### Квалификационная работа Мурашка А.В.

### AML-32

### по курсу «Машинное обучение: фундаментальные инструменты и практики»

## Тема:

## Анализ тональности текста. Выделение морфологических сущностей на основе топологической близости.

### Почему выбрана эта тема диплома ?

Актуальных и интересных для меня тем с готовыми датасетами, не требующими отдельного исследования, на данный момент у меня нет. Поэтому я выбрал классическую тему, востребованную практически в любой отрасли человеческой деятельности, где обратная связь с целевой аудиторией является значимым фактором. (Практически любая деятельность, за исключением поиска внеземного разума).

Эта тема позволила мне еще раз изучить и сравнить различные подходы к ее реализации (и разные инструментальные техники ML), чтобы выбрать оптимальные для использования в дальнейшей работе. После решения классической задачи я попробовал расширить исследование, и попытался извлечь из текстов не только интонацию, но и отдельные сущности и отношение к ним через выделение наиболее частотных существительных и глаголов и их взаимных отношений. Эта мысль пришла в голову после анализа структуры выбранного датасета. В него входит набор отзывов на более чем 30 слабо связанных между собой “сущностей”, по каждой из которых выборка не является настолько большой, чтобы на ней можно было применить полноценное обучение нейронной сети. И мне представляется, что это типичная ситуация для большинства объектов/субъектов интернета.

### Данные

Датасет для работы взят из репозитория kaggle.com из задания:

Twitter Sentiment Analysis

https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis

представляет собой вручную размеченную коллекцию твитов.

Таблица имеет следующие поля:

TweetID - id твита   
 entity - сущность, которой твит посвящен   
 sentiment - целевая переменная - метка настроения   
 Tweet content - содержание твита

Целевая переменная sentiment имеет 4 значения:

Positive - позитивный отзыв   
 Neutral - нейтральный отзыв   
 Negative - негативный отзыв   
 Irrelevant - ошибочный отзыв (не по теме)

### Исходные тексты программ:

eda\_vectorizing\_v2\_dipl\_sent\_anlz..ipynb — содержит первоначальную очистку данных, EDA, предобработку текста и сравнительный анализ результатов классификации логистической регрессии при простой токенизации и векторизации TfidfVectorizer, при дополнительной лемматизации и векторизации TfidfVectorizer и при лемматизации и векторизации на предобученной модели word2vec.

grid\_v2\_dipl\_sent\_anlz.ipynb - содержит эксперименты по улучшению метрик на основе классических подходов ML, таких как GridSearchCV и RandomizedSearchCV на коллекции из разных моделей классификации с перебором параметров, с целью нахождения оптимальной модели при оптимальных параметрах.

BERT\_SVM\_v3\_dipl\_sent\_anlz.ipynb - содержит эксперимент по применению векторизации (выделению признаков) с использованием наиболее современной предобученной модели BERT с последующим обучением выбранной на предыдущем этапе модели SVM с целью улучшения результата.

entity\_ner\_v2\_dipl\_sent\_anlz.ipynb — эксперимент по построению аналитической системы по выделению отдельных сущностей из датасета, с целью нахождения в них конкретных подчиненных сущностей и отношения к ним, через морфологический анализ, метод частотного выделения и анализ косинусной близости.

eda\_vectorizing\_v2\_dipl\_sent\_anlz.ipynb

https://drive.google.com/file/d/13EJpOtEOIic\_4MeEWwR2XkhzDyNJYruD/view?usp=sharing

grid\_v2\_dipl\_sent\_anlz.ipynb

https://drive.google.com/file/d/1V1vWzYfDKKO3myqrAkl-Sv6wkzV1hOGB/view?usp=sharing

BERT\_SVM\_v4\_dipl\_sent\_anlz.ipynb

https://colab.research.google.com/drive/18mQldsJ8GMLrzSHVgEWeDx\_B7al-AICr?usp=sharing

entity\_ner\_v2\_dipl\_sent\_anlz.ipynb

https://drive.google.com/file/d/12H7O1TVab3mk6AuJhTrfWvAhFlWAobEB/view?usp=sharing

### Используемые метрики

Для оценки результатов классификации будут использоваться следующие метрики:

accuracy (точность) измеряет, как часто модель в среднем дает правильный прогноз. Эта метрика рассчитывается следующим образом:

accuracy: (TP + TN) / (все прогнозы).

precision (точность) также известна как положительное прогнозное значение. Она измеряет вероятность того, что модель верно спрогнозировала, что значение является истинным.

Эта метрика рассчитывается следующим образом: TP / (TP + FP).



recall (полнота) – это метрика, которая оценивает способность модели обнаруживать все положительные примеры в данных. Эту метрику можно использовать для оценки производительности модели классификации, особенно когда ложно отрицательные результаты имеют больший вес, чем ложно положительные.



Оценка F1 (F-мера) – среднее значение Точности измерений (Accuracy) и Отзыва (Recall) с Весами (Weight) при наличии. F1 обычно более полезна, чем точность измерений, особенно если распределение классов неравномерно.

F1 = 2 \* (precision \* recall) / (precision + recall)



Для быстрой визуальной оценки используется Confusion Matrix.

Матрица неточностей — это таблица или диаграмма, показывающая точность прогнозирования классификатора в отношении двух и более классов. Прогнозы классификатора находятся на оси X, а результат (точность) — на оси Y. Ячейки таблицы заполняются количеством прогнозов классификатора. Правильные прогнозы идут по диагонали от верхнего левого угла в нижний правый.

## Первая часть работы: исследуем разные подходы к предобработке текста и их влияние на результат классификации.

### Очистка данных

Тренировочную выборку почистили от пропусков и дубликатов. Проверочная выборка проблем не имела. Заменили класс 'Irrelevant', высказываний не по теме на 'Neutral', т.к. по смыслу задания это на результат не влияет, но данные упрощает - 3 класса вместо 4, и очищает, т.к. "не по теме" очень размытый критерий.

После токенизации и лемматизации текстов твитов образовалось дополнительно 1458 пустых строк за счет междометий и прочих ошибочных символов, к которым не нашлось стандартной формы. Их я тоже удалил.

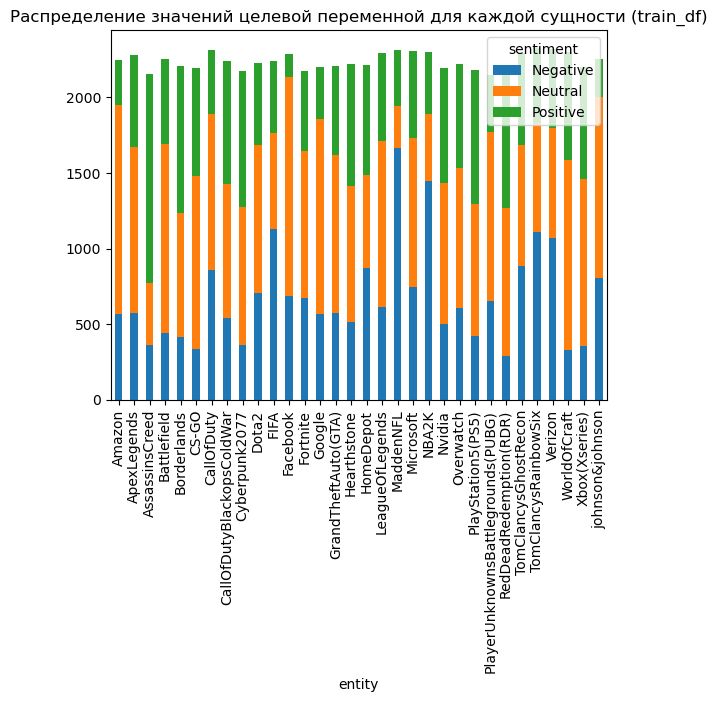
Изначально в тренировочной таблица train\_df имела 74681 строк. После очистки записей осталось 71655.

В валидационной таблице val\_df было 999, осталось 998 строк.

### **Описательная статистика, поиск взаимосвязей**

### 

Большой разбалансированности целевой переменной не наблюдается. Дополнительных мероприятий не требуется.



В значениях entity имеем 32 достаточно разноплановые сущности. Здесь и порталы и бренды и игры и приставки и производители софта и оборудования. Довольно странный набор.

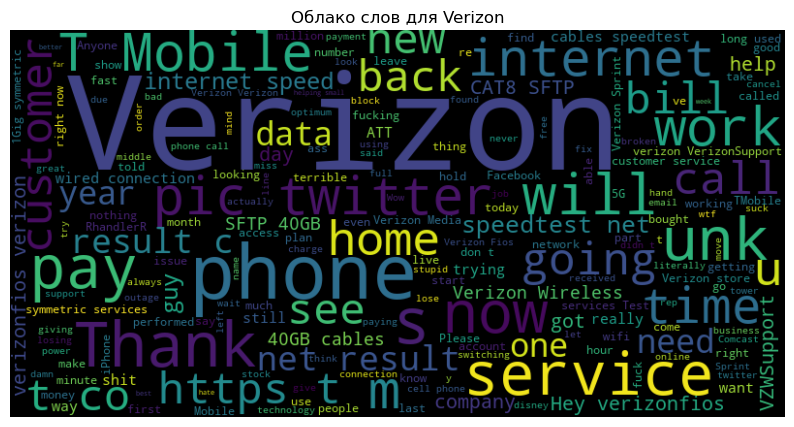
Длина твитов сильно варьируется.

Количество твитов для каждой сущности чуть больше 2000. Это не очень много. Для качественного обучения нейронной сети с нуля этих данных мало. Зато видно, что за нас постарались выровнять данные по всем сущностям по количеству твитов. У всех entity примерно равное количество твитов.

Также визуально посмотрел на наиболее частотные слова для разных настроений. Для позитивного настроения например:



Визуально посмотрел на наиболее частотные слова для разных сущностей. Для Verizon например:



Много каши и шума, т.к. слишком много разноплановых сущностей со своими частотными особенностями, и нужно что то в этих сущностях понимать чтобы оценивать адекватность представленного вывода. В целом структура данных понятна и неприятностями не угрожает.

### Предобработка и токенизация

Для предобработки стоп слов, пунктуации и токенизации я воспользовался утилитами пакета nltk.

Для кодирования текстовых меток целевой переменной числами я воспользовался утилитой LabelEncoder из пакета sklearn.preprocessing.

### 1 подход к классификации: Преобразование текста в векторное пространство методом Tfidf и классификация логистической регрессией.

Для преобразование текста в векторное пространство (здесь и в других аналогичных случаях) использовался метод Tfidf из библиотеки sklearn.

{tf-idf}(t,d,D)= {tf}(t,d)х{idf}(t,D)

Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

Для разделения данных на обучающую и тестовую выборки здесь и в других аналогичных случаях использовалась функция train\_test\_split из библиотеки sklearn. Перемешивание данных из датасета не нарушает исследуемую структуру данных.

Для классификации использовалась LogisticRegression из библиотеки sklearn с параметрами по умолчанию.

Были достигнуты следующие результаты:

precision recall f1-score support

Negative 0.89 0.92 0.91 266

Neutral 0.91 0.91 0.91 456

Positive 0.91 0.88 0.89 277

accuracy 0.91 999

macro avg 0.91 0.91 0.91 999

weighted avg 0.91 0.91 0.91 999

Accuracy: 0.9069069069069069

### 2 подход к классификации: Преобразование лемматизированного текста в векторное пространство методом Tfidf и классификация логистической регрессией.

Во втором случае я произвел лемматизацию текста перед векторизацией. Лемматизация производилась функцией WordNetLemmatizer из библиотеки nltk.

Для классификации использовалась LogisticRegression с параметрами по умолчанию.

Были достигнуты следующие результаты:

precision recall f1-score support

Negative 0.89 0.91 0.90 266

Neutral 0.91 0.92 0.91 456

Positive 0.91 0.88 0.89 277

accuracy 0.90 999

macro avg 0.90 0.90 0.90 999

weighted avg 0.90 0.90 0.90 999

Accuracy: 0.9039039039039038

Результат оказался несколько хуже чем, до лемматизации.

Видимо, приведя к стандартной форме слова, мы убрали некоторую детализацию эмоциональной окраски за счет крепких устойчивых устных выражений и тому подобному.

Следующим этапом я решил посмотреть, может ли применение более продвинутой векторизации word2vec по предробученной модели от Google (GoogleNews-vectors-negative300) улучшить этот результат.

### 3 подход к классификации: Преобразование лемматизированного текста в векторное пространство методом word2vec и классификация логистической регрессией.

Для векторизации word2vec использовался пакет gensim и предварительно обученная модель вектора слов Word2vec (GoogleNews-vectors-negative300) для корпуса новостей Google (3 миллиарда слов) (3 миллиона 300-мерных векторов английских слов).

Для классификации использовалась LogisticRegression с параметрами по умолчанию.

Были достигнуты следующие результаты:

precision recall f1-score support

Negative 0.61 0.59 0.60 266

Neutral 0.63 0.69 0.66 456

Positive 0.62 0.54 0.58 277

accuracy 0.62 999

macro avg 0.62 0.61 0.61 999

weighted avg 0.62 0.62 0.62 999

Accuracy: 0.6236236236236237

Полученный результат можно назвать катастрофическим.

Точность предсказаний еле дотягивает до 60 процентов. Особенно много позитивных и негативных отзывов было приписано нейтральному контексту.

Сначала мы снизили точность прогноза убрав некоторые особенности текстов лемматизацией, а потом векторизовали слова в этих лемматизированных текстах с помощью предобученной модели wor2vec от Google. Эта модель была обучена на очень большом корпусе новостей, но их тематика слишком далеко лежит от тематики наших твитов и векторизация перестала отвечать контексту. В результате точность предсказаний логистической регрессии кардинально снизилась.

### Предварительный вывод по первой части работы

Можно сделать вывод, что для наших целей лучше подходит использование токенизатора nltk.tokenize и векторизатора TfidfVectorizer. Так как TfidfVectorizer работает именно с корпусом наших документов, результат его работы более актуален для задач классификации. К тому же он проще и скорость обработки таким образом выше.

Во второй части работы я попробую поднять метрику классификации за счет использования разных классификаторов с разными вариациями параметров.

## Вторая часть работы: Выбор оптимального классификатора и оптимальных параметров с использованием оптимизирующих техник машинного обучения.

Используя для обработки текстов токенизатор nltk.tokenize и векторизатор TfidfVectorizer

подберем лучшую модель классификатора и ее параметры методами GridSearchCV и RandomizedSearchCV.

### Применение метода GridSearchCV к логистической регрессии

Использовалась функция GridSearchCV из библиотеки sklearn. Параметры логистической регрессии изменялись в диапазонах:

param\_grid = {'C':[0.1,0.2,0.3,0.5,0.7,1],

'penalty': ['l1', 'l2'],

'solver': ['liblinear', 'saga']

Лучший результат был достигнут при параметрах:

mean\_test\_score: 0.8192239892128583

{'C': 1, 'penalty': 'l2', 'solver': 'saga'}

LogisticRegression(C=1, solver='saga')

Логистическая регрессия при этих параметрах показала следующие результаты:

precision recall f1-score support

Negative 0.89 0.94 0.91 265

Neutral 0.93 0.91 0.92 456

Positive 0.90 0.88 0.89 277

accuracy 0.91 998

macro avg 0.91 0.91 0.91 998

weighted avg 0.91 0.91 0.91 998

Accuracy: 0.908817635270541

Улучшить удалось незначительно, но стабильность — признак мастерства.

### Применение метода RandomizedSearchCV к набору разных классификаторов

Использовалась функция RandomizedSearchCV из библиотеки sklearn.

В тестировании использовались следующие классификаторы:

Для тестирования использовались следующие классификаторы из библиотеки sklearn :

LogisticRegression

Ridge

SVC

BernoulliNB

RandomForestClassifier

KneighborsClassifier

DecisionTreeClassifier

К ним применялись следующие параметры:

models=[

{'name':'NB',"model":BernoulliNB(), 'params':{'alpha': uniform(loc=0, scale=4)}},

{'name':'R',"model": Ridge(), 'params':{'alpha': uniform(loc=0, scale=4), 'solver':['svd', 'cholesky', 'lsqr', 'sparse\_cg', 'sag', 'saga']}},

{'name':'SVC',"model": SVC(), 'params':{'kernel':['linear', 'poly', 'rbf', 'sigmoid'], 'gamma': ['scale', 'auto']}},

{'name':'RF',"model": RandomForestClassifier(), 'params':{'n\_estimators': [10,25,50,100,150,200], 'criterion':['gini', 'entropy'], 'max\_depth':[3,5,7,9,11]}},

{'name':'KN',"model": KNeighborsClassifier(), 'params': {'n\_neighbors':list(range(1,30)),'weights': ['uniform', 'distance'], 'p':[1,2,3]}},

{'name':'DT',"model": DecisionTreeClassifier(), 'params':{'criterion':['gini', 'entropy'], 'max\_depth':[3,5,7,9,11]}}

]

Были получены следующие результаты:

NB 0.8054772745723324 {'alpha': 0.21485488646690554}

R 0.5613369622155375 {'alpha': 2.242044673691745, 'solver': 'sag'}

SVC 0.9471134628703641 {'kernel': 'poly', 'gamma': 'scale'}

RF 0.4563553661717063 {'n\_estimators': 10, 'max\_depth': 11, 'criterion': 'entropy'}

KN 0.9149364745936429 {'weights': 'distance', 'p': 1, 'n\_neighbors': 6}

DT 0.5114676360379834 {'max\_depth': 11, 'criterion': 'gini'}

Наилучшего результата достигла модель на основе SVC при параметрах. {'kernel': 'poly', 'gamma': 'scale'}

При использовании модели SVM (метод опорных векторов) с лучшими параметрами были получены следующие результаты:

precision recall f1-score support

Negative 0.99 0.97 0.98 265

Neutral 0.97 1.00 0.98 456

Positive 0.99 0.95 0.97 277

accuracy 0.98 998

macro avg 0.98 0.97 0.98 998

weighted avg 0.98 0.98 0.98 998

Accuracy: 0.9789579158316634

Показатель 0.97895 - это очень хороший результат. Времени на обсчет ушло очень много, но кода для этого написано совсем мало.

### Предварительный вывод по второй части работы

Можно сделать вывод, что применение метода RandomizedSearchCV очень эффективный метод повышения качества классификации.

### Поиск и исследование аналогичных работ в Интернете

Классическую задачу классификации я решил c хорошим результатом. Но осадочек от неудачного использования предобученного word2vec остался.

Изучение интернета на предмет аналогичных исследований выявило две интересных работы.

Первый материал, это квалификационная работа Седова Д.Э.(Ссылка 1).

Он провел более широкие исследования влияния разных способов токенизации, лемматизации и классификации на метрики при классификации текстов.

В целом мои результаты подтверждают его выводы.

И в его работе точность при векторизации методом TF-IDF превзошла точность при векторизации методом word2vec.

Самое лучшее соотношение скорости обучения к точности были получены на паре TF-IDF + LinearSVC, практически как и в нашем случае.

Лучший же результат по точности дал метод BERT.

Изучая материалы других авторов, я увидел, что по общему мнению, новые предобученные модели на основе трансформеров, работают намного более удачно.

Так работа Kiruthika Nallichery Subramanian и Thailambal Ganapath (Ссылка 2) навела на мысль, что результат можно еще улучшить, используя для векторизации предобученный BERT, а затем классифицировать эти признаки в модели SVM.

Ссылка 1 : «Тольяттинский государственный университет» Институт математики, физики и информационных технологий

Д.Э. Седов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА) на тему «Разработка приложения для классификации текста»

Тольятти 2022

https://dspace.tltsu.ru/bitstream/123456789/23855/1/%D0%A1%D0%B5%D0%B4%D0%BE%D0%B2%20%D0%94.%D0%AD.\_%D0%9F%D0%98%D0%B1%D0%BF-1702%D0%B2.pdf

Ссылка 2 :

Облегченная система рекомендаций для анализа социальных сетей с использованием гибридного алгоритма классификатора BERT-SVM

Kiruthika Nallichery Subramanian, Thailambal Ganapathy

Научно-технический вестник информационных технологий, механики и оптики, 2022, том 22, № 4

https://ntv.ifmo.ru/file/article/21366.pdf?ysclid=llgbpfs3np236963695

## Третья часть работы: Извлечение признаков по методу BERT с последующей классификацией по модели SVM.

### Использование предобученной модели BERT

Чтобы добиться лучшего результата отказались от лемматизации и токенизации, выполненных ранее. Вместо этого вернулись к исходным текстам и использовали родной бертовский токенизатор.

Для работы с моделью и загрузки данных я использовал библиотеку pytorch и ее утилиты DataLoader, SequentialSampler, TensorDataset, а также библиотеку transformers.

После классификации признаков, выделенных BERT’ом из текстов в модели SVM получили следующий результат:

precision recall f1-score support

0 0.69 0.75 0.72 265

1 0.74 0.74 0.74 456

2 0.70 0.63 0.66 277

accuracy 0.71 998

macro avg 0.71 0.71 0.71 998

weighted avg 0.71 0.71 0.71 998

Accuracy: 0.7124248496993988

Результат уже лучше 0,7 , но все равно не блестящий. Есть куда совершенствоваться. С 0,97 на SVM с TF-IDF не сравнить.

### Предварительный вывод по третьей части работы

Я завершил выполнение «классической» задачи определения настроения по текстовым отзывам.

Лучше всего себя зарекомендовали для этой задачи алгоритмы векторизации и классификации из области «классического» машинного обучения.

Применение метода RandomizedSearchCV очень эффективный метод повышения качества классификации.

Самый точный результат дал метод классификации SVM с векторизацией TF-IDF с результатом Accuracy = 0.97 .

Самый быстрый результат дала логистическая регрессия с векторизацией TF-IDF с результатом Accuracy = 0.91 .

Использование модели BERT дало результат Accuracy = 0.71 . Возможно обучение на больших массивах «замыливает» ньюансы, когда те же слова и междометия используются для крепких выражений и бурлящих эмоций. Возможно если провести дообучение именно на этом корпусе, метрика возрастет, но сейчас для этого просто не хватило времени.

Откровенно разочаровал метод word2vec, но думаю, это потому, что мы его не совсем правильно использовали. Все таки обучаться для классификации желательно именно на конкретном корпусе документов.

Это я и попробую сделать в четвертой части работы, экспериментируя с косинусной близостью и частотностью в попытке создать прототип аналитической системы, способной выжать из наших твитов еще капельку дополнительного смысла кроме настроения.

Можно ли придумать способ узнать а на что конкретно ругаются клиенты?

## Четвертая часть работы: Выделение морфологических сущностей на основе топологической близости

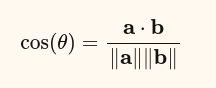
Краткое название плохо отражает суть затеи.

Попробуем начать по порядку.

Разбираясь, почему word2vec так некрасиво сработал на моих данных, я обратил внимание, что количество твитов в обучающей выборке по каждой конкретной entity порядка 2000 , чего маловато для обучения больших моделей нейронных сетей. И это как раз типовая ситуация, т. к. отзывов на каждую конкретную позицию прайса или на книгу или на блюдо бывает действительно немного. Особенно немного их бывает за последний временной период (скажем месяц), а анализировать прошлогодний снег в целях маркетинга не так уж и эффективно. Это с одной стороны.

С другой стороны желательно знать не только факт, что что-то не нравится. Хочется знать, что конкретно не нравится, как не нравится и что по этому поводу предпринимают или собираются предпринимать наши клиенты.

В этом, на небольшой по объему выборке нам как раз и может помочь word2vec, т. к. он строит свои вектора на основе косинусной близости, которая берется из контекстной близости и только при обучении на больших объемах текста это как то приближается к семантической близости. А если остановить обучение на небольшом объеме текстов, метод просмотра близости word2vec покажет нам контекстную близость слов в нашей конкретной коллекции.



Я смакетировал реализацию следующего алгоритма:

1) Обучаем классификатор на определение настроения по текстовым отзывам. Для этой цели я выбрал логистическую регрессию, т. к. она работает с точностью 0,91 на три порядка быстрее, чем наилучшая модель SVM с точностью 0,97. В нашем случае с аналитической программой будет работать человек/специалист, поэтому скорость работы важнее точности. И точность адекватную 0,97 при попытке решить нашу проблему мы все равно обеспечить не сможем, поэтому легко пожертвуем 0,06 процентами изначальной точности. Надо отметить, что отсутствие метрик для данной задачи делает невозможным ее полную автоматизацию и даже частичное исключение человека из этого анализа. Возможно в будущем эту проблему удастся решить.

2) Выбираем из наших отзывов за последний месяц, отзывы по конкретной entity, например nNvidia с предсказанной негативной меткой.

3) Обучаем на этой выборке модель word2vec

4) Токенизируем и лемматизируем тексты, и с помощью морфологического анализатора выбираем в два отдельных списка только существительные и только глаголы. Для морфологического анализа я использовал функцию nltk.pos\_tag(), которая вызывается из моей пользовательской функции ProperNounVerbExtractorLST().

5)С помощью модуля коллекций строим для этих списков счетчики (Counter()) и выводим по 100 самых часто встречающихся в этом корпусе существительных и глаголов. Это у нас два списка на обработку.

6) Добавляем еще 3 списка слов , но уже неважно каких частей речи:

- список хороших слов (позитивно окрашенных), например:

good = ['trust', 'good', 'fine', 'wow', 'popularity', 'acquire']

- список плохих слов (негативно окрашенных), например:

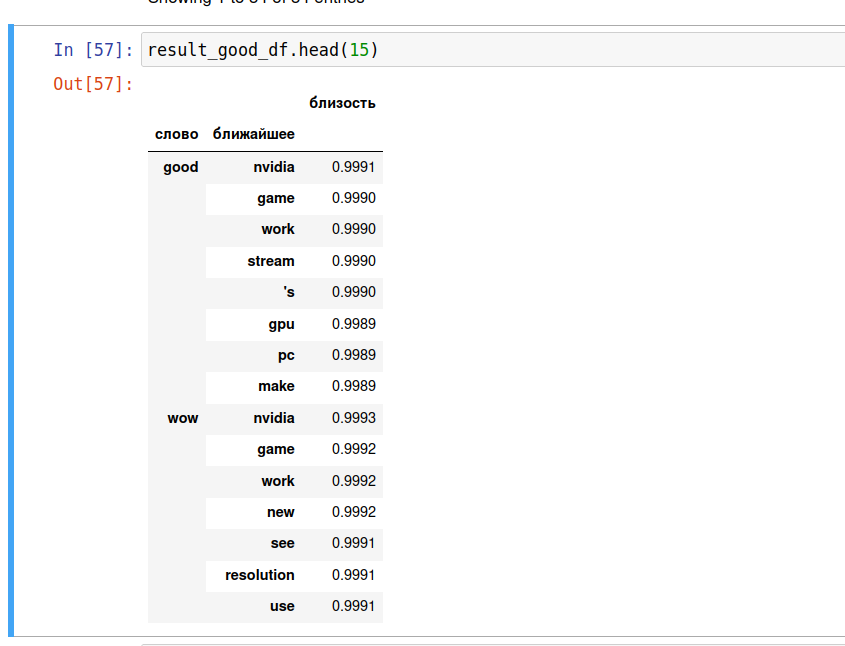
bad = ['shit', "n't", 'hate', 'fuck', 'wait', 'spend', 'suck']

- список для фильтрации (убрать из показа) общеупотребительных слов, которые не могут являться искомыми сущностями и так же для подавления мусора ввода, если остался. Например:

filter\_lst = ['’', '..', '...', '....', 'people', "'m"]

7) Первые четыре списка мы обрабатываем с помощью нашей обученной на этом корпусе модели word2vec, выбирая для каждого слова из этого списка энное количество наиболее близких по косинусному расстоянию в модели слов с указанием значения этой близости и обрезая вывод или по достижения заданного счетчика или по незначительному значению близости. Слова из списка фильтрации удаляются из вывода в процессе обработки. Обработка запускается функцией get\_ner\_lst().

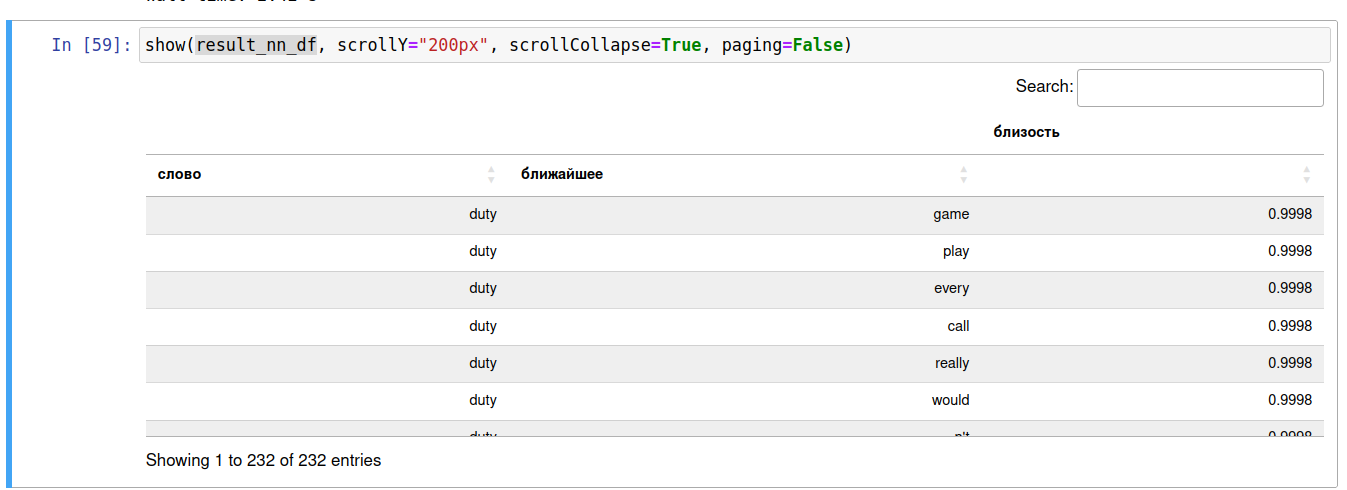
Результатом являются четыре таблицы, которые отображаются, каждая в своем интерфейсе. Например:



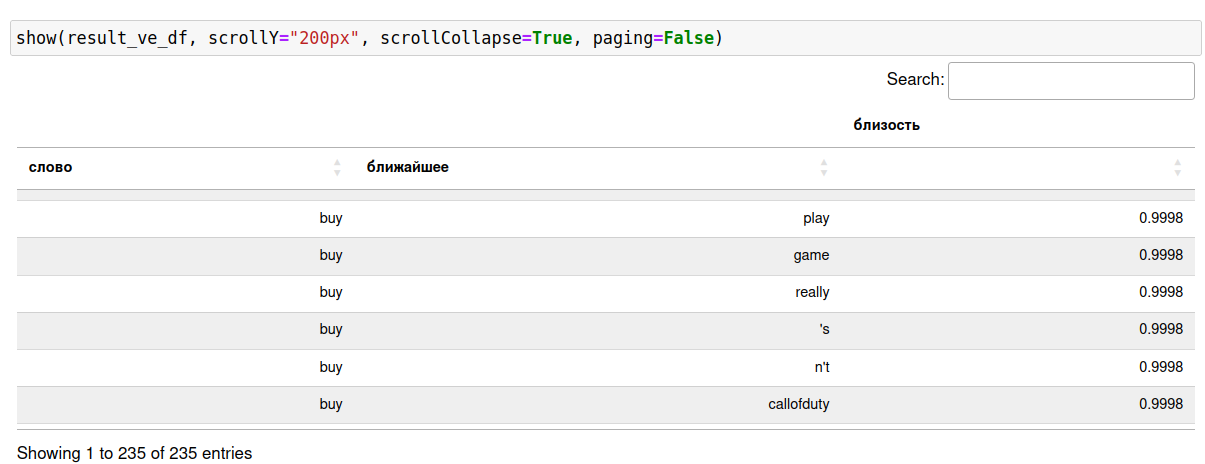
Это выборка из позитивных отзывов на Nvidia по списку хороших слов.

А ниже выборки из негативных отзывов на игру CallOfDuty

по списку существительных



по списку глаголов



по списку плохих слов



Надо сказать, что сама идея топологической близости из косинусного расстояния на несильно обученном word2vec (просто недообученном в классическом понимании) работает.

Связки слов полученных функцией w2v.wv.most\_similar() не имеют никакой семантической близости, но наиболее часто расположены рядом в наших текстах, и зная о чем речь можно догадаться о направлении мыслей клиентов, что и о чем они пишут. Но очень много белого шума. Наиболее правильной зарекомендовала себя идея с составленными заранее списками ругательных и хвалительных слов, которые позволяют «ловить на живца» сущности, которые хвалят или ругают.

К сожалению на большее количество экспериментов у меня просто не хватило времени.

Эта идея работает, но как говорят, «мазать можно, есть нельзя». Ее надо доводить до ума.

Способы улучшить результат:

1) Типичный подход, когда для каждой «сущности» составляется свой, типичный только для нее, словарь хвалебных и ругательных слов, и «вторичных сущностей», как gpu для Nvidia.

2) Можно сделать перекрестные ссылки, когда с помощью «слова ловушки» находится конкретное слово, и по клику на него делается дополнительный запрос к модели w2v.wv.most\_similar(), что бы увидеть связанные уже с ним слова и так далее.

3) Шум можно подавить мощным фильтром. Например можно выводить только слова, частотность употребления которых в этом корпусе в 3 раза больше чем их же частотность скажем по корпусу новостей.

4) Надо определить на каком количестве обучающих текстов косинусная близость по модели word2vec из топологической становится семантической. Есть подозрение, что если это так, то сильно обученные модели станут работать хуже для этой задачи, даже если обучались только на этом корпусе документов.

5) Если действительно есть казус описанный в 4 пункте, можно посмотреть, как выбрать или может быть создать модель, в которой косинусная близость будет всегда отражать топологическую/контекстную близость в предложении без эффектов переобучения на семантическую.

6) Если вникнуть в суть того, что мы ищем, то получается, что ищем мы “слова-выбросы по частоте употребления именно для этого контекста”, т.е. имеет смысл заменить вывод топа по частотности в этом корпусе выводом найденных выбросов по частоте и с ними уже работать. Но близость искать все равно по полной базе слов корпуса. Поиск “выбросов” уже более типовая задача, поддающаяся автоматизации.

Примерно такими путями можно улучшить работу моего прототипа своеобразной «аналитической системы» по поиску дополнительных смыслов в заранее классифицированных отзывах.

Такая система может быть интересна для небольших блогов, магазинов с очень разнообразным ассортиментом, «моментальных снимков» отношений к «сущностям» (на коротких интервалах времени), там, где обучение большой нейросети не рентабельно, невозможно или не требуется. А требуется быстрота, дешевизна, актуальность реального времени.

Ограничением метода является то, что это не классический ML, а для получения и анализа результатов требуется человек и вряд ли возможно построить адекватную метрику для автоматического обучения.

Скорее из этого можно сделать «аналитическую систему с применением ML» для специалиста с доработанным графическим интерфейсом.

В целом и учебой и выбранной темой доволен. Это дает возможность развиваться и по прежнему с интересом заглядываться на будущее.