разными алгоритмами ML и сравнить их по выбранным метрикам. Немаловажно провести анализ результов, понять интерпретацию итоговой модели.

Задачи работы.

1) Проанализировать датасет, сделать выводы по данным

2) Исследовать алгоритмы машинного обучения, реализовать больше трех алгоритмов для дальнейшего сравнения

3) Построить итоговую модель для классификации сентимента текстовых данных

4) Провести анализ полученных результатов, сделав выводы, которые будут основываться на метриках качества

Актуальность темы исследования. Каждый год число компаний, которые внедряют в свои системы различных чат-ботов, растет огромными темпами. Также в 2022 году доля пользователей интернетом в России приблизилась к 90 процентам от всего населения страны. Это означает, что социальные сети, мессенджеры берут вверх над обычными телефонными разговорами. Следовательно, фразы, которые говорят клиенты нужно каким-то образом фильтровать для самых разных нужд. Для этого нам на помощь приходят алгоритмы машинного обучения, а именно классификация текстов.

Объекты и предметы исследования - алгоритмы машинного обучения, исходный датасет.

Обработка и анализ данных будет производиться в среде Jupyter Notebook с использованием языка программирования Python версии 3.9.7 и встроенных библиотек numpy (для работы с массивами), seaborn, matplotlib (для работы с графиками), sklearn для ML, а также pytorch.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1. Основные термины машинного обучения

Алгоритмы машинного обучения в основном делятся на два вида: обучение с учителем (supervized learning), обучение без учителя (unsupervized learning). Основное различие в том, что при обучении с учителем у нас есть правильные ответы, на основе которых будем проверять качество работы нашей модели. К supervized learning относят классификацию и регрессию, а к unsupervized learning - кластеризацию и задачу понижения размерности.

Общая концепция следующая: Решение можно оформить как функцию, которая отображает независимые наблюдения в объекты предсказания (таргеты). Метод, отображающий X в y, называется моделью, а набор значений, которыми мы обладаем - обучающей выборкой. По обучающей выборке мы хотим построить модель, предсказания которой достаточно хороши. По тренировочной выборке мы желаем построить модель, предикты которой будут устраивать требования бизнеса. Данное качество предсказания фиксируется метриками качества.

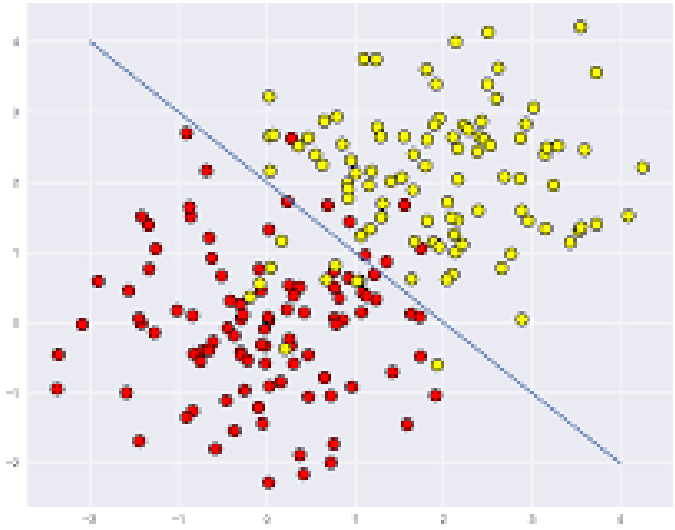
2. О задаче классификации

Это задача, где имеются независимые признаки, которые можно разделить на конечное число классов. Мы имеем обучающую выборку для которой мы знаем, к какому лейблу относится каждая запись. Остальная выборка без разметки. Необходимо сделать алгоритм, который может классифицировать любой объект из нашего множества.

Математическая постановка задачи. Пусть X - независимые переменные (feature), а Y - конечное множество классов. Существует отображение $y^* : X- > Y$, данные значения известны только на тренировочном датасе-

те $X^m = [(x_1, y_1), ..., (x_m, y_m)]$. Необходимо сделать модель $a : X- > Y$, который имеет возможность классифицировать любой объект $x \in X$.

Пример работы алгоритма в задаче бинарной классификации:



3. Векторное представление слов

Это один из самых важных моментов при работе с текстовой информацией. Как известно, методы машинного обучения не умеют работать не с числовыми данными, поэтому наша первостепенная задача - преобразавать наш корпус текстов в числовые векторы (эмбеддинги). Это можно делать разными способами, но я кратко опишу самые базовые и часто используемые:

1) Мешок слов (bag of words). Такой способ выделяет вектору весь документ, и каждый токен преобразуется в единичку по порядку токенов в словаре. Основные проблемы такого метода - 1) получаем большую разреженную матрицу; 2) не учитываем важность токенов (здесь все токены равнозначны, следующий метод помогает учитывать это)

2) TF-IDF. Как видно из названия, использует два значения: term frequency и inverse document frequency. Основная идея - будем считать, что слово важ-

но, если оно встречается в нашем документе, а в других не встречается. Это помогает бороться с самыми популярными стоп-словами - так как они будут встречаться во всех документах, то числовое значение у них будет маленькое (не важное слово). Такой же недостаток - слишком большая матрица. Такой метод будем использовать для обучения классических алгоритмов машинного обучения.

3) Word Embeddings. Это векторное представление слов. Главная идея - похожие слова будут находиться близко в нашем пространстве. Яркий пример - можно сложить два слова "король" и "женщина", получим слово "королева". Еще основное преимущество - векторы заданной длины (например, 300). Такой способ будем использовать для обучения нейронной сети.

4. Логистическая регрессия

Это один из самых популярных алгоритмов классификации. Основная идея - разделить данные на классы с помощью линейной функции. Пример работы можно увидеть на рисунке выше.

Главное отличие от простой регрессии - логистическая не предсказывает Y как числовую переменную, которая может принимать бесконечно много различных чисел. Наоборот, output'ом нашей функции будет служить вероятность принадлежности к определенному классу p, где $p \in [0, 1]$.

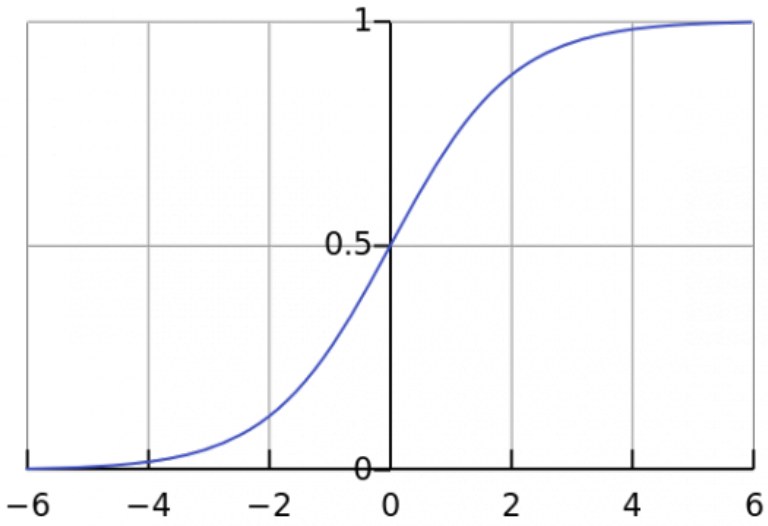
Как я уже сказал, модель пытается аппроксимировать линейной функцией. Но как это понять? В случае двумерного измерения - это линия, трех измерений - будет плоскость, и так по порядку. Измерения как правило зависят от количества независимых переменных. Если нам удалось поделить датасет с помощью линейной функции, то можно сказать, что точки линейно разделяемые.

Как можно понять разделение? Если имеем x_1, x_2, то функция, которая описывает границы будет такого вида $y = k_0 + k_1x_1 + k_2x_2$.

По сути после скалярного произведения вектора коэффициентов k на вектор переменных X, мы должны наложить нелинейную функцию активации, чтобы полученное число стало вероятностью принадлежности класса, которое лежит в промежутке от 0 до 1. Чаще всего такой функцией выступает сигмоида, которая отображает предикты как сигмовидную зависимость от сэмплов.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Функция имеет следующий вид:



Из графика четко видно, что если значения x большие, то y будет стремиться к 1, иначе - стремление к 0. Интересный момент возникает, когда вероятность 0.5, это говорит о неуверенности нашей модели: она не поняла, к какому из двух классов относить наше наблюдение.

5. Байесовский классификатор

Наивный байесовский классификатор является популярным алгоритмом классификации, в основном отражает точность наших данных. Он говорит, что переменные в нашем датасете не коррелируют между собой, и это его основная фича.

С помощью теоремы Байеса мы можем вычислить условную вероятность:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

где $P(A|B)$ - вероятность, что B принадлежит классу A (ее нужно рассчитать); $P(A)$ - независимая вероятность увидеть событие класса A среди всех; и так аналогично.

Данный классификатор берет в основу оценку апостериорного максимумама для определения вероятного класса.

$$c_{map} = argmax_{c \in C} \frac{P(B|A) * P(A)}{P(B)}$$

Следовательно нужно посчитать P для всех классов и выбрать с максимальной вероятностью. Также видно, что знаменатель является константной, поэтому его можно игнорировать

$$c_{map} = argmax_{c \in C} [P(B|A) * P(A)]$$

6. Метод опорных векторов

Такой вид классификации имеет множество приминений для разного рода задач. Основная идея - построить такую гиперплоскость, расстояние от которой до точек будет максимальным.

Метод опорных векторов определяет функцию для классификации следующего вида:

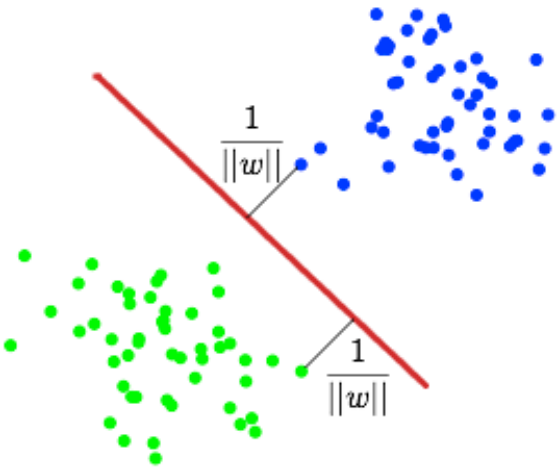
$$F = sign(< w, X > +b)$$

где $<, >$ - является скалярным произведением, w - вектор нормали к гиперплоскости, b - вспомогательный параметр.

Следовательно, те кейсы, для которых $F(X) = 1$ относят к одному

классу, $F(X) = -1$ - к другому. Почему такая логика? Потому что любая гиперплоскость определяется $< w, X > +b$ для каких-либо w, b.

Наглядная иллюстрация:



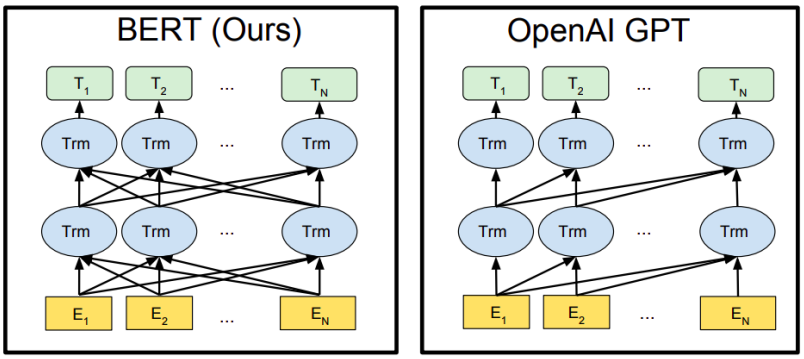
7. Архитектура BERT

Это нейросетевая модель от Google, которая является передовой в сфере обработки текстов. Она появилась не так давно, в 2018, и уже на основе нее работают многие движки компаний. Преимущество - данные модели уже были предобучены для разных языков (в том числе для русского) на огромных массивах, и любой человек может скачать такую архитектуру, чтобы не тратить свое время на обучение нейронной сети с нуля, так как это длительный процесс, также требующий больших вычислительных ресурсов.

Идея BERT: будем на вход нейронной сети отправлять предложения, некоторые слова в которых закрыты [MASK]. Теперь остается только обучить модель делать предсказания для этих закрытых слов.

Плюсы по сравнению с другими моделями - нейронная сеть является глубокой двунаправленной. Она не учитывает слова слева и справа, а потом

склеивает два аутпута. BERT смотрит в обе стороны. Сравнение с GPT, однонаправленная и двунаправленная сеть:



Мы берем готовую модель, и поверх нее добавляет еще один слой нейронов, специфичный конкертно для нашей задачи. Матрица весов w будет иметь случайные числа, и наша задача - найти минимум нашего лосса. Такой способ называется fine-tuning.

8. Аугментация текста

Аугментация - обогащение существующих данных для увеличения выборки. Данный подход может повысить качество исходной модели. Зачастую данные преобразовывают с помощью различных способ, не меняя при этом значение класса. Выделяют три базовых типа аугментации текстовых данных: замена (синонимы, сокращение, анализ окружения), перестановки (токенов, предложений), зашумление (ошибки в словах, изменение синтаксиса предложений). Опишу подробнее способы, которые я буду использовать в своей работе.

1) Замена синонимами. Это один из самых простых способов. Можно взять словарь синонимов и постепенно заменять слова. Однако стоит это делать осторожно, так как смысл предложения может сильно измениться, что приведет к изменению лейбла.

2) Эмбеддинги слов. Искать похожие векторы слов в пространстве. Это могут быть кейсы, которые встречают в похожих контекстах.

3) Обратный перевод. Идея - с помощью переводчика переводить слова на иностранный язык, а затем обратно на нужный нам. С помощью такого способа можем получить разные предложения, которые имеют одинаковый смысл.

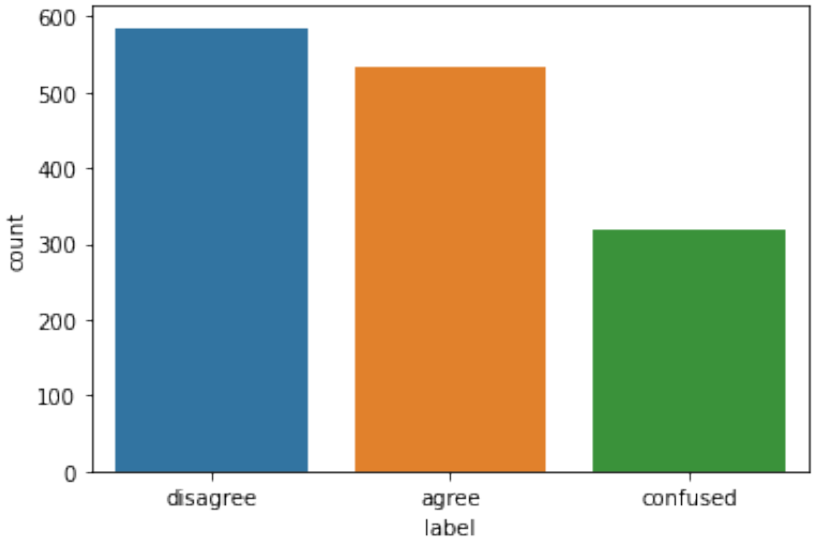
ПРАКТИЧЕСКАЯ ЧАСТЬ

9. Работа с данными и описание датасета

Описание датасета. В качестве текстовых данных я взял выборку ответов человека на вопросы звонящего ему робота, который предлагает подключить какую-либо услугу. Фразы размечены на три лейбла: agree, disagree, confused. Некоторые примеры:

```
agree: ну хорошо попробуем
disagree: не звоните мне больше
confused: что вы хотели
```

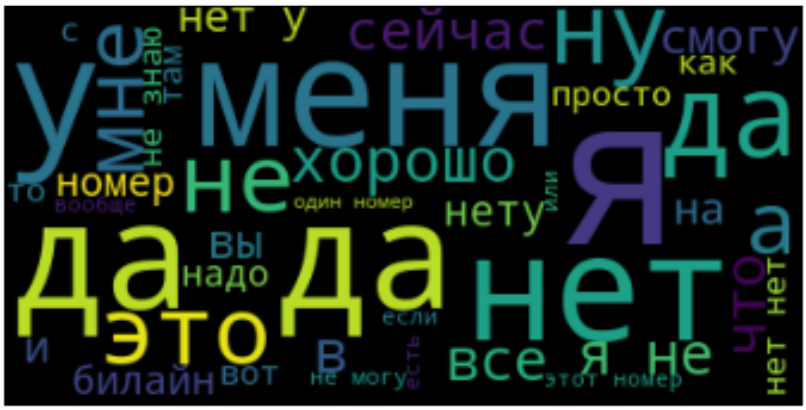
Распределение классов следующее: disagree - 585, agree - 532, confused - 320 записей. Также можно увидеть на диаграмме: Наблюдается небольшой дис-



баланс классов, но не критичный. Однако в дальнейшем в коде попробуем приметь ряд методов аугментации текстов, чтобы расширить нашу выборку для лучшего результата.

Сделаем минимальные преобразования текста: приведение к нижнему регистру, удаление запятых и других символов, редуцирование чисел. После таких манипуляций наши данные полностью готовы к обучению. Они не имеют пропущенных значений, нет пустых строк и строк, которые не соответсвуют логике нашей задачи.

Также интересно построить облако слов, чтобы понять, какие слова чаще всего встречаются в нашем датасете. Оставим топ 40 наиболее часто встречающихся слов:



10. Реализация моделей машинного обучения

Пайплайн работы, связанный с обучением моделей:

1) Получение векторного представления слов с помощью TF-IDF и предобученной модели DistilBert

2) Обучение классических моделей машинного обучения. А также обучение нейронной сети

3) Сравнение результатов по метрикам. Будем рассматривать метрики accuracy и f1 score. Почему их? Первая хорошо интерпретируема и будет понятна бизнесу (людям, которые далеки от математических формул). Вторая же объединяет две метрики (precision - точность и recall - полнота), на которые смотрят чаще всего.

4) Анализ по проделанной работе, выявление точек успеха, а также роста для дальнейших улучшений моих моделей.

10.1. Классические модели машинного обучения

Поделим наш датасет на train и test выборки в пропорции 0.7 и 0.3. Также добавим стратификацию по y, чтобы в выборках не было дисбаланса. Приведем данные к числовым векторам с помощью TF-IDF. Такая реализация есть в пакете sklearn, поэтому все можно запускать прямо из коробки. У такой реализации есть отличный параметр $ngram_range$, который отвечает за наши токены. Мы укажем следующий range: (2, 4). Это говорит о том, что минимальное количество слов в нашем токене равно 2. Обучим на тренировочных данных, затем трансформариуем тестовые. Пример кода:

```
df_train, df_test = train_test_split(yes_no, test_size=0.3,
random_state=RANDOM_SEED, shuffle=True, stratify=yes_no['label'])
text_transformer = TfidfVectorizer(ngram_range=(2, 4),       lowercase=True,
   max_features=150000)
X_train_text = text_transformer.fit_transform(df_train['text'])
X_test_text = text_transformer.transform(df_test['text'])
```

Логистическая регрессия. Отличный алгоритм для бейзлайн модели, который я предпочитаю использовать в своих наработках. Для начала попробуем обучить модель, не подбирая гиперпараметры, но используя кроссвалидацию:

```
lr_model = LogisticRegression(random_state=42, n_jobs=-1)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_results = cross_val_score(lr_model, X_train_text, df_train['label'],
   cv=skf, scoring='f1_micro')
cv_results.mean()
```

Значение метрики f1 micro в среднем равно 0.62, что уже очень неплохо. Однако если подобрать параметр регуляризации с помощью GridSearchCV, качество сразу становится 0.7 при C равным 4.

Байесовский классификатор. Проделаем аналогичные действия с классификатором Байеса, который возьмем так же из библиотеки sklearn.

```
mnb_model = MultinomialNB()
cv_results = cross_val_score(mnb_model, X_train_text, df_train['label'],
   cv=skf, scoring='f1_micro')
```

```
cv_results.mean()
```

Качество также около 0.62.

Метод опорных векторов.

```
svm_model = SVC()
cv_results = cross_val_score(svm_model, X_train_text, df_train['label'],
   cv=skf, scoring='f1_micro')
cv_results.mean()
```

Качество этого алгоритма из коробки хуже всех моделей - 0.57, но с помощью того же GridSearchCV я подобрал параметры и качество поднялось до > 0.7:

```
C=1.0,
kernel='linear',
degree=3,
gamma='auto'
```

Основной вывод по классическим моделям:

Модели дают неплохое качество на кроссвалидации, но это связано с тем, что довольно лояльный и хорошо размеченный датасет, поэтому кажется, что если кейсы будут не совпадать с нашей выборкой, качество будет сильно ниже. Алгоритмы обрабатываются моментально по времени. Это связано со всеми классическими моделями, а также с методом векторизации слов, ведь если нужно будет предсказать слово, которого нет в нашем словаре, модель не справится. Для нивелирование данных проблем я и решил использовать векторизацию слов с помощью BERT.

10.2. Нейронная сеть

Возьмем предобученную модель ребят из DeepPavlov $DeepPavlov/distilruber tiny-cased-conversational$. Модель обучалась на популярных русскоязычных текстовых источниках: "OpenSubtitles, Dirty, Pikabu, and a Social Media segment of Taiga corpus". Данная реализация обучалась 80 часов на видеокартах 8 nVIDIA Tesla P100-SXM2.0 16Gb. Я решил взять данную модель, так

как она не полносвязная и довольно легко и быстро обучается, давая неплохие результаты. Для начала загрузим токенайзер:

```
PRE_TRAINED_MODEL_NAME = 'DeepPavlov/distilrubert-tiny-cased-conversational'
tokenizer = DistilBertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

Весь код можно будет найти на моем github, здесь я опишу главные шаги:

1) Подготовим классы для преобразования наших данных в pytorch формат DataLoader. Берем длину вектора 512, размер батча 16, как было описано в оригинальной статье.

2) Добавим mean pooled слой в классе YesNoClassifier

3) В качестве оптимизатора AdamW, функции ошибки - кросс энтропия.

4) Запустим процесс обучения (3 эпохи, тоже из статьи). И видно, что уже на второй эпохе хорошее качество (низкая ошибка). Также заметим, что ошибка одинаковая на трейн и тест данных, это говорит нам о том, что модель не переобучилась. Сохраним в файл .bin нашу лучшую модель.

Главный итог. Основное преимущество такого подхода в том, что модель обучена на огромном корпусе текстов, что помогает модели быть максимально гибкой при виде новых кейсов. Время обучение составило - примерно 51 минута. Classification report:

	precision	recall	f1-score	support
disagree	0.93	0.93	0.93	585
confused	0.88	0.92	0.90	320
agree	0.94	0.91	0.93	532
accuracy			0.92	1437
macro avg	0.92	0.92	0.92	1437
weighted avg	0.92	0.92	0.92	1437

11. Заключение

В данной курсовой работе были рассмотрены три метода классического машинного обучения и нейронная сеть для задачи бинарной классификации. Я описал краткую теорию по каждому методу, объяснил, как работает, для чего нужна векторизация слов в контексте нашей задачи. Также затронул теорию про алгоритм BERT, а еще описал свои наработки по задаче аугментации текста, которые я применил в своем jupyter ноутбуке, ссылка на который будет в приложении.

В практической части были реализованы описанные выше методы на реальном датасете из сферы телекома. Хорошее качество показали логистическая регрессия с подобранным коэффициентом регуляризации и метод опорных векторов с подобранными гиперпараметрами. Метрика f1 на кроссвалидации была около 0.7. Результаты нейронной сети оказались еще лучше. И главный плюс нейронной сети, который я для себя выделил - это способность к обобщению данных, гибкая работа с неизвестными текстами. В коде можно будет увидеть функцию, которая может отнести любую переданную строку к одному из трех классов: agree, disagree, confused.

Следует заметить, что задачи NLP сейчас пользуются огромной популярностью, потому что решают большой пул современных задач. Это говорит о том, что машину можно научить понимать хорошо человеческий язык, следовательно, можно автоматизировать какие-либо ручные процессы, например, звонки оперторов клиентам с просьбой подключить какую-либо услугу. Исходя из этого, можно сделать вывод, что и актуальность данной работы не теряется, так как обзорная статья будет полезна для начинающих специалистов в области анализа данных.

12. Источники

Список литературы

Alina Kolesnikova, Yuri Kuratov, Vasily Konovalov, and Mikhail Burtsev. Knowledge distillation of russian language models with reduction of vocabulary, 2022. URL https://arxiv.org/abs/2205.02340.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

Kowsari, Jafari Meimandi, Heidarysafa, Mendu, Barnes, and Brown. Text classification algorithms: A survey. Information, 10(4):150, apr 2019. doi: 10.3390/info10040150. URL https://doi.org/10.3390%2Finfo10040150.

Ying Chen, Peng Liu, and Chung Piaw Teo. Regularised text logistic regression: Key word detection and sentiment classification for online reviews, 2020. URL https://arxiv.org/abs/2009.04591.

Учебник Яндекса. Машинное обучение и data science: погружение в тему. URL https://academy.yandex.ru/dataschool/book.

13. Приложение

https://github.com/dsborisovv/coursework-text-classification

```
# Алгоритм обучения нейронной сети
%%time

history = defaultdict(list)
best_accuracy = 0
best_loss = 10

for epoch in range(EPOCHS):

    print(f'Epoch {epoch + 1}/{EPOCHS}')
    print('-' * 10)

    train_acc, train_loss = train_epoch(model, train_data_loader, loss_fn,
   optimizer, device, scheduler, len(df_train))

    print(f'Train loss {train_loss}, accuracy {train_acc}')

    val_acc, val_loss = eval_model(model, val_data_loader, loss_fn, device,
   len(df_test))

    print(f'Val loss {val_loss}, accuracy {val_acc}')

    history['train_acc'].append(train_acc)
    history['train_loss'].append(train_loss)
    history['val_acc'].append(val_acc)
    history['val_loss'].append(val_loss)

    if val_acc > best_accuracy and val_loss < best_loss:
        torch.save(model.state_dict(), 'best_model_state.bin')
        best_accuracy = val_acc
        best_loss = val_loss
```