Обработка естественного языка (Natural Language Processing, NLP) **—** общее направление искусственного интеллекта и математической лингвистики. Оно изучает проблемы компьютерного анализа и синтеза естественных языков. NLP включает в себя большое число задач. В этой работе будут рассмотрены следующие задачи: построение модели языка и классификация текста.

## **1. Пирамида обработки естественного языка**

Обработку естественного языка можно представить в виде пирамиды, состоящей из следующих частей:

1. Морфология: определение основных характеристик слова, таких как: часть речи, лемматизация и т.д.
2. Синтаксис: происходит построение дерева зависимостей и синтаксического дерева.
3. Семантика: происходит работа с фактическим пониманием естественного языка.
4. Прагматика: производится анализ текста в целом: определение темы текста, выделение подтем в тексте и т.д.

Особый интерес для нас представляет 3 этап этой пираиды.

## **2. Распределенные представления слов**

Согласно дистрибутивной гипотезе Харриса, слова с похожим смыслом будут встречаться в схожих контекстах. Именно эта гипотеза легла в основу современного подхода к распределенному представлению слов.

Распределенное представление слов (word embeddings) — это класс методов, в которых отдельные слова представлены в виде вещественных векторов заданной размерности. Мы рассмотрим 2 модели для построения распределенного представления слов: Word2Vec, Fasttext.

## **3. Word2Vec**

Идея, положенная в основу модели word2vec, была заложена в работе Yoshua Bengio под названием «A Neural Probabilistic Language Model» и включает в себя 2 метода: метод непрерывного мешка слов (continuous bag of words, CBOW) и метод архитектуры skip-ngram.

Суть метода CBOW заключатся в том, чтобы по заданному контексту слова восстановить само слово.  Предсказание искомого слова делается неглубокой нейронной сетью. Архитектура нейронной сети, соответствующей модели CBOW, изображена на слайде.

Процесс построения векторного пространства можно описать следующим алгоритмом:

1. На вход подается корпус Т для которого составляется словарь W, т.е. список всех уникальных слов. Из словаря удаляются наиболее редкие слова.
2. Для каждого слова собирается контекст, т.е. набор слов , удаленных не более чем на s позиций в последовательности слов корпуса Т:

.

1. Выполняется унитарное кодирование (one-hot encoding) словаря W, т.е. каждому слову ставится в соответствие вектор из нулей и одной единицы, размер вектора равен размеру словаря W, позиция единицы в векторе соответствует номеру i-ого слова в словаре W.
2. Для формирования распределенных векторов слов применяется нейронная сеть, состоящая из 3-х слоёв.
3. Входной слой представляет собой матрицу C x U, где C – количество слов в контексте, U – количество слов в словаре.
4. Скрытый слой представляет собой матрицу L, где n-ая - строка матрицы содержит вектор n-ого слова из словаря. При вычислении выхода берется среднее всех входных векторов из скрытого слоя. Выход представляет собой некоторую оценку kj для каждого слова в словаре.
5. Для расчета апостериорного распределения модели применяется функция softmax:

;

1. Функция потерь на одном окне выглядит следующим образом:

.

Метод skip-ngram работает противоположным образом: теперь не предсказывается слово по усредненному контексту, а предсказывается каждое слово контекста по данному слову.

Ключевые отличия метода skip-ngram от CBOW можно описать следующими пунктами:

1. На выходном слое нейронной сети теперь получается n мультиноминальных распределений, по одному для каждого слова контекста:

;

1. Функция потерь модели на одном окне выглядит:

.

## **4 Fasttext**

В последние годы было предложено много методов для включения морфологической информации в представления слов, чтобы лучше моделировать редкие слова. Однако ближе всего к этой цели подобралась исследовательская группа из Facebook AI Research, которая предложила свой метод, для построения векторного пространства языка, под названием Fasttex.

Основная концепция Fasttext’а была позаимствована из модели word2vec. Fasttext имеет два метода для построения векторного пространства: CBOW и skip-ngram.

Модель Fasttext исправляет один из самых важных недостатков word2vec, а именно - игнорирование внутренней структуры слов. В модели Fasttext каждое слово представлено как "мешок" n-грамм, т.е. из слова выделяются всевозможные последовательности букв длины n, также каждому слову добавлены специальные граничные символы "<"и ">", для разделения префиксов и суффиксов от других символов последовательности. Само слово также включается в набор его n-грамм, чтобы иметь представление для него.

В связи с представлением слова как мешка n-грамм, создателям Fasttext’a потребовалось изменить функцию правдоподобия между векторами. Предположим, что дан словарь n-грамм размера G. Через обозначим множество n – грамм входящих в слово w. Для каждой n-граммы gi сопоставим вектор zg. Представим слова в виде суммы его n-грамм и получим следующую функцию оценки правдоподобия:

,

где w и слова, а – вектор слова c.

## **5 GloVe**

Самой популярной альтернативой word2vec являются модели GloVe (global vectors). Для описания модели введем следующие обозначения: пусть дан словарь размера V, – матрица совместной встречаемости слов, где Xij показывает, сколько раз слово i встречается вместе со словом j, – общее число раз, которое все слова из словаря встречались со словом i. Пусть - вероятность того, что слово j появится в контексте слова i. Отправной точкой для идеи GloVe является отношение вероятностей совместного появления слов в корпусе: , т.е. отношение вероятностей встречи слов i и j в контексте слова k. В общем, модель GloVe можно выразить в следующем виде:

,

где – вектор слова, - вектор контекста слова.

В качестве функции F модель GloVe использует экспоненту и тогда итоговую функцию можно записать в виде:

.

Чтобы векторы слов и векторы контекстов стали симметричными достаточно добавить по свободному члену и . Тогда модель примет окончательный вид:

.

Так как модель GloVe использует метод обучения без учителя, то требуется определить целевую функцию. Целевую функцию для модели GloVe можно записать в следующем в виде:

,

где функция f - выступает в качестве регуляризации и имеет вид:

.

## **6 Деревья решений**

Классификаторы на основе деревьев решений (decision tree), или решающих деревьев, являются привлекательными моделями, в случае если позаботиться об интерпретируемости. Как предполагает название термина «дерево решений», эту модель можно представить как разбиение данных на подмножества путем принятия решений, основываясь на постановке серии вопросов. Опираясь на признаки в тренировочном наборе, модель дерева решений обучается серии вопросов, чтобы сделать выводы о метках классов образцов.

Используя алгоритм выбора решения, мы начинаем в корне дерева и расщепляем данные по признаку, который ведет к самому большому приросту информации (information gain, IG); этот показатель получит более подробное чуть позже. Далее мы повторяем процедуру расщепления в итеративном режиме в каждом дочернем узле, пока не получим однородных листов. То есть все образцы в каждом узле принадлежат одному и тому же классу. На практике в результате такой операции может образоваться очень глубокое дерево со многими узлами, что легко может привести к переобучению. В силу этого дерево обычно подрезается путем установления предела для его максимальной глубины.

## **7 Максимизация прироста информации**

Для того чтобы расщепить узлы в самых информативных признаках, нам нужно определить целевую функцию, которую мы хотим оптимизировать алгоритмом обучения на основе дерева. Целевая функция состоит в максимизации прироста информации при каждом расщеплении, которую мы определяем следующим образом:

*.*

Здесь f — это признак, по которому выполняется расщепление, Dp и Dj – набор данных родительского и j -го дочернего узла, I - мера неоднородности, речь о которой пойдет далее, Np - общее число образцов в родительском узле и Nj - число образцов в j-ом дочернем узле. Как можно убедиться, прирост информации — это просто разница между неоднородностью родительского узла и суммой неоднородностей дочерних узлов: чем ниже неоднородность дочерних узлов, тем больше прирост информации.

Вместе с тем для того, чтобы уменьшить комбинаторное пространство поиска, в большинстве библиотек (включая scikit-learn) реализованы бинарные деревья решений. То есть каждый родительский узел расщепляется на два дочерних узла, Dправый и Dлевый:

В бинарных деревьях решений в качестве меры неоднородности используется энтропия, которая рассчитывается как:

*.*

Здесь p(i|t) — это доля образцов, которая принадлежит классу i для отдельно взятого узла t. Следовательно, энтропия равна 0, если все образцы в узле принадлежат одному и тому же классу, и энтропия максимальна, если у нас равномерное распределение классов. Можно сказать, что энтропийный критерий пытается максимизировать взаимную информацию в дереве.

## **8 Случайный лес**

В течение прошедшего десятилетия огромную популярность в приложениях машинного обучения получили случайные леса (random forests), и причина тому – их хорошая классификационная способность, масштабируемость и простота использования. Интуитивно случайный лес можно рассматривать как ансамбль деревьев решений. В основе ансамблевого обучения лежит идея объединения слабых учеников для создания более устойчивой модели, т. е. сильного ученика, с более хорошей ошибкой обобщения и меньшей восприимчивостью к переобучению. Алгоритм случайного леса можно резюмировать в четырех простых шагах:

1. Случайным образом отобрать из тренировочного набора данных n образцов.
2. Вырастить дерево решений из выборки. В каждом узле:

1) случайным образом отобрать d признаков без возврата, т. е. бесповторным способом;

2) расщепить узел, используя признак, который обеспечивает наилучшее расщепление согласно целевой функции, например путем максимизации прироста информации;

3) повторить шаги 1 и 2 к раз;

4) для назначения метки класса агрегировать прогноз из каждого дерева на основе большинства голосов.

Несмотря на то, что случайные леса не предлагают интерпретируемости того же уровня, что и деревья решений, большое преимущество случайных лесов состоит в том, что не приходится слишком переживать о выборе подходящих значений гиперпараметров. Как правило, нам не нужно подрезать случайный лес, поскольку ансамблевая модель довольно устойчива к шуму из отдельных деревьев решений. Единственный параметр, о котором мы действительно должны позаботиться на практике, — это число деревьев k (шаг 3), который мы выбираем для случайного леса. Как правило, чем больше число деревьев, тем выше качество классификатора на основе случайного леса, достигаемая за счет повышения вычислительной емкости.

Для работы со случайным лесом библиотека scikit-learn предоставляет класс RandomForestClassifier.

В методе \_\_init\_\_ класса RandomForestManager создаем объект класса RandomForestClassifier со следующими параметрами:

* n\_estimators=130 – количество деревьев в лесу;
* random\_state=46568781 – начальное число, используемое генератором случайных чисел.

В дальнейшем мы будем использовать его для обучения модели. Но, для начала необходимо обработать данные, на которых будет обучаться модель.

## **9 TFIDF**

Вектора для слов можно получить не только с помощью word embeddings, но и с помощью других эффективных подходов. Одним из таких способов получения векторов для слов, является метод на основе оценки важности конкретного слова (термина) в контексте всего документа, входящего в базу.

Термин TF/IDF имеет англоязычное происхождение, где TF дословно означает частотность вхождения термина (от англ. словосочетания term frequency), а IDF расшифровывается, как обратная (инвертированная) частота документа (от англ. inverse document frequency). В соответствии с отношением TF/IDF весомость определенного слова (термина) прямо зависит от количества раз его использования в конкретном тексте и обратно зависима от числа использования данного слова в множестве остальных документов (текстов).

TF или частота слова — это отношение количества вхождения конкретного термина к суммарному набору слов в исследуемом тексте (документе). Этот показать отражает важность (весомость) слова в рамках определенной статьи/публикации.

IDF или обратная (инвертированная) частота документа — это инверсия частотности, с которой определенное слово фигурирует в коллекции текстов (документов). Благодаря данному показателю можно снизить весомость наиболее широко используемых слов (предлогов, союзов, общих терминов и понятий).

Для каждого термина в рамках определенной базы текстов предусматривается лишь одно единственное значение IDF. Показатель TF/IDF будет выше, если определенное слово с большой частотой используется в конкретном тексте, но редко - в других документах.

Формула TFIDF имеет следующий вид:

,

где nt - число вхождений слова t в документ d,  — общее число слов в документе d, - число документов в базе, - число документов из коллекции D, в которых встретится терм t.

Класс TfidfVectorizer библиотеки scikit-learn предоставляет оптимизированную реализацию подсчета tfidf.

В методе train() класса RandomForestManager загрузим данные с помощью метода load\_dicts(), из которого получим список словарей, где по ключу «type id» лежит идентификатор класса, а по ключу «text» - тренировочный пример (предложение), относящееся к этому классу.

Для того, чтобы посчитать tfidf, создадим объект класса TfidfVectorizer. Далее соберем наш текст в лист таким образом, чтобы каждый элемент листа представлял собой строку, в которой через пробел перечислены слова, принадлежащие одному классу, приведенные в нормальную форму. Используем метод fit() для расчета tfidf.

## **10 Тэги pymorphy2 и сочетание буквенных 3-грамм**

Для обработки текстов, немаловажной характеристикой является описание отдельно взятого слова. Отдельное слово можно описать большим числом признаков, таких как: часть речи, число, род, …. Для выделения таких характеристик используется Python библиотека под названием pymorphy2, которая умеет выделять большое число характеристик слова, называемых граммемами.

Рассмотри класс PyMorphManager, объект которого, мы уже создали в методе \_\_init\_\_ класса RandomForestManager. При создании объекта класса PyMorphManager был создан объект класса MorphAnalyzer, предоставляемого библиотекой pymorphy2. Также, мы получили все существующие тэги pymorphy2, используя TagClass.KNOWN\_GRAMMEMES.

Для того, чтобы получить слова в нормальной форме используем метод get\_normal\_form(), где для каждого слова используем метод normalise(). Здесь, с помощью метода parse() объекта MorphAnalyzer получаем тэги для слова. Так как этот метод возвращает список вариантов разбора, отсортированных по статистике употребления – берем нулевой элемент. Используем специальное приведение для некоторых частей речи или просто берем нормальную форму.

Для представления тэгов pymorphy, используем подход, описанный в методе get\_tags. Создадим вектор, заполненный нулями и имеющий размерность равную количеству тэгов в pymorphy, затем получим тэги для слова и поставим единицы в соответствующие координаты вектора.

Современным подходом для описания внутренней структуры слова считается метод буквенных 3 – грамм, которые способны симулировать некоторые морфемы слов, такие как: окончание, суффикс, приставка и некоторых корней.

Для построения 3 – грамм требуется пройтись по корпусу текста с помощью окна шириной 3, т.е. перебрать всевозможные сочетания 3 – х подряд идущих символов.

Для работы с 3 – граммами рассмотрим реализацию класса CharFeaturer.

Для начала – объявим алфавит. Установим минимальное и максимальное вхождение 3-грамм в текст (10 и 2500 соответственно). Пронумеруем буквы в алфавите.

Затем, используя метод get\_hash\_ch закодируем наши предложения. Для этого создадим вектор длинной размер\_алфавита \* 3, заполненный нулями, и для каждого предложения применим функцию get\_ords(). В функции get\_ords() мы берем последовательно каждую 3-грамму и с помощью принципа:

сумма(номер\_буквы \* размер\_алфавита ^ индекс\_буквы\_в\_3\_грамме)

получаем хэш-код для каждой 3-граммы в предложении.