ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

УДК 004.896

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»,
профессор департамента программной
инженерии, канд. техн. наук

_____ В.В. Шилов
«___» _____ 2019 г.

Выпускная квалификационная работа

на тему «Исследование методик распознавания именованных сущностей»

по направлению подготовки 09.03.04 «Программная инженерия»

Научный руководитель
_____ доцент ДПИ_____
Должность, место работы
__ канд. техн. наук, доцент _
ученая степень, ученое звание
_____С. Л. Макаров_____
И.О. Фамилия

_____
Подпись, Дата

Выполнил
студент группы _____
4 курса бакалавриата
образовательной программы
«Программная инженерия»
_____А. Е. Мачнев_____
И.О. Фамилия

_____
Подпись, Дата

Москва 2019

Реферат

В настоящее время задача выделения характеристик из названий товаров является актуальной для многих предприятий. Универсального автоматического решения данной задачи пока не найдено.

Данная задача похожа на задачи распознавания именованных сущностей в тексте, выделения объектов из текста, информационного поиска, а также связана с задачами векторизации и классификации текстов и его частей. На эти темы имеется множество научных работ. Однако, во всех этих работах уделяется внимание именно текстам на естественных языках. Эти тексты, как правило, состоят из фиксированного набора слов, а их размеры значительно превышают размеры названий товаров. Это делает невозможным применить результаты этих работ напрямую для данной задачи.

Однако, названия товаров также состоят из символов, и в целом имеют похожую структуру; в названиях товаров, как и в текстах, имеются скрытые взаимосвязи между символами. Таким образом, методики и принципы, описанные в этих работах, возможно адаптировать и применить к задаче выделения характеристик из названий товаров. В данной работе исследуется эта возможность.

Данный отчет состоит из 42 страниц, 3 формул, 17 рисунков, одной таблицы и двух приложений. Приведены ссылки на 41 источник.

Ключевые слова: выделение характеристик; методы машинного обучения; анализ текстов; классификация; информационный поиск.

Abstract

Now, many companies have a problem of extracting characteristics values from stock items names. There is still no common solution for this problem.

The problem is alike the problem of named entity recognition in text and connected with the problem of texts and parts of text embedding problem. There are some papers on these topics, however, in those papers they analyze natural language text, which could be simply splitted into words, most of which are used in texts more than once, and from which a dictionary could be built. This makes impossible to simply apply results of those papers in described problem.

However, stock item names also consist of symbols and have a common structure. As well as in text, there are some hidden connections between symbols. Therefore, methods and algorithms described in those papers could be adapted to the problem of characteristics extraction. In this work, this approach is researched.

The present report comprises 42 pages, 3 equations, 17 images, one table and 2 appendix. 41 links to sources are present.

Keywords: characteristics extraction, machine learning methods, texts analysis, classification, information retrieval.

Оглавление

Используемые определения и сокращения

Название товара – строка, описывающая товар и содержащая некоторые его характеристики

Слово – подпоследовательность идущих подряд символов в названии товара или значении характеристики, отделенная от остальных символов пробелами, знаками пунктуации, скобками и другими служебными символами.

Токен – отдельная подстрока названия товара. Токенами могут быть слова и отдельные символы

Эмбеддинг – векторное представление слова, построенное таким образом, чтобы для близких по значению или связанных по смыслу слов были построены вектора, близкие в Евклидовом пространстве

Языковая модель, модель естественного языка – модель представления текста, содержащая ключевые знания о слове, предложении, тексте. С помощью языковой модели можно выделить наиболее значимую информацию из входного предложения, текста и использовать ее для решения конкретной задачи.

Введение

В настоящие время множество компаний, работающий в различных отраслях, обладают базами товаров, где каждый товар представлен в виде текста длиной в одну строку, содержащего название товара и его ключевые характеристики. Работать с такими базами затруднительно, по следующим причинам:

- невозможен поиск по товарам с фильтрацией по значениям характеристик
- если две разные строки описывают один и тот же товар, это соответствие нельзя установить автоматически
- вследствие предыдущей причины, в базе невозможно автоматически обеспечить уникальность товаров

Таким образом, наличие данной проблемы приводит к дополнительным затратам человеческих ресурсов на поддержку и работу с такими базами. Кроме того, данная проблема блокирует возможность решить задачи, для которых требуется уникальность записей или их категоризация по значениям характеристик.

По опыту общения с сотрудниками некоторых компаний, эту проблему частично удалось решить с применением разных подходов, однако общее и открытое решение проблемы в данный момент найти не удается.

Данная проблема похожа на проблему распознавания именованных сущностей [28]. Этот метод изучены для работы с текстами, т.е. строками, в которых наблюдается явное деление на слова, и почти все слова встречаются в языке, и в том числе в других текстах. В большинстве работ по этим темам не изучается обработка коротких (до 20 слов) текстов, значительная часть которых состоит из редких или уникальных слов. Таким образом, эти методики нельзя использовать без специальной адаптации для решения данной проблемы.

В данной работе исследуется возможность применения методик по направлению распознавания именованных сущностей для задачи выделения характеристик из названий товаров.

Цель данной работы – найти и применить наиболее подходящие методы машинного обучения из направления «Распознавание именованных сущностей» для решения задачи выделения характеристик из названий товаров.

Цель исследования достигается путем выполнения перечисленных задач:

1. Поиск и изучение источников по направлениям распознавания именованных сущностей и работе с текстом.

2. Разработка способа оценки качества, применимого к данной задаче
3. Адаптация моделей данных и методик из каждой работы под данную проблему, разработка на их основе алгоритмов выделения характеристик
4. Применение каждого алгоритма на тестовых данных и получение значений метрик оценки качества
5. На основе значений метрик получить вывод, получилось ли найти алгоритм, применимый для решения данной задачи.

В главе 1 производится краткий обзор алгоритмов, описанных в найденных источниках по указанным направлениям.

В главе 2 формулируется задача, приводятся метрики оценки качества и критерии успешности исследования, описывается набор собранных данных и производится описание алгоритмов, применяемых к данной задаче.

В главе 3 описывается, то, как производится исследование, какие средства были использованы, какие результаты получены и производится анализ полученных результатов.

1 Существующие модели и алгоритмы

1.1 Обучение с частичным привлечением учителя

При решении задач обработки естественного языка часто возникает ситуация, когда имеется очень много неразмеченных данных (как правило, просто текстов) и не очень много размеченных данных. Поскольку для решения задачи часто требуется информация, которую не представляется возможным получить из размеченных данных, производится предварительное обучение (т.е. выделение информации) на неразмеченных данных. Найденная информация затем позволяет решить задачу, обучившись на небольшом наборе размеченных данных.

В частности, эмбеддинги слов и некоторые языковые модели обычно строятся с использованием большого объема (порядка миллиарда слов) текстов, т.е. неразмеченных данных, а затем полученные эмбеддинги (языковая модель) позволяют решить задачу (например, задачу распознавания именованных сущностей), имея небольшой объем размеченных данных (несколько тысяч – десятков тысяч примеров текстов, содержащих именованные сущности).

1.2 Обучение на множестве задач

Концепция обучения на множестве задач состоит в оптимизации решения сразу нескольких задач на одних и тех же входных данных. Такой подход способствует повышению обобщающей способности полученной в результате обучения модели.

Обучение с частичным привлечением учителя может являться примером обучения на множестве задач, так как этап выделения знаний из неразмеченных данных часто производится путем обучения некоторой модели на решение некоторой задачи, не совпадающей с основной задачей.

1.3 Способы векторизации слов

Поскольку ключевые алгоритмы машинного обучения могут работать только с числами (принимать на вход и выдавать на выходе), для решения многих задач, в частности, задач классификации, принято представлять в векторном виде подаваемые на вход объекты из предметной области. Вектора обычно подбирают такими, чтобы похожим объектам соответствовали близкие в Евклидовом пространстве вектора. Термин, обозначающий такие вектора – *эмбеддинг* [1].

В частности, в задачах обработки естественного языка принято строить эмбеддинги для слов, иногда – для частей слов и отдельных символов. Существует множество разных способов строить эмбеддинги слов. Способы можно условно поделить на «старые», основанные на статистическом подходе, и «новые» - появившиеся после выхода в 2013 году статьи [2] (эмбеддинги word2vec), вызвавшей ажиотажный интерес исследователей к теме эмбеддингов и спровоцировавшей выход нескольких статей в год (вплоть до данного момента) про построение эмбеддингов и языковых моделей.

Эмбеддинги слов можно воспринимать как вектора, содержащие скрытую информацию о значении слова и о контексте слова, в котором оно чаще всего встречается.

1.3.1 Латентно-семантический анализ

Метод латентно - семантического анализа впервые представлен в [4], как способ решения задач обработки естественного языка на основе скрытых тематик текстов, построенных на основе взаимной частоты встречаемости слов в документах. В общем виде, данный метод можно представить как разложение с некоторой ошибкой матрицы «слова на документы» (в которой строки соответствуют документам, а столбцы – словам, а в каждой ячейке стоит частота слова в соответствующем документе) на произведение двух матриц, где в первой матрице каждая строка соответствует документу, во второй каждый столбец соответствует слову, а столбцы первой матрицы и строки второй соответствуют *скрытым тематикам* в соответствующих документах и словах, и количество столбцов первой матрицы равно количеству строк второй.

В Вероятностном латентно – семантическом анализе (pLSI) [5] считается, такое разложение матриц описывается в виде генеративной модели [16], в которой каждое слово в документе получено из некоторого распределения вероятностей слов, описанное некоторой *тематикой* этого документа, а документ представляется в виде взвешенной «смеси» этих тематик, или распределения вероятностей тематик.

Значения в ячейках матриц разложения трактуются как вероятности. Тематикой в этом случае считается некоторая общая тематика нескольких документов, определяемая по частотам встречаемости некоторых слов в документе. Значения, стоящие в ячейках матриц разложения, можно трактовать как вероятности встретить соответствующее слово в соответствующей тематике и вероятность того, что соответствующий документ относится к соответствующей тематике, соответственно.

Модель Латентное размещение Дирихле (LDA) [6] развивает модель pLSI, добавляя предпосылки о том, что

1) все вектора документов (распределение тематик в документе) генерируются из распределения Дирихле [18] с некоторыми параметрами;
2) все вектора тематик (распределения вероятностей встретить каждое слово в соответствующей тематике) генерируются из некоторого (отличного от первого) распределения Дирихле с некоторыми параметрами.

Вектора слов (распределение тематик для слова) во всех случаях можно трактовать как эмбеддинги соответствующих слов.

1.3.2 Word2Vec

Word2Vec строит эмбеддинг слова на основе контекста, в котором это слово стоит. В основе этого метода лежит предположение о том, что близкие по значению слова стоят в похожих контекстах. Например:

1) Мама мыла **раму**.
2) Мама мыла **окно**.

Реализация word2vec – это двухслойная нейронная сеть, обрабатывающая большой набор входных текстов. В оригинальной статье представлено две модели (рисунок 1):

1) continuous bag of words, CBOW
2) continuous Skip-gram, Skip-gram

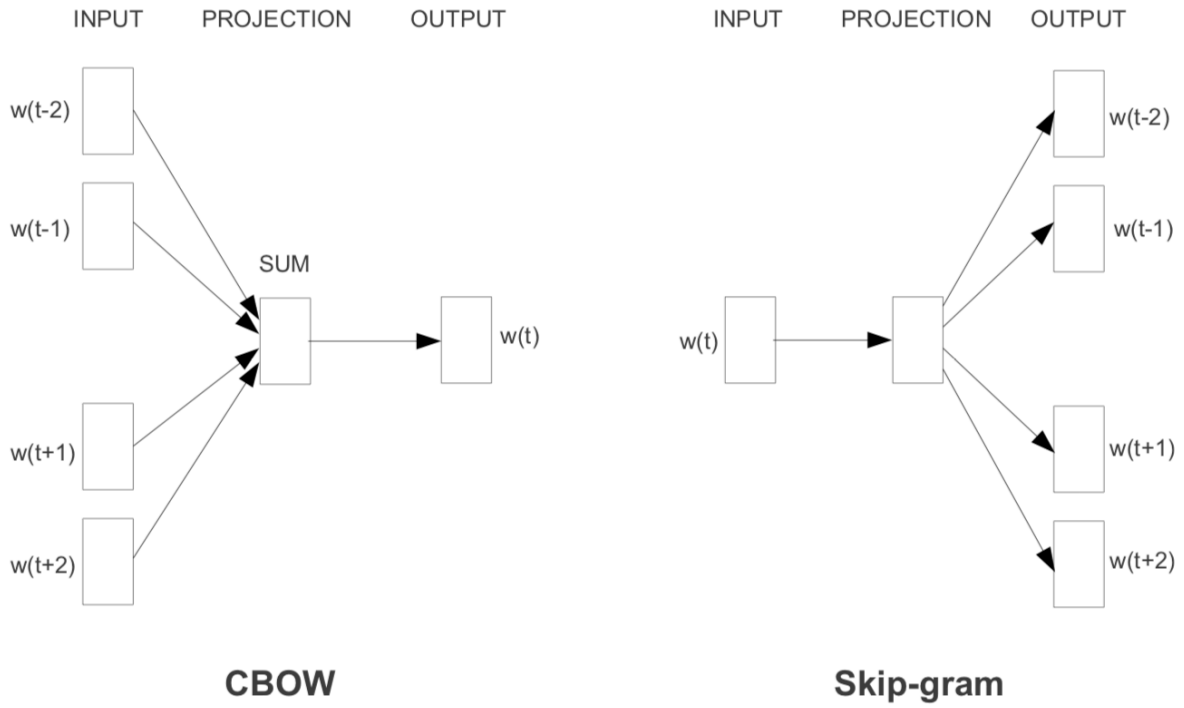

Рисунок 1. Модели Continuos Baf Of Words и Skip-gram.

Составим словарь из всех слов, имеющихся в обучающих текстах, и присвоим каждому слову индекс – порядковый номер в словаре. Размер словаря – V. One-hot вектор для *i-го* слова в словаре w_i – это такой вектор v^i размерности V, у которого все компоненты, кроме i-й, равны 0, i-я равна 1: $v_0^i = 0, v_1^i = 0, \dots, v_{i-1}^i$=0, $v_i^i = 1, v_{i+1=0}^i, \dots, v_n^i = 0$.

В [19] описана модель, взятая за основу моделей CBOW и SkipGram. Каждая модель представлена в виде нейронной сети, состоящей последовательно из слоев: слой входа, слой проекции, скрытый слой, слой выхода. Входной слой получает на вход 1 или несколько one-hot векторов слов. Вектора всех слов из входного слоя проецируются на *слой проекции* с помощью общей для всех слов *матрицы проекции*. Матрица проекции подбирается общей для всех входных слов (модель CBOW) вместо подбора весов всех связей, как в классических нейронных сетях, для того чтобы не учитывался порядок слов. В итоге на слой проекции попадает сумма проекций входных слов. В оригинальной статье про эту модель, слой проекции соединен с выходным слоем через скрытый слой. В статье про word2vec скрытый слой удаляется, слой проекции соединен с выходным напрямую. Наибольшую ценности в данной модели представляет выход слоя проекции, с помощью которого считаются эмбеддинги слов (см. ниже).

Для обучения обеих сетей тексты из обучающего набора читаются скользящим окошком ширины 2K + 1, соответственно, во внимание берется центральное слово этого окошка (для него строится эмбеддинг). Соответственно, на каждом шаге обучения прочитываются слова w(t-K) … w(t+K), во внимание берется слово w(t).

Модель CBOW обучается предсказывать w(t) по данным w(t-K) … w(t – 1), w(t+1) … w(t+K), то есть предсказывать слово по контексту, в котором оно стоит. Модель Skip-gram учится для w(t) предсказывать w(t-K) … w(t – 1), w(t+1) … w(t+K), то есть по слову предсказывать контекст, в котором оно стоит (рис. 1). В обоих случаях, для построения эмбеддинга берется выход слоя проекции. В модели CBOW, эмбеддингом слова считается усредненное значение выхода слоя проекции для всех контекстов этого слова, поданных на вход. В модели CBOW эмбеддинг слова – это сам выходной вектор слоя проекции при подаче в нейронную сеть входного слова.

Две описанные модели построения эмбеддингов вызвали интерес в широких исследовательских кругах, а идея получила дальнейшее развитие. Так, в статьях [7] и [8] показывается и развивается идея о том, что в таких эмбеддингах содержатся взаимосвязи

семантические взаимосвязи между словами. Например, $x_{queen} - x_{woman} + x_{man} \approx x_{king}$, где x_w – word2vec эмбеддинг слова w. В статье [9] описывается расширения метода построения эмбеддингов посредством представления слова в виде набора n-gram – частей слова фиксированной длины. Наконец, в статье [10] показывается, что данный метод можно представить как факторизацию (SVD – разложение) матриц [11].

Данный метод построения эмбеддингов успешно показал себя во многих задачах обработки естественного языка и представляет интерес для данной работы.

1.3.3 Сверточная нейронная сеть

Сверточная нейронная сеть [31] состоит из:

1) Набора матриц-сверток размерности сильно меньшей размерности входных данных
2) Слоя субдискретизации

На вход сети подается матрица. При выполнении сверточной нейронной сети, чередуются два действия:

1) фильтр «двигается» по всей входной матрице, и к элементам, стоящим в области данного фильтра, применяется бинарная операция с данной матрицей (скалярное произведение или умножение), результат записывается в другую матрицу.
2) К матрице, полученной на предыдущем шаге, субдискретизация: некоторое значение (обычно, максимум, или среднее) находится в строке, и записывается следующую матрицу.

На выходе такой сети получаем векторное представление входной матрицы. Коэффициенты матриц – сверток подбираются в процессе обучения.

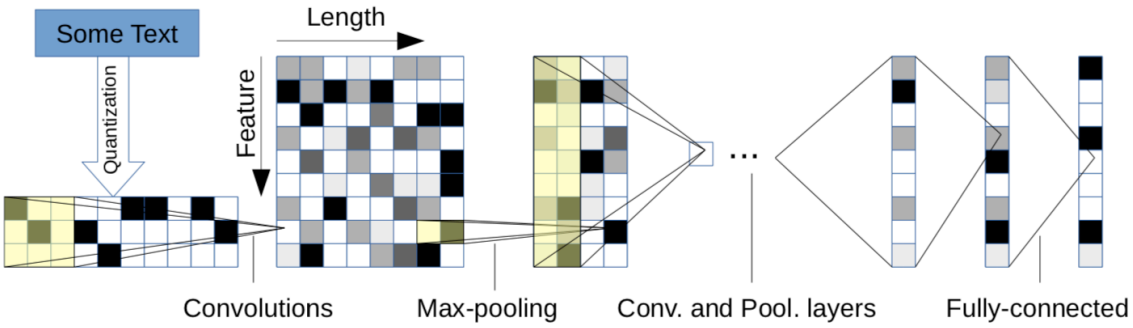

Рисунок 2. Сверточная нейронная сеть для обработки естественного языка

В статье [31] описывается идея применения матриц свертки к последовательности векторов символов слова, записанных в виде матрице (рис. X). На выходе получается вектор слова, содержащий информацию символьного уровня об этом слове.

Данный подход к векторизации слова планируется применить в текущей работе.

1.3.4 Рекуррентная нейронная сеть

В отличие от нейронной сети прямого распространения [36], рекуррентная нейронная сеть позволяет обрабатывать последовательность подаваемых значений. В классической архитектуре рекуррентной нейронной сети, некоторые значения, полученные на выходе сети после подачи очередного входного значения, подаются на вход этой же сети вместе со следующим значением. Эта модель схематично представлена на рисунке 3.

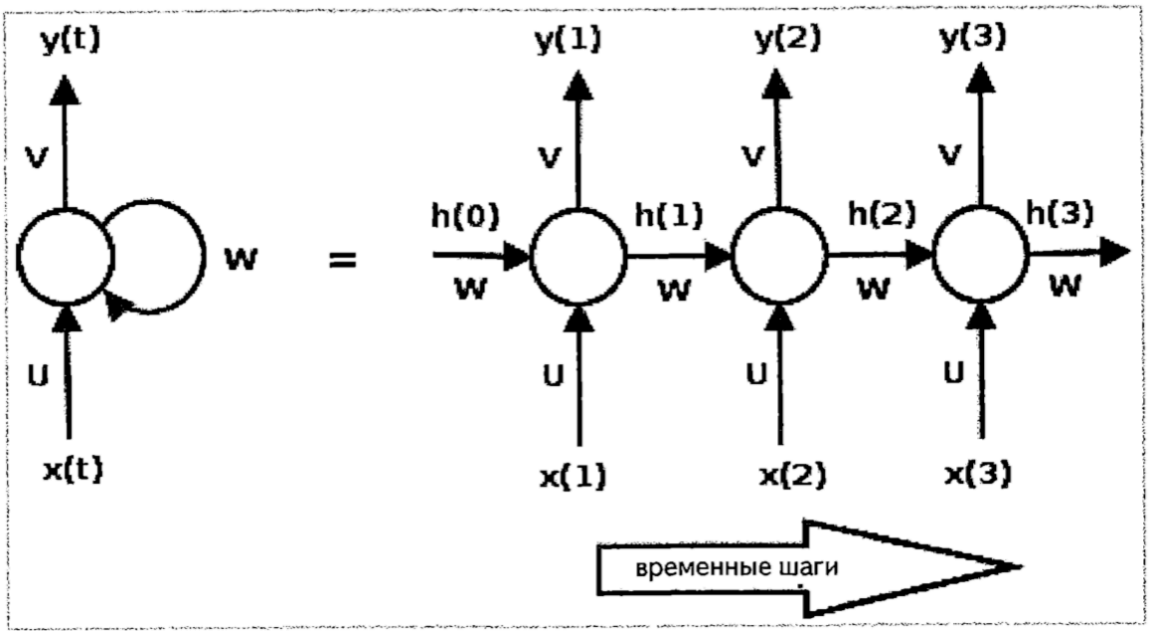

Рисунок 3. Модель рекуррентной нейронной сети. Изображение взято из [37]

Таким образом, значения с предыдущего шага, подаваемые на вход во время текущего шага, можно интерпретировать как «память» о предыдущих значениях. Благодаря этому, рекуррентные нейронные сети очень хорошо подходят для задач обработки последовательностей нефиксированной длины, в частности, для задач обработки текстов.

Для построения вектора слова, можно последовательно передавать символы этого слова в сеть и использовать выходное значение (y на рис. x) или скрытое значение (h на рис. X) в качестве вектора для слова.

Согласно [22], в классических рекуррентных нейронных сетях существует *проблема угасания памяти*: чем дальше предыдущий элемент в последовательности стоит от текущего, тем меньше информация о нем влияет на результат при подаче текущего элемента. Эта проблема решена в одной из самых современных разновидностей рекуррентных нейронных сетей – в модели «долгой краткосрочной памяти», LSTM [22]. По этой причине, а также поскольку именно LSTM используется в самых современных статьях по обработке

Мачнев А.Е. Исследование методик распознавания именованных сущностей естественного языка и распознаванию именованных сущностей ([25], [38], [39]), именно этот тип рекуррентных нейронных сетей будет использоваться в этой работе.

1.4 Языковые модели

Помимо информации о значении и наиболее частом контексте слова, для решения задач обработки естественного языка может требоваться информация о том, *в каком контексте стоит встреченное в данный момент слово* (помимо наиболее частого контекста этого слова). Также может быть важна информация о порядке слов и их взаимном расположении.

Простейшая языковая модель – «мешок слов», где имеется информация только о количестве раз, которое каждое слово языка встретилось в тексте. Современные языковые модели последовательно «читают» слова текста и выдают «контекстный вектор» на каждом шаге (прочитанном слове).

1.4.1 ELMO

В статье [25] описывается языковая модель, которая строит *вектор контекста* для каждого прочитанного слова в тексте. Идея модели состоит в том, что одно слово может иметь разные значения в разном контексте. Модель позволяет добавлять вектор этого контекста к *эмбеддингу* слова, и затем использовать оба вектора (эмбеддинг и контекстный) для обучения с учителем.

Модель обучается на большом объеме неразмеченного текста. Модель организованна следующим образом: на вход двух LSTM [22] – сетей последовательно и посимвольно подается каждое слово текста, и производится обучение этих LSTM предсказывать следующее слово (рисунок 4). Подаваемые на вход вектора могут иметь различный вид, как правило, это эмбеддинги слов и символов. В оригинальной статье, двум LSTM предшествует один сверточный слой [31] и два слоя highway – сети [32]. На вход сверточному слою подаются эмбеддинги символов соответствующего слова и его окружения.

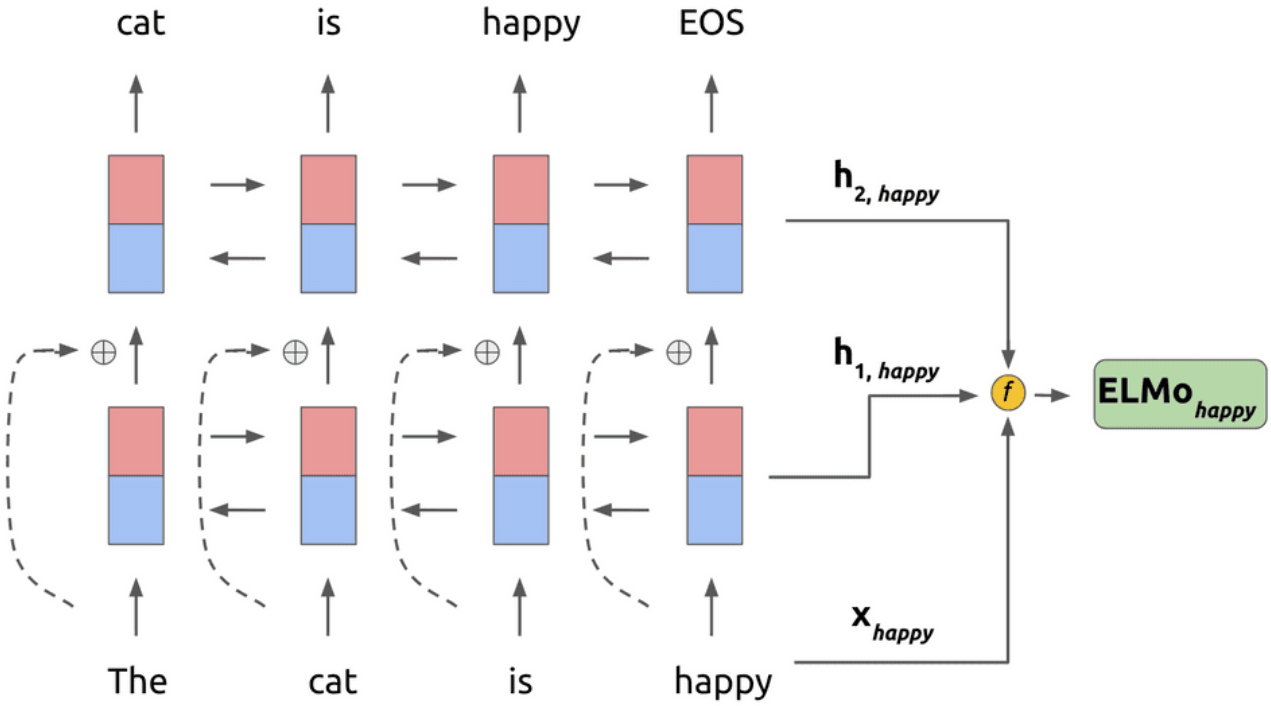

Рисунок 4 – Модель ELMO. Изображение взято из https://www.mihaileric.com/posts/deep-contextualized-word-representations-elmo/.

После обучения модели на неразмеченных текстов выходы предпоследнего слоя LSTM используются, как контекстные вектора соответствующих слов. Полученный контекстный, вектор, вместе с другими нужными векторами, используются при обучении с учителем на конкретной задаче.

Поскольку модель обучается на неразмеченных данных, которые легко найти в больших количествах, ее можно использовать в данной задаче для повышения качества путем добавления контекста.

1.5 Модели, применимые к задаче

1.5.1 Скрытая Марковская модель

Марковская цепь – последовательность случайных событий с конечным или счетным числом исходов, в которой при фиксированном настоящем будущие исходы не зависят от прошлых [12]. Формально цепь Маркова можно представить в виде вероятностного автомата с фиксированными вероятностями перехода из состояния i состояние j.

Название товара можно представить в виде последовательности слов - *наблюдений*, каждое из которых соответствует некоторому *состоянию* – характеристике, которую описывает это слово. Мы можем определить по обучающему набору вероятность встречи каждого следующего состояния при обнаружении предыдущего.

Таким образом, скрытые Марковские цепи полезны для вычисления вероятности последовательностей таких состояний для слов, когда данные состояния можно обнаружить для каждого слова [13]. Однако, в случае с распознаванием именованных сущностей на тестовых данных мы не можем обнаружить эти состояния, так как не знаем, к какой характеристике относится каждое из слов. Такие события будем называть *скрытыми состояниями.*

Скрытая Марковская модель позволяет учитывать не только явные события, но и скрытые. Модель содержит следующие элементы:

1) Набор состояний $q_1 \dots q_N$
2) Матрица вероятностей переходов между состояниями
3) Последовательность наблюдений $o_1 \dots o_T$
4) Совместное распределение наблюдений: вероятность $b_i(o_t)$, что наблюдение t будет обнаружена при состоянии i
5) Стартовое и конченое состояния, не имеющие наблюдений q_0, q_F, для которых определены вероятности переходов в / из остальные состояния $a01 a02 .. a0n$ и $a1F\ a2F \dots anF$ соответственно

Модель основана на двух допущениях:

1) Вероятность перехода в данное состояние зависти только от предыдущего состояния: $P(qi|q1 \dots qi-1) = P(qi|qi-1)$
2) Вероятность наступления события зависит только от сгенерировавшего его состояния, и не от каких других состояний или наблюдений: $P(oi|q1 \dots qi, \dots, qT, o1, \dots, oi, \dots, oT) = P(oi|qi)$

Матрица переходов между состояниями и совместное распределение наблюдений аппроксимируется по обучающим данным. По имея полученные значения, модель позволяет по имеющимся наблюдениям вычислить вероятности скрытых событий. Таким образом, модель можно обучить предсказывать последовательность указанных в названии товара характеристик (состояний) по последовательности слов (наблюдений) названия этого товара.

1.5.2 Марковская модель максимальной энтропии

Марковская модель максимальной энтропии [15] отличается от скрытой Марковской модели тем, что ослаблено первое допущение, наблюдение может зависеть не только от породившего его *текущего* состояния, но и от предыдущего состояния. В отличие от скрытой Марковской модели, Марковская модель максимальной энтропии является

Мачнев А.Е. Исследование методик распознавания именованных сущностей

дискриминативной [16][17], что обеспечивает более высокое качество модели на качественных обучающих данных.

1.5.3 Марковские случайные поля

В отличие от Марковской модели максимальной энтропии, в которой моделируется условная вероятность перехода в следующее состояние при данном текущем состоянии, в Марковских случайных полях [3] моделируется совместная вероятность данной последовательности скрытых состояний для данной последовательности наблюдений. Модель схематично представлена на рисунке 5.

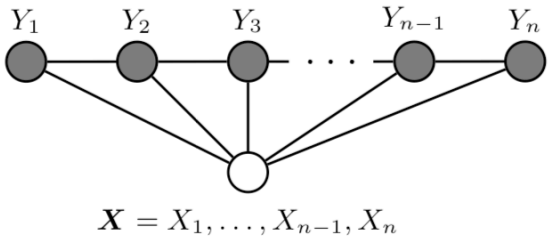

Рисунок 5. Модель Марковских случайных полей. X – набор наблюдений, Y_i – состояния. Рисунок взят из [21]

В Марковской модели максимальной энтропии существует проблема смещения метки (label bias problem) [3]: если согласно обучающим данным, вероятность перехода из состояния 0 в состояние 4 меньше, чем в состояние 1, то наиболее вероятной последовательностью состояний будет признана последовательность 0 – 1 – 2 – 3, даже если в действительности вероятность последовательности 0 – 4 – 5 – 3 выше (рисунок 6).

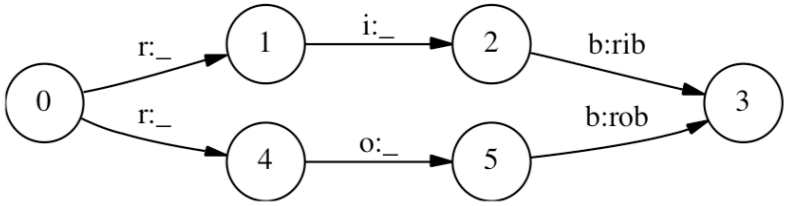

Рисунок 6 – проблема смещения метки. Взят из [3]

Поскольку в модели Марковских случайных полей находится совместное распределение вероятностей данных наблюдений и состояний, проблема смещения метки в этой модели решена.

1.5.4 Модели с использованием рекуррентных нейронных сетей

В статье [38] предлагается подавать эмбеддинги слов из входной строки в LSTM, предсказывающую категории для каждого из этих слов. В статье [39] описывается похожая

архитектура, в дополнение к которой выход LSTM подается в Марковские случайные поля, предсказывающие категории подаваемых слов. Как уже говорилось в разделе 1.3.3, рекуррентные нейронные сети, в частности, LSTM хорошо подходят для задач обработки последовательностей, а именно текстов. Так как в данных статьях показаны наилучшие результаты в задачах распознавания именованных сущностей, в данной работе будут применены именно эти модели.

1.6 Выводы по главе

В данной главе были рассмотрены проанализированные источники по обработке естественного языка. В начале была описана идея обучения на неразмеченных данных для улучшения качества модели решения конкретной задачи. Затем были рассмотрены способы векторизации слов. Далее были описаны изученные языковые модели и, наконец, рассмотрены методики, применимые к данной задаче.

2 Выделение характеристик из названий товаров

2.1 Формальная постановка задачи

Имеется набор названий товаров. В каждом названии содержится информация о товаре – ключевые характеристики этого товара.

Пусть есть некоторый способ разбиения названия товара на *слова*. Каждое слово может являться значением или частью значения некоторой характеристики. Известен набор характеристик, которые могут быть у каждого из товаров. Требуется для каждого слова определить, является ли оно частью значения какой – либо характеристики и частью какой именно характеристики оно является.

В такой постановке задача эквивалентна задаче тегирования последовательности [34].

2.2 Оценка качества модели

Хорошей метрикой оценки качества бинарной классификации является F1 – мера [41]. F1 – мера определяется через две другие метрики, полноту (recall) r и точность p (precision). Пусть TP_c – количество слов, верно отнесенных к характеристике c, FP_c – количество объектов, неверно отнесенных к характеристике c , FN_c – количество слов, являющихся частью c и не отнесенных к с. Тогда полноту r_c и точность p_c для характеристики c определяются как (1).

$$r_c = \frac{TP_c}{TP_c+FN_c},\ p_c = \frac{TP_c}{TP_c+FP_c} \qquad (1)$$

F1 – мера для характеристики с определяется, как гармоническое среднее полноты и точности (2):

$$F_1^c = 2\frac{p_c \times r_c}{p_c+r_c} \qquad (2)$$

Для оценки качества всей модели с помощью F1 меры необходимо знать значения F1 для всех характеристик. Для получения единой метрики для всех характеристик воспользуемся вычислим значение *macro F1* [35] с помощью формулы (3):

$$F_1^{macro} == \frac{1}{k}\sum_c F_1^c \qquad (3)$$

2.3 Структура моделей

Структура модели представлена на рисунке 7. Сначала название товара разбивается на слова. Все слова подаются в *слой векторизации слов,* на выходе получается вектора поданных

слов. Полученные вектора передаются в *классификатор*, который выдает последовательность характеристик, к которым относятся соответствующие входные слова.

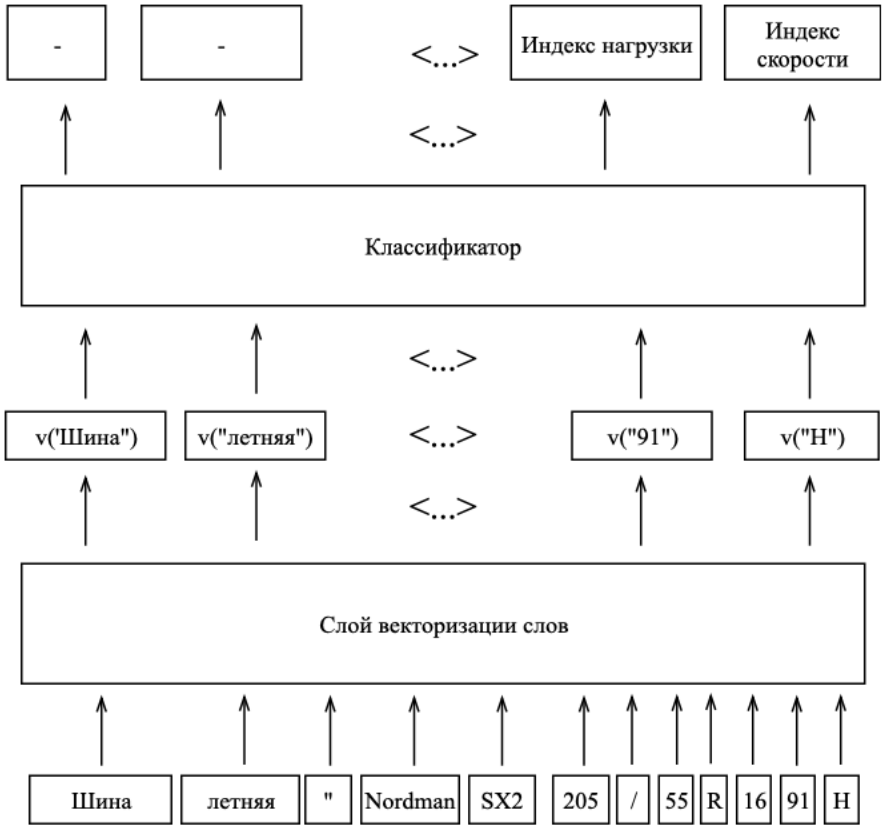

Рисунок 7. Общая структура моделей

2.3.1 Структура слоя векторизации слов

Поскольку множество слов в названиях товаров почти уникальны, из них нельзя построить словарь по обучающим данным, в котором будет содержаться большинство слов из тестового набора данных. Кроме того, построение таких словарей вызовет *переобучение* и снизить качество модели на названиях товаров, в которых отсутствуют или почти не представлены слова из составленного словаря. В связи с этим, решено строить вектора слов только на основе символов.

Вектор для каждого слова в слое векторизации слов строится отдельно. В данной работе рассматривается два способа получения вектора слова:

1) С помощью свёрточной нейронной сети
2) С помощью двунаправленной LSTM

Для каждого из способов вначале для каждого символа определяется его вектор размерности, меньшей количества символов. Вектора для символов подбираются в процессе обучения.

Далее, получение вектора для слова с помощью свёрточной нейронной сети производится следующим образом:

1) Путем построчно записи векторов всех символов слова для слова строится матрица
2) Слово подается в свёрточную нейронную сеть. Выходное значение сети используется в качестве вектора слова.

Получение вектора для слова с помощью двунаправленной LSTM производится следующим образом:

1) Вектора символов слова последовательно передаются в LSTM
2) Выходное значение LSTM после подачи последнего символа слова используется в качестве вектора этого слова.

2.3.2 Структура классификатора

В задаче, рассматриваемой в данной работе, важную роль играет наличие взаимосвязей (возможно, скрытых) между словами, стоящими в названии одного товара. Вследствие этого, предполагается, что модели, в которых в классификатор передается информация о других словах в названии товара, будут иметь более высокое качество. Рассматриваются две модели классификатора, учитывающие информацию о других словах:

1) Модель, основанная на Марковских случайных полях
2) Модель, основанная на LSTM

В данной работе будет произведено сравнение этих двух моделей с моделью, не учитывающую информацию о других словах – основанную на нейронной сети прямого распространения.

В классификаторе, основанном на LSTM, вектора слов названия товара последовательно с начала до конца и в обратном порядке передаются в LSTM. Выходы каждой из двух цепочек ячеек LSTM подаются в качестве входа цепочке, идущей в противоположном направлении (рис. X). Для каждого подаваемого слова выходные вектора обеих направлений конкатенируются, и получается скрытый вектор размерности L соответствующего слова.. Полученный вектор умножаются на матрицу размера $k \times L$ (k – количество характеристик), получается вектор размерности k (рис. X). Коэффициенты матрицы подбираются в процессе обучения. К полученному вектору применяется softmax [40]. i – е значение этого вектора можно трактовать, как вероятность отношения слова к характеристике i. Характеристикой, к

которой относится слово, является характеристика, соответствующая наибольшему значению результирующего вектора.

В модели Марковских случайных полей (CRF) по векторам слов и информации о их категориях строится вероятностный автомат со скрытыми состояниями (категориями слов) и наблюдениями (векторами слов), на основе которого затем предсказываются категории подаваемых слов.

В модели, основанной на нейронной сети прямого распространения, вектор каждого слова передается в нейронную сеть, проходит преобразования в слоях нейронной сети, и полученный вектор, как и в случае с LSTM – классификатором, умножается на матрицу проекции и проходит softmax – преобразование для получения вероятностей отношения к каждой из характеристик.

Кроме того, будут проведены эксперименты, в которых выходные значения LSTM подаются на вход CRF.

2.3.3 Все рассматриваемые модели

В данной работе будут сравнены модели, состоящие из двух описанных способов векторизации и трех способов классификации слов (рисунок 9).

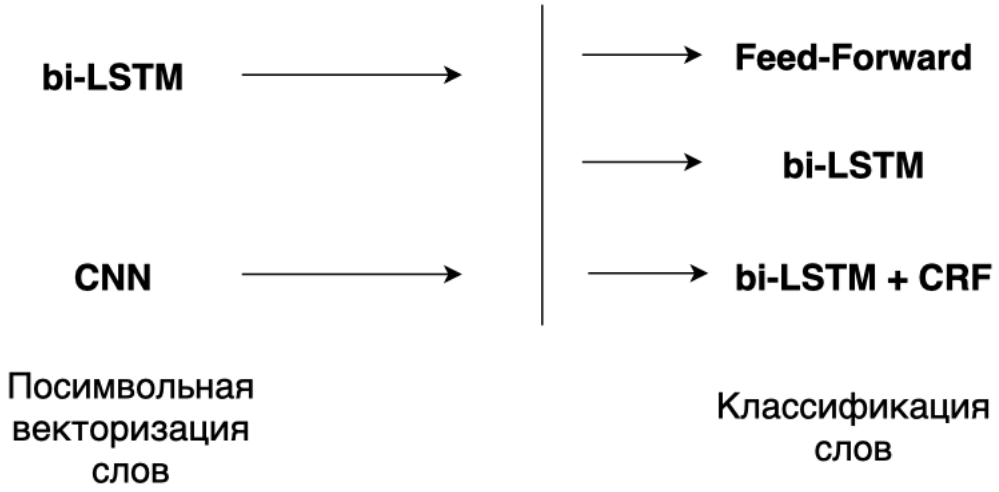

Рисунок 9. Схема построения моделей

Ниже перечислены модели, для которых будут проведены эксперименты. В приведенных обозначениях на первом месте стоят способы векторизации слов, на втором и третьем – один или два последовательных слоя классификации. CNN – сверточная нейронная сеть, Feed-Forward – сеть прямого распространения, biLSTM – двунаправленная рекуррентная сеть LSTM, CRF – Марковские случайные поля:

1) CNN - Feed-Forward
2) CNN – CRF
3) CNN – biLSTM
4) CNN – biLSTM - CRF
5) biLSTM – Feed-Forward
6) biLSTM – CRF
7) biLSTM – biLSTM
8) biLSTM – biLSTM – CRF

2.4 Выводы по главе

В данной главе была описана формальная постановка задачи и приведена метрика оценки качества моделей. Затем была описана общая архитектура рассматриваемых моделей, и ключевые ее аспекты: способы векторизации слов и способы их классификации. После чего приведен общий список экспериментально сравниваемых моделей.

3 Проведение исследования

3.1 Алгоритм исследования

Общий алгоритм исследования схематично представлен на рисунке 10.

Рисунок 10. Общий алгоритм исследования

Сначала исходный собранный набор данных нормализуется для возможности дальнейшего обучения на этом наборе. Затем производится разбиения набора на обучающий, тестовый и проверочный поднаборы.

Далее производится обучение разработанных моделей на тестовом наборе. Процесс обучения следующий: модели подается набор названий товаров из обучающего набора и делается предсказание сопоставления слов в названии товара и характеристик. На основе правильных ответов для данного набора названий товаров вычисляется ошибка, методом, определенным в модели. После чего оптимизируются коэффициенты модели методом обратного распространения ошибки.

В процессе обучения производится проверка качества текущей модели на тестовом наборе. Результирующие коэффициенты модели соответствуют максимальному значению метрики качества модели (макро F1 мера) на тестовом наборе, полученном в процессе обучения. Такой способ выбора коэффициентов модели делает модель зависимой от тестового набора. В связи с этим, для оценки реального качества модели требуется использовать набор данных, отдельный от обучающего и тестового наборов. Таким образом, оценка качества модели производится на третьем, проверочном наборе данных.

После оценки качества моделей производится анализ результатов оценки качества и делаются выводы.

3.2 Используемые средства разработки программы для исследования

3.2.1 Язык и интерпретатор Python

Код программы для исследования написан на языке Python версии 3.7, для ее выполнения используется соответствующий интерпретатор языка Python.

Выбор языка обусловлен тем, что большинство библиотек для машинного обучения, а также примеров реализации алгоритмов машинного обучения написаны на этом языке, имеют программную оболочку для этого языка, что сильно экономит время написания программы для исследования.

3.2.2 Библиотека для работы с наборами данных Pandas

Библиотека используется для чтения файлов в форматах excel и CSV.

3.2.3 Библиотека машинного обучения PyTorch

В библиотеке реализованы численные методы для большинства имеющихся алгоритмов машинного обучения. Некоторые исследуемых модели реализованы с использованием PyTorch.

3.2.4 Библиотека обработки естественных языков AllenNLP

В библиотеке реализованы основные алгоритмы, применяемые к задачам обработки естественных языков, при этом имеются возможности расширения и переопределение

Мачнев А.Е. Исследование методик распознавания именованных сущностей

существующих алгоритмов и моделей. Важной особенностью данной библиотеки является возможность описания модели в виде json – подобной конфигурации, что ускоряет проведение исследования.

3.3 Собранные данные

Для применения любых алгоритмов машинного обучения, в первую очередь необходимы большие наборы размеченных данных. Были собраны данные о товарах с сайтов https://exist.ru (24835 строк) и https://www.krepmarket.ru (16510 строк). Были выделены названия товаров и все характеристики. Формат полученных таблиц: в одной колонке стоит название товара, в остальных – значения характеристик, в первая строка каждой колонке записано название характеристики. Пример набора данных представлен на рисунке 11.

title	title_prelabeled	Материал	Шлиц	Тип головки	Диаметр, мм	Производитель	Длина, мм	Применение
Саморез 3 х35 PZ по дер.полная резьба потай оцинк		Оцинкованная стал	Pozidriv (PZ)	Потайная	3	MUSTAD	35	по дереву
Саморез PH 3,5х25 по дер.полная резьба потай оцин		Оцинкованная стал	Philips (PH)	Потайная	3.5	MUSTAD	25	по дереву
Саморез по дереву PH 3,5 х25 полная п/сф оцинк MUS	Саморез по дереву PH 3,5 х25 полная <Тип головки>п/сф</Тип головки> оцинк MUS	Оцинкованная стал	Philips (PH)	Полусфера	3.5	MUSTAD	25	по дереву
Саморез 2,5х25 PZ по дер.полная резьба потай бронз			Pozidriv (PZ)	Потайная	2.5	MUSTAD	25	по дереву
Саморез 4 х16 PZ по дер.полная резьба потай оцинк		Оцинкованная стал	Pozidriv (PZ)	Потайная	4	MUSTAD	16	по дереву
Саморез 2,5х16 PZ по дер.полная резьба потай латун			Pozidriv (PZ)	Потайная	2.5	MUSTAD	16	по дереву
Саморез 2,5х20 PZ по дер.полная резьба потай бронз			Pozidriv (PZ)	Потайная	2.5	MUSTAD	20	по дереву
Саморез 2,5х20 PZ по дер.полная резьба потай латун			Pozidriv (PZ)	Потайная	2.5	MUSTAD	20	по дереву
Саморез 3,0х20 PZ по дер.полная резьба потай бронз			Pozidriv (PZ)	Потайная	3	MUSTAD	20	по дереву
Саморез 3,5 х16 PZ по дер.полная резьба потай лату			Pozidriv (PZ)	Потайная	3.5	MUSTAD	16	по дереву
Саморез PH 2,2х 9,5 по дер.полная резьба потай оц		Оцинкованная стал	Philips (PH)	Потайная	2.2	MUSTAD	9.5	по дереву
Саморез 3,0х25 PZ по дер.полная резьба потай бронз			Pozidriv (PZ)	Потайная	3	MUSTAD	25	по дереву
Саморез PH 2,9х13 по дер.полная резьба потай оцин		Оцинкованная стал	Philips (PH)	Потайная	2.9	MUSTAD	13	по дереву
Саморез 3 х20 PZ по дер.полная резьба потай латуни			Pozidriv (PZ)	Потайная	3	MUSTAD	20	по дереву
Саморез 3,5х20 PZ по дер.полная резьба потай бронз			Pozidriv (PZ)	Потайная	3.5	MUSTAD	20	по дереву
Саморез 2,5х30 PZ по дер.полная резьба потай оцинк		Оцинкованная стал	Pozidriv (PZ)	Потайная	3	MUSTAD	20	по дереву
Саморез с насечками 3,5х30 PZ по дер.плоская голов		Оцинкованная стал	Pozidriv (PZ)	Плоская	3.5	MUSTAD	30	по дереву
Саморез 4 х16 PZ по дер.полная резьба потай бронза			Pozidriv (PZ)	Потайная	4	MUSTAD	16	по дереву
Саморез 3 х20 PZ по дер.полная резьба потай черный			Pozidriv (PZ)	Потайная	3	MUSTAD	20	по дереву
Саморез PH 4,2х13 по дер.полная резьба потай оцин		Оцинкованная стал	Philips (PH)	Потайная	3.9	MUSTAD	13	по дереву
Саморез 3 х16 PZ по дер.полная резьба потай латуни			Pozidriv (PZ)	Потайная	3	MUSTAD	16	по дереву
Саморез по дереву 3 х16 PZ полная п/сф хромир MUST	Саморез по дереву 3 х16 PZ полная <Тип головки>п/сф</Тип головки> хромир MUST	Хромированная ста	Pozidriv (PZ)	Полусфера	3	MUSTAD	16	по дереву
Саморез оцинк 4 х18 PZ по дер.полная резьба полупо			Pozidriv (PZ)	Потайная	4	MUSTAD	18	по дереву
Заклепка 3,2х 8 комбинированная /цинк/1000/						РОССИЯ		
Саморез оцинк 3,5 х16 PZ по дер.полная резьба полу			Pozidriv (PZ)	Потайная	3.5	MUSTAD	16	по дереву
Саморез по дереву PH 4,2 х13 полная п/сф оцинк MUS	Саморез по дереву PH 4,2 х13 полная <Тип головки>п/сф</Тип головки> оцинк MUS	Оцинкованная стал	Philips (PH)	Полусфера	4.2	MUSTAD	13	по дереву
Саморез по дереву 3 х12 PZ полная п/сф хромир MUST	Саморез по дереву 3 х12 PZ полная <Тип головки>п/сф</Тип головки> хромир MUST	Хромированная ста	Pozidriv (PZ)	Полусфера	3	MUSTAD	12	по дереву

Рисунок 11. Пример исходного набора данных

Для работы с данными было произведено сопоставление значений характеристик с значениями в названии товара, найдены и размечены точные позиции характеристик в названии. Формат таблиц после такой обработки описан в разделе 3.3.1.

3.4 Структура программы для исследования

Для обучения и сравнения результатов модели разработан набор скриптов с интерфейсом командной строки. Скрипты можно объединить в три модуля:

1) Скрипты обработки наборов данных
2) Модуль машинного обучения
3) Скрипт выделения характеристик

Каждый модуль запускается отдельно; предшествующий модуль использует результаты, полученные после выполнения предыдущего.

3.4.1 Скрипты обработки набора данных

Разработаны скрипты на языке Python, выполняющие следующие задачи:

1) Преобразование исходного набора данных с выделенными отдельно значениями характеристик к набору данных с размеченными названиями товаров, в котором для каждого слова указано, к какой характеристике оно относится
2) Разделение размеченного набора данных на обучающий, проверочный и тестовый наборы

Формат входных данных для скрипта (1): Excel – таблица, содержащая колонки:

1) «title» - название товара
2) «title_prelabeled» - название товара с размеченными вручную характеристиками, может отсутствовать. Формат разметки ниже.
3) <название характеристики> - множество колонок, названия которых соответствует названиям соответствующих характеристик, а значения – выделенным значениям. Выделенные значения могут не присутствовать в названии товара в точности в том же виде, в котором они указаны в значении соответствующих колонок, или не содержаться в названии товара вовсе

Задача скрипта (1) – сопоставить значения в соответствующих колонках со значениями в названии товара.

Формат выходных данных для скрипта (1):

1) «title» - название товара
2) «title_labeled» - размеченное название товара

Формат разметки названия товара: в тексте названия товара содержатся элементы вида <Название характеристики> и </Название характеристики>. Первый элемент вставляется строго перед началом значения характеристики, а второй – строго после его конца. Все остальные символы размеченного названия товара сохраняются, т.е. если вырезать эти элементы из размеченного названия, то получится исходное название товара. Пример таблицы, подаваемой на вход скрипту, указан на рисунке 11. Пример таблицы, получившейся в результате работы скрипта, указан на рисунках 12 и 13.

title	title_labeled
Саморез 2,5х16 PZ по дер.полная резьба потай оцинк	Саморез <DIAMETER>2,5</DIAMETER>х<LENGTH>16</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1
Саморез 2,5х20 PZ по дер.полная резьба потай оцинк	Саморез <DIAMETER>2,5</DIAMETER>х<LENGTH>20</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1
Саморез 4 х20 PZ по дер.полная резьба потай желтый	Саморез <DIAMETER>4</DIAMETER> х<LENGTH>20</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез 3 х30 PZ по дер.полная резьба потай оцинк	Саморез <DIAMETER>3</DIAMETER> х<LENGTH>30</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез 4 х35 PZ по дер.полная резьба потай оцинк MUSTAD (500)	Саморез 4 х<LENGTH>35</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1> <MATERIAL>оцинк</MATE
Саморез 3,5х30 Pz желтый потай с просечкой	Саморез <DIAMETER>3,5</DIAMETER>х<LENGTH>30</LENGTH> Pz желтый <TYPE1>потай</TYPE1> <TYPE2>с просечкой</TYPE2>
Саморез 3,5х50 Pz желтый потай с просечкой	Саморез <DIAMETER>3,5</DIAMETER>х<LENGTH>50</LENGTH> Pz желтый <TYPE1>потай</TYPE1> <TYPE2>с просечкой</TYPE2>
Саморез 3,5х35 Pz желтый потай с просечкой	Саморез <DIAMETER>3,5</DIAMETER>х<LENGTH>35</LENGTH> Pz желтый <TYPE1>потай</TYPE1> <TYPE2>с просечкой</TYPE2>
Саморез 3,0х40 Pz желтый потай с просечкой	Саморез <DIAMETER>3,0</DIAMETER>х<LENGTH>40</LENGTH> Pz желтый <TYPE1>потай</TYPE1> <TYPE2>с просечкой</TYPE2>
Саморез 3 х35 PZ по дер.полная резьба потай оцинк	Саморез <DIAMETER>3</DIAMETER> х<LENGTH>35</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез РН 3,5х25 по дер.полная резьба потай оцин	Саморез РН <DIAMETER>3,5</DIAMETER>х<LENGTH>25</LENGTH> <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1
Саморез по дереву РН 3,5 х25 полная п/сф оцинк MUS	Саморез <TYPE1>по</TYPE1> дереву РН <DIAMETER>3,5</DIAMETER> х<LENGTH>25</LENGTH> <PURPOSE>полная</PURPOSE> <Тип головки>п/сф</Тип
Саморез 2,5х25 PZ по дер.полная резьба потай бронз	Саморез <DIAMETER>2,5</DIAMETER>х<LENGTH>25</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез 4 х16 PZ по дер.полная резьба потай оцинк	Саморез <DIAMETER>4</DIAMETER> х<LENGTH>16</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез 2,5х16 PZ по дер.полная резьба потай латун	Саморез <DIAMETER>2,5</DIAMETER>х<LENGTH>16</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез 2,5х20 PZ по дер.полная резьба потай бронз	Саморез <DIAMETER>2,5</DIAMETER>х<LENGTH>20</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез 2,5х20 PZ по дер.полная резьба потай латун	Саморез <DIAMETER>2,5</DIAMETER>х<LENGTH>20</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез 3,0х20 PZ по дер.полная резьба потай бронз	Саморез <DIAMETER>3,0</DIAMETER>х<LENGTH>20</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>
Саморез 3,5 х16 PZ по дер.полная резьба потай лату	Саморез <DIAMETER>3,5</DIAMETER> х<LENGTH>16</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1
Саморез РН 2,2х 9,5 по дер.полная резьба потай оц	Саморез РН <DIAMETER>2,2</DIAMETER>х <LENGTH>9,5</LENGTH> <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE
Саморез 3,0х25 PZ по дер.полная резьба потай бронз	Саморез <DIAMETER>3,0</DIAMETER>х<LENGTH>25</LENGTH> PZ <PURPOSE>по дер</PURPOSE>.<TYPE2>полная</TYPE2> резьба <TYPE1>потай</TYPE1>

Рисунок 12. Пример обработанного набора данных (1)

title	MATERIAL	TYPE1	TYPE2	MODEL	COLOR	LENGTH	WIDTH	HEIGHT	DIAMETER
Саморез 2,5х16 PZ по дер.полная резьба потай оцинк	Оцинкованна	Потайная	Полная			16			2.5
Саморез 2,5х20 PZ по дер.полная резьба потай оцинк	Оцинкованна	Потайная	Полная			20			2.5
Саморез 4 х20 PZ по дер.полная резьба потай желтый	Оцинкованна	Потайная	Полная			20			4
Саморез 3 х30 PZ по дер.полная резьба потай оцинк	Оцинкованна	Потайная	Полная			30			3
Саморез 4 х35 PZ по дер.полная резьба потай оцинк MUSTAD (500)	Оцинкованна	Потайная	Полная			35			5
Саморез 3,5х30 Pz желтый потай с просечкой		Потайная	С просечкой			30			3.5
Саморез 3,5х50 Pz желтый потай с просечкой		Потайная	С просечкой			50			3.5
Саморез 3,5х35 Pz желтый потай с просечкой		Потайная	С просечкой			35			3.5
Саморез 3,0х40 Pz желтый потай с просечкой		Потайная	С просечкой			40			3
Саморез 3 х35 PZ по дер.полная резьба потай оцинк	Оцинкованна	Потайная	Полная			35			3
Саморез РН 3,5х25 по дер.полная резьба потай оцин	Оцинкованна	Потайная	Полная			25			3.5

Рисунок 13. Пример обработанного набора данных (2)

Скрипт указан в приложении Б, файлы utils/dataset_ normalizer.py и utils/normalize_dataset.py.

Скрипт (2) разделяет обработанный набор данных на обучающий, тестовый и проверочный. В качестве обучающего берется набор строк, составляющий указанную долю от данного обработанного набора, тестовый и проверочный – по половине от оставшегося после выделения обучающего набора.

Скрипт имеет следующие аргументы командной строки:

1) Путь к файлу с набором данных
2) Директория, в которую нужно сохранять результирующие наборы данных
3) Доля обучающего набора от данного
4) Доля тестового набора от данного

Скрипт указан в приложении Б, файл utils/split_train_test.py

3.4.2 Модуль машинного обучения

Модуль выделения характеристик принимает на вход набор данных в установленном формате, и выдает на выходе выделенные характеристики для каждой строки данного на вход набора данных. Методика выделения характеристик указывается с помощью отдельных опций.

Модуль имеет структуру, состоящую из пакетов layers, models, utils.

Пакет *layers* содержит описания *слоев*, т.е. шагов выделения характеристик, а именно:

1) Объекты, выполняющие векторизацию символов в названии товара
2) Объекты, осуществляющие классификацию символов входных данных

В пакете layers осуществляется переопределение некоторых классов из библиотеки AllenNLP для реализации собственной функциональности, в частности – для добавления метрики *макро – F1* в описание модели. В пакете models содержатся jsonnet – описания моделей для фреймворка AllenNLP. Модели, описываемые в конфигурациях, ссылаются на классы библиотеки AllenNLP или описанные пользователем. Пакет layers.utils содержит классы для чтения наборов данных. Формат входных данных соответствует формату выходных данных скрипта преобразования исходного набора данных.

Для обучения моделей разработан скрипт models/train.sh, приведенный в приложении Б. Скрипт имеет следующие аргументы командной строки:

1) Имя модели
2) Путь к набору данных внутри директории с наборами данных
3) Путь к конфигурации модели
4) Количество эпох (шагов) обучения

Пример вывода в консоль во время выполнения скрипта обучения, представлен на рисунке 14.

```
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.momentum_scheduler = None
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.optimizer.type = adam
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.optimizer.parameter_groups = None
2019-05-28 14:14:58,185 - INFO - allennlp.training.optimizers - Number of trainable parameters: 31559
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.optimizer.infer_type_and_cast = True
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - Converting Params object to dict; logging of default values will
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - CURRENTLY DEFINED PARAMETERS:
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.optimizer.lr = 0.01
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.num_serialized_models_to_keep = 20
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.keep_serialized_model_every_num_seconds = None
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.model_save_interval = None
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.summary_interval = 100
2019-05-28 14:14:58,185 - INFO - allennlp.common.params - trainer.histogram_interval = None
2019-05-28 14:14:58,186 - INFO - allennlp.common.params - trainer.should_log_parameter_statistics = True
2019-05-28 14:14:58,186 - INFO - allennlp.common.params - trainer.should_log_learning_rate = False
2019-05-28 14:14:58,186 - INFO - allennlp.common.params - trainer.log_batch_size_period = None
2019-05-28 14:14:58,190 - INFO - allennlp.training.trainer - Beginning training.
2019-05-28 14:14:58,190 - INFO - allennlp.training.trainer - Epoch 0/39
2019-05-28 14:14:58,190 - INFO - allennlp.training.trainer - Peak CPU memory usage MB: 268.947456
2019-05-28 14:14:58,196 - INFO - allennlp.training.trainer - Training
accuracy: 0.5431, accuracy3: 0.7501, f1_macro: 0.2218, loss: 1.3784 ||: : 79it [00:24,  3.33it/s]
```

Рисунок 14. Пример консольного вывода во время обучения.

Для проверки качества всех моделей разработан скрипт models/eval.sh, приведенный в приложении Б. Скрипт имеет следующие аргументы командной строки:

1) Имя модели
2) Поддиректория с набором данных внутри директории с всеми наборами данных

3) Директория, в которую следует записывать результаты работы скрипта

Пример файла с результатами работы скрипта указан на рисунке 15.


```
{
  "accuracy": 0.6180944286509774,
  "accuracy3": 0.827921221535125,
  "f1_Наименование1": 0.0,
  "f1_Размерность": 0.0,
  "f1_Отверстия": 0.0,
  "f1_Размер": 0.0,
  "f1_Цвет_производителя": 0.0,
  "f1_Индекс_нагрузки": 0.0,
  "f1_Индекс_скорости": 0.0,
  "f1_Диаметр": 0.0,
  "f1_Модель": 0.0,
  "f1_Модификация": 0.0,
  "f1_Вылет,_мм": 0.0,
  "f1_Диаметр_ступицы,_мм": 0.0,
  "f1_Шипы": 0.0,
  "f1_Код_цвета": 0.0,
  "f1_RunFlat": 0.0,
  "f1_Ось_установки": 0.0,
  "f1_Цвет_диска": 0.0,
  "f1_Материал": 0.0,
  "f1_macro": 0.20266913947055595,
  "loss": 1.0417545248822468
}
```


Рисунок 15. Пример результата работы скрипта оценки качества модели

Код всех классов и конфигурации всех моделей представлены в Приложении Б.

3.4.3 Скрипт выделения характеристик

Данный скрипт применяет указанную модель к указанному набору данных для выделения в нем характеристик. Входные данные скрипта – путь к сохраненной модели и excel-таблице с заголовком «title». В результате выполнения скрипта создается excel – файл, где в отдельных колонках указаны значения выделенных характеристик.

3.5 Проведение экспериментов

3.5.1 Разбиение наборов данных на обучающий, проверочный и тестовый

В AllenNLP на каждом шаге обучения производится оптимизация коэффициентов модели для минимизации установленной функции потерь, а затем результаты проверяются на указанном тестовом наборе. Коэффициенты, при которых получилось наименьшее значение функции потерь на тестовом наборе среди всех шагов обучения, используются в

Мачнев А.Е. Исследование методик распознавания именованных сущностей

результирующей модели. Таким образом, указанный тестовый набор косвенно участвует в процессе обучения и влияет на коэффициенты модели. Чтобы обнаружить переобучение, создается проверочный набор, отдельный от тестового, на котором после обучения проверяются качество модели. Код выделения этого набора представлен в приложении Б (файл utils/split_train_test.py). Используются следующее соотношения объемов данных:

1) обучающий набор составляет 67% от полного набора, тестовый – 10 %, проверочный – 23%.
2) обучающий набор составляет 20 % от полного набора, тестовый – 5%, проверочный – 75 %
3) Обучающий и тестовый наборы составляют по 5 % от полного набора, проверочный – 90 %.
4) Обучающий и тестовый наборы составляют по 1 % от полного набора, проверочный – 99 %

3.5.2 Разбиение названий товаров на слова

Разбиение наименований товаров на слова не является тривиальной задачей: некоторые символы не относятся к значениям характеристик (например, двойные кавычки или скобки), в то время как другие относятся (как, например, тире является частью значения характеристики модель). Было принято решение отделять все не пробельные, не являющиеся буквами и цифрами, символы пробелами от остальных слов, и трактовать их как отдельные слова. После такого преобразования название товара можно делить на слова пробелами.

Код разделения названия товара на слова представлен в файлах layers/utils/split_utils.py и layers/utils/splitter.py.

3.5.3 Описания конфигураций экспериментов

Эксперименты описаны в виде конфигураций на языке jsonnet. Конфигурации экспериментов приведены в приложении Б.

3.5.4 Параметры моделей и обучения

Во всех моделях используются следующие параметры:

1) Размерность эмбеддинга символа: 20
2) Размерность эмбеддинга слова: 40
3) Размер скрытого слоя LSTM и скрытых слоев нейронной сети прямого распространения: 100
4) Количество слоев нейронной сети прямого распространения: 3

5) Количество наборов фильтров свертки сверточной нейронной сети: 20
6) Размерности фильтров свертки сверточной нейронной сети: 2, 3
7) Количество шагов (эпох) обучения: 80
8) Размер «пачки» подаваемых на каждом шаге данных: 100
9) Алгоритм обучения: стохастический градиентный спуск

3.6 Результаты экспериментов

В таблице 1 приведены значения метрик для сочетаний методов векторизации и классификации слов.

Таблица 1. Значения метрики F1 для проверочного набора данных. В столбцах указаны способы векторизации слова, в строках – способы классификации. В ячейках последовательно записаны значения макро F1 меры на наборах данных: обучающий / тестовый / проверочный. Жирным выделено значение макро F1 меры на проверочном наборе

	exists.ru		Krepmarket.ru	
	CNN	biLSTM	CNN	biLSTM
Тренировочный набор = 0.1% от полного				
FeedForward	0.000 / 0.000 / **0.000**	0.014 / 0.016 / **0.012**	-	-
biLSTM + CRF	0.984 / 0.925 / **0.882**	0.988 / 0.915 / **0.889**	-	-
biLSTM	0.896 / 0.823 / **0.796**	0.915 / 0.910 / **0.821**	-	-
CRF	0.976 / 0.797 / **0.811**	0.981 / 0.821 / **0.866**	-	-
Тренировочный набор = 1% от полного				
FeedForward	0.832 / 0.759 / **0.753**	0.803 / 0.737 / **0.727**	0.000 / 0.000 / **0.000**	0.000 / 0.000 / **0.000**
biLSTM + CRF	1.000 / 0.905 / **0.878**	0.999 / 0.898 / **0.853**	0.000 / 0.000 / **0.000**	0.015 / 0.011 / **0.009**
biLSTM	0.999 / 0.897 / **0.871**	0.993 / 0.902 / **0.841**	0.000 / 0.000 / **0.000**	0.000 / 0.000 / **0.000**
CRF	0.979 / 0.901 / **0.893**	0.970 / 0.861 / **0.849**	0.670 / 0.270 / **0.239**	0.004 / 0.004 / **0.004**

Тренировочный набор = 5% от полного				
FeedForward	0.806 / 0.817 / **0.797**	0.808 / 0.820 / **0.801**	0.274 / 0.202 / **0.227**	0.205 / 0.193 / **0.174**
biLSTM + CRF	0.963 / 0.930 / **0.932**	0.979 / 0.933 / **0.936**	0.988 / 0.561 / **0.580**	0.957 / 0.512 / **0.539**
biLSTM	0.992 / 0.951 / **0.950**	0.991 / 0.923 / **0.930**	0.951 / 0.535 / **0.505**	1.000 / 0.528 / **0.546**
CRF	0.955 / 0.939 / **0.935**	0.975 / 0.967 / **0.956**	0.614 / 0.340 / **0.433**	0.390 / 0.311 / **0.325**
Тренировочный набор = 20% от полного				
FeedForward	0.776 / 0.803 / **0.810**	0.765 / 0.794 / **0.790**	0.379 / 0.299 / **0.336**	0.394 / 0.327 / **0.363**
biLSTM + CRF	0.935 / 0.984 / **0.976**	0.992 / 0.986 / **0.916**	0.934 / 0.734 / **0.784**	0.921 / 0.725 / **0.733**
biLSTM	0.987 / 0.968 / **0.917**	0.990 / 0.986 / **0.972**	0.982 / 0.777 / **0.787**	1.000 / 0.800 / **0.791**
CRF	0.916 / 0.958 / **0.951**	0.932 / 0.961 / **0.966**	0.427 / 0.361 / **0.389**	0.480 / 0.377 / **0.440**
Тренировочный набор = 67% от полного				
FeedForward	0.778 / 0.828 / **0.820**	0.773 / 0.825 / **0.816**	0.234 / 0.268 / **0.250**	0.353 / 0.304 / **0.363**
biLSTM + CRF	0.926 / 0.983 / **0.973**	0.989 / 0.984 / **0.976**	0.815 / 0.810 / **0.753**	0.894 / 0.809 / **0.811**
biLSTM	0.983 / 0.983 / **0.972**	0.967 / 0.983 / **0.923**	0.842 / 0.791 / **0.812**	0.854 / 0.833 / **0.811**
CRF	0.918 / 0.974 / **0.967**	0.904 / 0.974 / **0.956**	0.389 / 0.426 / **0.403**	0.552 / 0.468 / **0.520**

Построены графики зависимости F1-меры при всех моделей на проверочном наборе данных от размера обучающей выборки для наборов данных exists.ru и krepmarket.ru. Графики представлены на рисунках 16 и 17 соответственно.

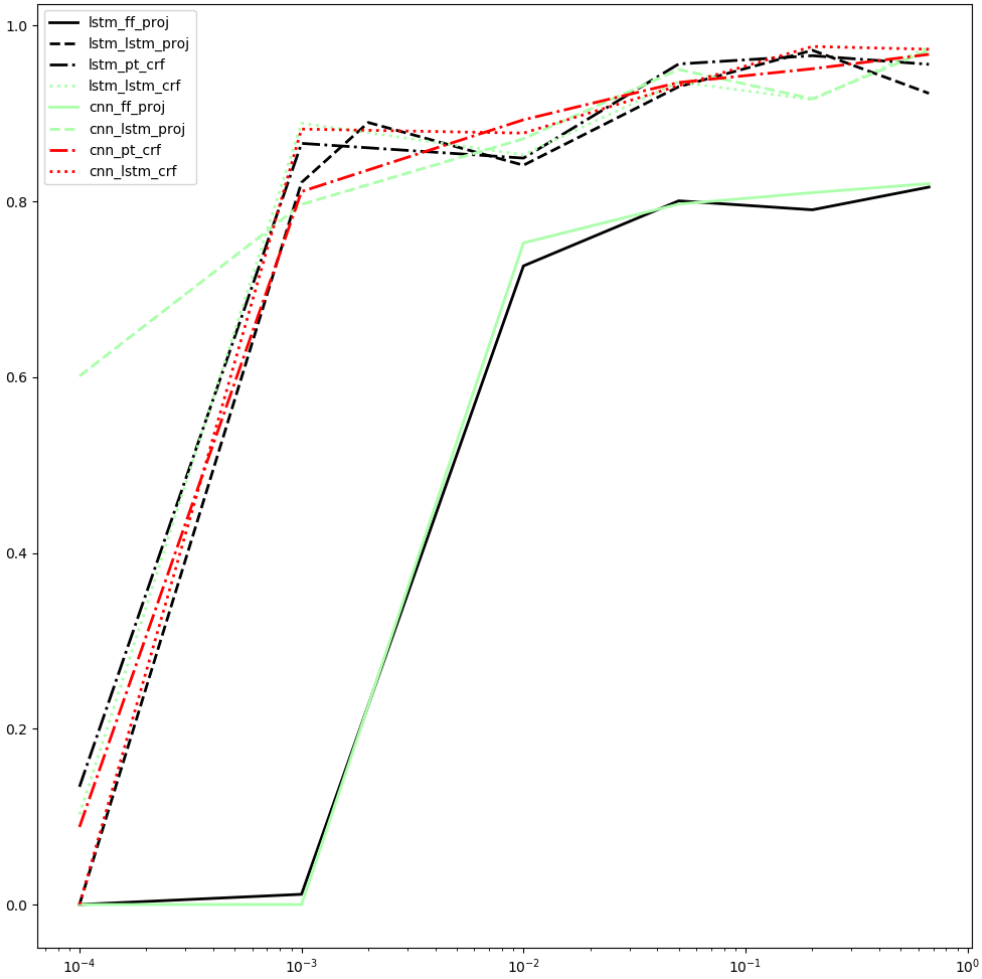

Рисунок 16. График зависимости макро – F1 меры от размера обучающей выборки на проверочном наборе данных exists.ru. По горизонтальной оси – десятичный логарифм от доли обучающего набора. По вертикальной оси – значение макро – F1 меры на проверочном наборе.

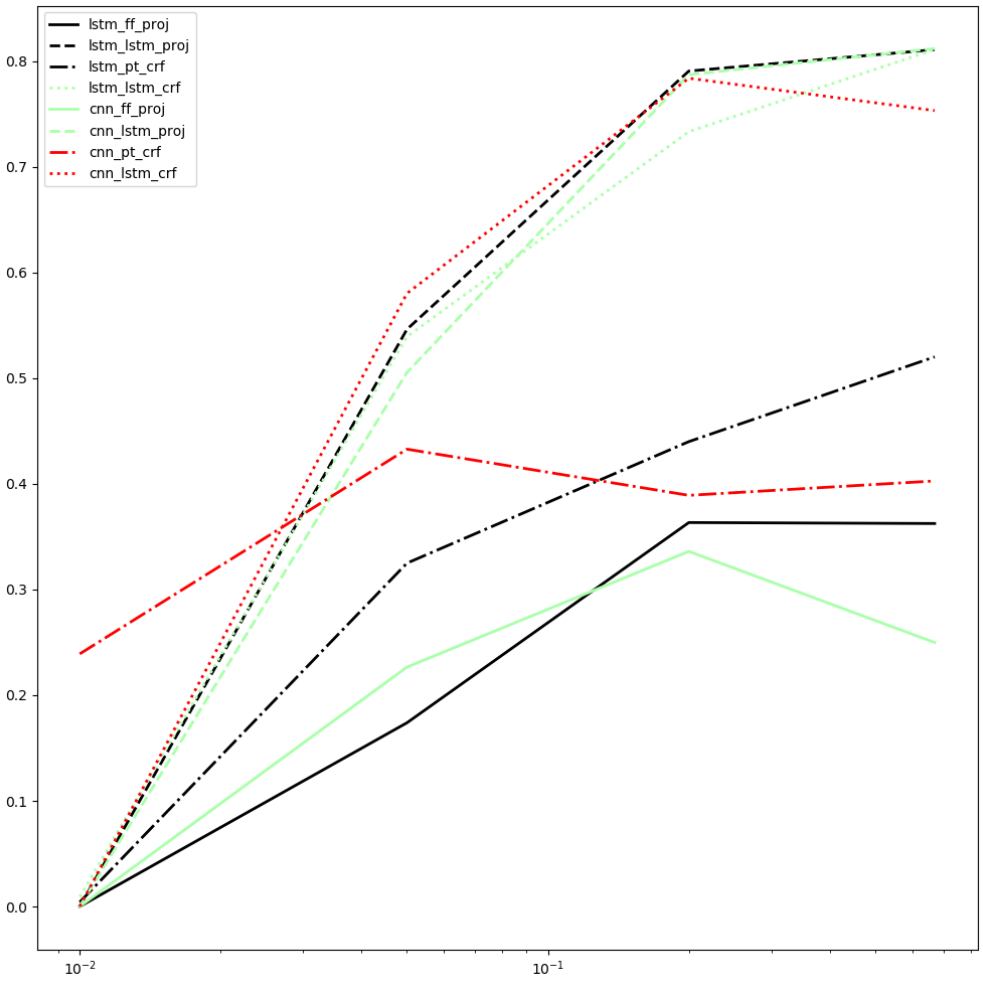

Рисунок 16. График зависимости макро – F1 меры от размера обучающей выборки на проверочном наборе данных krepmarket.ru. По горизонтальной оси – десятичный логарифм от доли обучающего набора. По вертикальной оси – значение макро – F1 меры на проверочном наборе.

3.7 Анализ результатов экспериментов

На рисунках 16 и 17 видно, что способ классификации FeedForward плохо работает на представленных наборах данных. Связано это с тем, что в большинстве названий товаров присутствует схожая структура последовательности указания характеристик, которая обнаруживается такими методами классификаторами, как LSTM и CRF. FeedForward же работает исключительно с информацией символьного уровня слова независимо от соседних слов.

Кроме того, можно заметить, что почти все модели достигают значение F1-меры, равное 0.8 на наборе данных exists.ru уже при размере обучающего набора 0.1%. Связано это с тем,

что почти все строки из этого набора имеют одинаковую структуру, отклонения от нее практически отсутствуют. Эту структуру можно выявить уже по нескольким десяткам строк.

В наборе данных krepmarket.ru имеется больше характеристик, а отклонение структуры большинства строк от общей более значительные. В связи с этим, требуется гораздо больший, чем в случае с exists.ru обучающий набор для получения качественной модели.

Как видно из рисунков 16 и 17, лучшие результаты показывают следующие модели:

1) CNN – LSTM – CRF
2) LSTM – LSTM – CRF
3) LSTM – LSTM
4) CNN – LSTM

Из рисунка 17 видно, что модель CNN – LSTM – CRF «переобучилась» на обучающем наборе размера 67 %, а модель LSTM – LSTM – CRF показала худший, чем некоторые другие, результат на обучающем наборе размера 20%. Если сравнивать оставшиеся наилучшие модели LSTM – LSTM и CNN – LSTM то в таблице 1 видно, что модель LSTM – LSTM показывает более высокие значения макро F1 – меры на обучающих наборах размера 5% и 20%. Это значит, что модель LSTM – LSTM показывает наиболее качественные результаты на небольших наборах данных и сопоставимые по качеству – на больших.

Таким образом, наилучшей моделью для решения задачи является модель LSTM – LSTM.

3.8 Выводы по главе

В данной главе была описана структура программы для исследования и изложены результаты проведенного с помощью нее эксперимента. Из результатов следует что алгоритмы выделения характеристик из названий товаров, обладающие высоким качеством, существуют, и в данной работе их удалось найти. Анализ полученных результатов показывает, что для наилучшей методикой выделения характеристик из названий товаров является применение модели с сетью LSTM для векторизации символьного представления слова и сетью LSTM для классификации слов в данной последовательности.

Заключение

В данной работе была рассмотрена задача выделения характеристик из названий товаров. Был произведен анализ современных направлений в задаче обработки естественного языка, после чего была выбрана общая структура модели для решения задачи. Были рассмотрены и экспериментально сравнены отдельные способы реализации компонентов этой модели.

Для решения задачи была предложена модель векторизации и последующей классификации слов названия товара. В качестве способов векторизации были исследованы рекуррентная нейронная сеть LSTM и свёрточная нейронная сеть. В качестве способов классификации – двунаправленная LSTM, нейронная сеть прямого распространения, Марковские случайные поля (CRF) и комбинация двунаправленной LSTM с CRF. Было показано, что наилучшей методикой выделения характеристик из названий товаров является применение модели с сетью LSTM для векторизации символьного представления слова и сетью LSTM для классификации слов в данной последовательности.

Таким образом, задачи исследования выполнены и цель достигнута.

Список источников

[1] Amir Globerson, Gal Chechik, Fernando Pereira, Naftali Tishby. Euclidean Embedding of Co-occurrence Data // Journal of Machine Learning Research 8 (2007) 2265-2295

[2] Tomas Mikolov, Greg Corrado, Kai Chen, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space // Proceedings of the International Conference on Learning Representations (ICLR 2013)

[3] John Lafferty, Andrew McCallum, Fernando C.N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data // Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001), pages 282-289

[4] Thomas Landauer, Peter W. Foltz, & Darrell Laham (1998). “Introduction to Latent Semantic Analysis” (PDF) // Discourse Processes. 25: 259—284

[5] Thomas Hofmann. Probabilistic Latent Semantic Indexing // Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in Information Retrieval (SIGIR-99), 1999

[6] David M. Blei, Andrew Y.Ng, Michael I. Jordan. Latent Dirichlet Allocation // Journal of Machine Learning Research 3 (2003) P. 993-1022.

[7] Tomas Mikolov Scott Wen-tau Yih Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations // Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)| May 2013

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality // Proceeding NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 Pages 3111-3119

[9] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov. Enriching Word Vectors with Subword Information // Transactions of the Association for Computational Linguistics, Volume 5, 2017, p. 135 - 146

[10] Yitan Li, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhon, Enhong Chen. Word Embedding Revisited: A New Representation Learning and Explicit Matrix Factorization Perspective // IJCAI'15 Proceedings of the 24th International Conference on Artificial Intelligence Pages 3650-3656

[11] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. 2.6 Singular Value Decomposition // Numerical Recipes in C. — 2nd edition. — Cambridge: Cambridge University Press.

[12] Кемени Дж., Снелл Дж. Конечные цепи Маркова. — М. : Наука, 1970. — 272 с.

[13] Chapter 6: “Hidden Markov and Maximum Entropy Models” in Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of September 18, 2007

[14] Susan Li. Named Entity Recognition and Classification with Scikit-Learn // https://spoonshot.com/blog/name-entity-recognition-using-crf/

[15] Andrew McCallum Dayne Freitag Fernando C. N. Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation // Proceeding ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning Pages 591-598

[16] Prathap Manohar Joshi. Generative VS Discriminative Models // https://medium.com/@mlengineer/generative-and-discriminative-models-af5637a66a3

[17] David S. Batista. Maximum Entropy Markov Models and Logistic Regression // http://www.davidsbatista.net/blog/2017/11/12/Maximum_Entropy_Markov_Model/

[18] Sue Liu. Dirichlet distribution // https://towardsdatascience.com/dirichlet-distribution-a82ab942a879

[19] Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Ma- chine Learning Research, 3:1137-1155, 2003.

[20] Phuong Le-Hong, Xuan-Hieu Phan, Tran The Trung. On the Effect of the Label Bias Problem in Part-of-Speech Tagging // The 2013 RIVF International Conference on Computing & Communication Technologies - Research, Innovation, and Vision for Future (RIVF)

[21] Hanna M. Wallach. Conditional Random Fields: An Introduction, 2004

[22] Alex Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network // ArXiv, 2018

[23] Hunter Heidenreich, Inroduction to Word Embeddings // http://hunterheidenreich.com/blog/intro-to-word-embeddings/, 2018

[24] Jacob Delvin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // Google AI Language, 2018

[25] Mattew E. Peters, Mark Neuman, Mohit Iyyer, Matt Gardner. Deep contextualized word representations // Association of Computation Linguistic, 2018

[26] Simeon Kostadinov, Understanding Encoder-Decoder to Sequence Model // https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346, 2019

[27] Aditya Mishra, Metrics to Evaluate your Machine Learning Model // https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234, 2018

[28] Mónica Marrero, Julián Urbano, Sonia Sánchez-Cuadrado, Jorge Morato, Juan Miguel Gómez-Berbí. Named Entity Recognition: Fallacies, Challenges and Opportunities // University Carlos III of Madrid

[29] Christopher D. Manning, Prabhakar Raghavan, Hinrich Shutze. An Introduction to

[30] В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 163.4:845-848. стр. 845 – 848

[31] Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level convolutional networks for text classification // Proceeding NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 Pages 649-657

[32] Rupesh Kumar Srivastava, Klaus Greff, Jürgen Schmidhuber. Highway Networks // ArXiv, 2015

[33] David D. Lewis. Evaluating text categorization // Proceeding HLT '91 Proceedings of the workshop on Speech and Natural Language Pages 312-318

[34] Adnan Akhundov, Dietrich Trautmann, Georg Groh. Sequence Labeling: A Practical Approach // ArXiv, 2018

[35] What is the best validation metric for multi-class classification? // https://sebastianraschka.com/faq/docs/multiclass-metric.html

[36] Daniel Svozil, Vladimir KvasniEka, JiE Pospichal. Introduction to multi-layer feed-forward neural networks // Chemometrics and Intelligent Laboratory Systems 39 (1997) p. 43-62

[37] Антонио Джулли, Суджит Палл. Библиотека Keras – инструмент глубокого обучения. Реализация нейоронных сетей с помощью библиотек Theano и TensorFlow // ДМК Пресс, 2018, стр. 178

[38] Jason P.C. Chiu, Eric Nichols. Named Entity Recognition with Bidirectional LSTM-CNNs // Transactions of the Association for Computational Linguistics, Volume 4, p. 357–370

[39] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer. Neural Architectures for Named Entity Recognition // Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, p. 260-270

[40] Multi-Class Neural Networks: Softmax // https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

[41] Koo Ping Shung. Accuracy, Precision, Recall or F1? // https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

Приложение А

Набор данных для обучения

Набор данных для обучения расположен в файле *exists_ru.xlsx* в директории *dataset* на носителе типа USB-Flash Card в связи с большим объемом.

Также в директории *dataset* расположены файлы *exists_ru_normalized.xlsx, exists_ru_train.xlsx, exists_ru_validation.xlsx* и *exists_ru_test.xlsx,* полный нормализованный, обучающий, проверочный и тестовый нормализованный набор данных, соответственно.

Приложение Б

Исходные коды программы для экспериментов

Файл utils/dataset_normalizer.py, набор методов для нормализации наборов данных:

```
'''

Given a dataset with title column, some valued characteristics columns,
some attribute columns and some junk columns (e.g. barcode, price, image link, etc.).
If there is title_prelabaled, it would be used

Purpose is to extract those values.

'''

import pandas as pd
import re

def replace_comma_in_numbers(s):
    return re.sub(r'(\d),(\d)', r'\1.\2', s)

def replace_dot_in_words(s):
    return re.sub(r'([^(\W|\d)] ?)\.( ?[^(\W|\d)])', r'\1 \2', s)

def replace_x_in_numbers(s):
    if stable:
        return re.sub(r'(\d\s?)[xx](\s?\d)', r'\1*\2', s)
    else:
        return re.sub(r'(\d\s?)[xx](\s?\d)', r'\1 * \2', s)

def replace_slash_in_words(s):
    return re.sub(r'([^(\W|\d)] ?)/( ?[^(\W|\d)])', r'\1 \2', s)

def is_number_with_slash(s):
    return re.match(r'\d/\d', s) is not None

def replace_slash_in_numbers(s):
    return re.sub(r'(\d ?)/( ?\d)', r'\1 \2', s)

def surround_with_spaces(s, regex, char):
    return re.sub(r'(\w ?)' + regex + '( ?\w)', r'\1 ' + char + r' \2', s)

def surround_with_spaces_words(s, regex, char):
    return re.sub(r'([^(\W|\d)] ?)' + regex + '( ?[^(\W|\d)])', r'\1 ' + char + r' \2', s)
```

```
def separate_words_and_numbers(s):
    s = re.sub(r'([^(\W|\d)])(\d)', r'\1 \2', s)
    s = re.sub(r'(\d)([^(\W|\d)])', r'\1 \2', s)
    return s

def prepare_value(s):
    s = s.lower()
    s = replace_comma_in_numbers(s)
    s = replace_x_in_numbers(s)
    s = replace_slash_in_words(s)
    s = replace_slash_in_numbers(s)
    s = replace_dot_in_words(s)
    s = separate_words_and_numbers(s)
    return s

def prepare_stable(s):
    s = s.lower()
    s = replace_comma_in_numbers(s)
    s = replace_x_in_numbers(s)
    s = replace_slash_in_words(s)
    s = replace_slash_in_numbers(s)
    s = replace_dot_in_words(s)
    return s

def split(s):
    s = prepare_value(s)

    tokens = re.split(r'\s|\n|;|,|"|-|\*|\(|\)', s)
    return [t for t in tokens if t is not None and t != '']


def parse_prelabeled(s):
    find_from = 0
    original_s = ''

    result = {}

    first_tag_start_index = 0
    while first_tag_start_index != -1 and first_tag_start_index < len(s):
        first_tag_start_index = s.find('<', find_from)
        if first_tag_start_index != -1:
            first_tag_end_index = s.find('>', first_tag_start_index)
            tag_name = s[first_tag_start_index+1:first_tag_end_index]
            second_tag_start_index = s.find('</', first_tag_end_index+1)
            second_tag_end_index = s.find('>', second_tag_start_index)

            value = s[first_tag_end_index+1:second_tag_start_index]
```

```
            original_s += s[find_from:first_tag_start_index]
            value_start_index = len(original_s)
            original_s += value
            value_end_index = len(original_s)

            result[tag_name] = (value_start_index, value_end_index)
            find_from = second_tag_end_index + 1

    return result


def fill_title_intersects(title_intersects, start, end):
    title_intersects[start:end] = [True] * (end - start)


def fill(title_raw, char_name, start, end, positions, title_intersects, to_resolve = None,
to_resolve_df = pd.DataFrame(),
         original_value = ''):
    if start and end:
        if char_name not in positions:
            positions[char_name] = []
        positions[char_name].append((start, end))
        fill_title_intersects(title_intersects, start, end)
    elif to_resolve:
        for start, end in to_resolve:
            to_resolve_df = to_resolve_df.append({'title': title_raw, 'char_name': char_name,
                                                  'char_value': title_raw[start:end],
'original_value': original_value},
                                                 ignore_index=True)

    return to_resolve_df


def parse_prelabeled_and_apply(title_prelabeled, positions, title_intersects):
    char_to_indices = parse_prelabeled(title_prelabeled)
    for char_name, (start, end) in char_to_indices.items():
        fill(title_prelabeled, char_name, start, end, positions, title_intersects)


def get_title_token_positions(title, title_tokens):
    result = []
    find_from = 0
    for t in title_tokens:
        index = title.find(t, find_from)
        result.append(index)
        find_from = index + len(t)
    return result
```

```
def extract_char_positions(title_raw, char_value,
                           step, max_step, title_intersects):
    extracted_char_positions = []

    title_tokens = split(title_raw)
    title_token_positions = get_title_token_positions(prepare_stable(title_raw), title_tokens)

    char_value_tokens = split(char_value)

    for i in range(len(title_tokens)):
        extracted_token_indices = []
        # Findings matching tokens sequence
        for j in range(len(char_value_tokens)):
            index = i + j
            if index >= len(title_tokens):
                break
            title_token = title_tokens[index]
            char_value_token = char_value_tokens[j]
            if char_value_token == title_token \
                    or char_value_token.startswith(title_token) \
                    or (len(title_token) >= 3 and title_token.startswith(char_value_token)) \
                    or (step > 1 and title_token.startswith(char_value_token)) \
                    or (step > 2 and title_token[:3] == char_value_token[:3]):
                extracted_token_indices.append(index)
            elif step < 2:
                break

        # Append match
        if len(extracted_token_indices) > 0:
            first = extracted_token_indices[0]
            last = extracted_token_indices[-1]

            start = title_token_positions[first]
            end = title_token_positions[last] + len(title_tokens[last])
            if not any(title_intersects[start:end]): # Must not intersects with already extracted
chars
                extracted_char_positions.append((start, end, len(extracted_token_indices)))

    if len(extracted_char_positions) > 0:
        max_tokens_count = max(arr[2] for arr in extracted_char_positions)
        extracted_char_positions = [(start, end) for start, end, tokens_num in
extracted_char_positions
                                    if tokens_num >= max_tokens_count]

        if (max_tokens_count == 1 and step > 0) or step > 1:
            extracted_char_positions = [max(extracted_char_positions, key=lambda tupl: tupl[1] -
tupl[0])]
```

```
            if len(extracted_char_positions) == 1:
                start = extracted_char_positions[0][0]
                end = extracted_char_positions[0][1]

                return start, end, None

            elif step == max_step:
                return None, None, extracted_char_positions

        return None, None, None


def get_char_value_if_need_extracting(row, char_name, original_char_name, extracted_chars):
    if char_name in extracted_chars:
        return None
    if pd.isna(row[original_char_name]):
        return None
    char_value = str(row[original_char_name])
    if not char_value or char_value == '':
        return None
    return char_value



def get_labeled_tittle(title_raw, positions):
    p2c = []
    for c, pos in positions.items():
        for start, end in pos:
            p2c.append((start, end, c))

    prev_index = 0
    title_labeled_tokens = []
    p2c.sort(key = lambda sec: sec[0])
    for start, end, c in p2c:
        if start - prev_index > 0:
            title_labeled_tokens.append(title_raw[prev_index:start])
        title_labeled_tokens.append('<{}>{}</{}>'.format(c, title_raw[start:end], c))
        prev_index = end
    if prev_index < len(title_raw):
        title_labeled_tokens.append(title_raw[prev_index:])

    return ''.join(title_labeled_tokens)

class DatasetNormalizer:
    def __init__(self, characteristics, attributes, attr_yes_values = ['да', 'есть']):
        print(characteristics)
        print(attributes)
        self.characteristics = characteristics
```

```
        self.attributes = attributes
        self.attr_yes_values = attr_yes_values
        self.all_chars = {}
        self.columns = []
        for char_name, originals in self.characteristics.items():
            for original_char_name in originals:
                self.all_chars[original_char_name] = char_name
            self.columns.append(char_name)
        for char_name in self.attributes:
            self.all_chars[char_name] = char_name
            self.columns.append(char_name)

    def normalize(self, input_file, output_dir, dataset_name):

        df = pd.read_excel(input_file)
        result_df = pd.DataFrame(columns=['title', 'title_labeled'] + self.columns)
        to_resolve_df = pd.DataFrame(columns=['title', 'char_name', 'char_value'])

        for index, row in df.iterrows():
            title_raw = str(row['title'])
            title_intersects = [False] * len(title_raw)

            result_row = {}
            result_row['title'] = title_raw

            positions = {}

            if not pd.isna(row['title_prelabeled']):
                parse_prelabeled_and_apply(str(row['title_prelabeled']), positions,
title_intersects)

            max_step = 3
            for step in range(max_step + 1):
                for char_name, originals in self.characteristics.items():
                    for original_char_name in originals:
                        char_value = get_char_value_if_need_extracting(row, char_name,
original_char_name, positions)
                        if char_value:
                            start, end, to_resolve = extract_char_positions(title_raw, char_value,
step, max_step, title_intersects)
                            to_resolve_df = fill(title_raw, char_name, start, end, positions,
title_intersects, to_resolve,
                                                 to_resolve_df, char_value)

                for char_name in self.attributes:
                    char_value = get_char_value_if_need_extracting(row, char_name, char_name,
positions)
                    if char_value:
```

```
                    if char_value.lower() in self.attr_yes_values:
                        start, end, to_resolve = extract_char_positions(title_raw, char_name,
step, max_step, title_intersects)
                        to_resolve_df = fill(title_raw, char_name, start, end, positions,
title_intersects, to_resolve,
                                             to_resolve_df, char_name)

            for original_char_name, char_name in self.all_chars.items():
                if not pd.isna(row[original_char_name]):
                    result_row[char_name] = row[original_char_name]

            result_row['title_labeled'] = get_labeled_tittle(title_raw, positions)
            result_df = result_df.append(result_row, ignore_index=True)

            for original_char_name, char_name in self.all_chars.items():
                if char_name in positions:
                    continue
                char_value = get_char_value_if_need_extracting(row, char_name, original_char_name,
positions)
                if char_value and char_name not in positions:
                    to_resolve_df = to_resolve_df.append({'title': title_raw, 'char_name':
char_name,
                                                          'char_value': None,
                                                          'original_value': char_value},
                                                         ignore_index=True)


            if index % 100 == 0:
                print('Processed {0} out of {1} ({2:.2f} %)'.format(index, len(df.index),
                                                                    (100 * (index /
len(df.index)))))

        result_df.to_excel(output_dir + '/' + dataset_name + '_normalized.xlsx')
        to_resolve_df.to_excel(output_dir + '/' + dataset_name + '_to_resolve.xlsx')
```

Файл utils/normalize_dataset, скрипт для нормализации набора данных:

```
from utils.dataset_normalizer import *
import utils.exists_ru.exists_ru_columns as exists_ru
import utils.krepmarket.krepmarket_columns as krepmarket

# Harcoded characteristics, so cannot use different dataset
# To use it, write own dataset normalizer
```

```
dn = DatasetNormalizer(krepmarket.TARGET_CHARACTERISTICS, krepmarket.ATTRIBUTES)
dn.normalize('datasets/krepmarket/krepmarket.xlsx', 'temp/datasets/krepmarket', 'krepmarket')

dn = DatasetNormalizer(exists_ru.CHARACTERISTICS, exists_ru.ATTRIBUTES)
dn.normalize('datasets/exists_ru/exists_ru.xlsx', 'temp/datasets/exists_ru', 'exists_ru')
```

Файл utils/split_train_test.py, скрипт для разделения набора данных на тестовый и проверочный:

```
import pandas as pd
import numpy as np
import sys
import os

dataset_file = sys.argv[1]

output_dir = sys.argv[2]
if not os.path.isdir(output_dir):
    os.makedirs(output_dir)

train_share = float(sys.argv[3])
test_in_other_share = float(sys.argv[4])

df = pd.read_excel(dataset_file)
df = df.sample(frac=1).reset_index(drop=True)

msk = np.random.rand(len(df)) < train_share

train = df[msk]
other = df[~msk]

msk_test = np.random.rand(len(other)) < test_in_other_share
test = other[msk_test]
other = other[~msk_test]

train.to_excel(output_dir + '/train.xlsx')
test.to_excel(output_dir + '/test.xlsx')
other.to_excel(output_dir + '/other.xlsx')
```

Файл utils/exists_ru/exists_ru_columns.py, набор характеристик для выделения:

```
CHARACTERISTICS_LIST = [
    'Наименование1', 'Размерность', 'Диаметр', 'Индекс нагрузки', 'Индекс скорости', 'Модификация',
'Ось установки',
    'Тип', 'Вылет, мм', 'Диаметр ступицы, мм', 'Модель', 'Отверстия', 'Размер', 'Материал', 'Цвет
диска',
    'Цвет производителя', 'Код цвета', 'Обод'
]

CHARACTERISTICS = { char_name : [char_name] for char_name in CHARACTERISTICS_LIST}

ATTRIBUTES = [
    'RunFlat', 'Шипы'
]

SIZES = ['Размерность', 'Размер']
```

Файл utils/krepmarket/krepmarket_columns.py, набор характеристик для выделения:

```
MATERIAL = ['Материал']
MATERIAL2 = ['Материал рукоятки']

TYPE_MAIN = ['Тип', 'Тип болта']
```

```
TYPE1 = ['Тип головки', 'Тип цепи', 'Тип крепления']
TYPE2 = ['Резьба', 'Тип Заклепки', 'Тип анкера', 'Вид зажима', 'Звено', 'Тип скоб']
TYPE3 = ['Наконечник', 'Тип Шплинта', 'Вид декорации']
TYPE_SIZE = ['Размер']

FORM = ['Форма', 'В комплекте']

TYPE_ATTRIBUTES = ['С наружными зубцами', 'Для бетона', 'Универсальный', 'Левая резьба',
                   'Серьга', 'Набор', 'Мелкий шаг резьбы', 'Клипса', 'Изоляции',
                   'Дюймовый крепеж', 'Со скобой', 'Оборудование', 'Высокопрочные',
                   'Хомут-стяжка', 'Антивандальные', 'Внутренняя резьба', 'Бортик',
                   'Для высоких нагрузок', 'Держатель труб', 'Насечка', 'Соединитель цепи',
                   'Устойчивость к УФ', 'С предохраняющей гайкой', 'Спиральная',
                   'Карабин', 'Асимметричный', 'С коушем', 'Открытый',
                   'С роликом', 'Защита от перегрева', 'Регулируемая скорость',
                   'Регулировка глубины строгания', 'Выборка четверти', 'Насадки в комплекте',
                   'Сменные сопла', 'Регулируемая температура', 'Клинья для молотка', 'Трещотка',
                   ]  # Yes / No values

MODEL = ['Модель', 'Стандарт', 'Бита']

COLOR = ['Цвет', 'Полоса']

LENGTH = ['Длина, мм', 'Длина', 'Длина, м', 'Длина шины, см']
WIDTH = ['Ширина', 'Ширина, мм', 'Ширина ножа, мм', 'Ширина режущей части, мм']
HEIGHT = ['Высота']
DIAMETER = ['Диаметр, мм', 'Диаметр резьбы, мм', 'Диаметр резьбы, М', 'Диаметр', 'Диаметр стержня,
мм']
DEPTH1 = ['Глубина строгания, мм']
DEPTH2 = ['Глубина выборки четверти, макс, мм']

SIZE_MIN = ['Минимальный размер, мм', 'Высота в сложенном состоянии', 'Минимальный диаметр, мм']
SIZE_MAX = ['Максимальный захват, мм', 'Максимальная рабочая высота', 'Максимальный диаметр, мм',
'Максимальный Размер, мм']

SIZE_OTHER = ['Посадочный квадрат', 'Размер скобы', 'Размер, мм', 'Размер гайки, М']

WEIGHT = ['Вес', 'Объем']

PHYS1 = ['Грузоподъемность', 'Производительность, г/мин', 'Грузоподъёмность на 1 единицу, кг',
         'Число оборотов, макс']
PHYS2 = ['Рабочая температура, макс, град.', 'Обороты, макс',
         'Максимальное усилие, Нм', 'Мах рабочая нагрузка, кг', 'Максимальная нагрузка']
PHYS3 = ['Расход воздуха, л/мин', 'Грузоподъёмность на пару, кг', 'Время нагрева, мин']

POWER = ['Мощность', 'Допустимый пусковой ток, А']

COUNT = ['Количество', 'Количество звеньев цепи', 'Количество ступеней', 'Количество клавиш']
COUNT2 = ['Количество ступеней в секции']
COUNT3 = ['Количество секций']


SUPPLIER = ['Производитель']
SUPPLIER_COUNTRY = ['Страна производитель']

PURPOSE = ['Применение']
DESTINATION = ['Назначение']

TARGET_CHARACTERISTICS = {
    'MATERIAL': MATERIAL,
    'MATERIAL2': MATERIAL2,
    'TYPE_MAIN': TYPE_MAIN,
    'TYPE1': TYPE1,
    'TYPE2': TYPE2,
    'TYPE3': TYPE3,
    'TYPE_SIZE': TYPE_SIZE,
    'FORM': FORM,
    'MODEL': MODEL,
    'COLOR': COLOR,
    'LENGTH': LENGTH,
    'WIDTH': WIDTH,
```

```
    'HEIGHT': HEIGHT,
    'DIAMETER': DIAMETER,
    "DEPTH1": DEPTH1,
    "DEPTH2": DEPTH2,
    "SIZE_MIN": SIZE_MIN,
    "SIZE_MAX": SIZE_MAX,
    "SIZE_OTHER": SIZE_OTHER,
    'WEIGHT': WEIGHT,
    'PHYS1': PHYS1,
    'PHYS2': PHYS2,
    'PHYS3': PHYS3,
    'POWER': POWER,
    'COUNT': COUNT,
    'COUNT2': COUNT2,
    'COUNT3': COUNT3,
    'SUPPLIER': SUPPLIER,
    'SUPPLIER_COUNTRY': SUPPLIER_COUNTRY,
    'PURPOSE': PURPOSE,
    'DESTINATION': DESTINATION
}

ORIGNAL_TO_AGGREGATED = { orig : aggr for aggr, chars in TARGET_CHARACTERISTICS.items() for orig in
chars }

CHARACTERISTICS = ['Материал', 'Материал рукоятки',
                   'Тип', 'Тип головки', 'Резьба', 'Наконечник', 'Тип Заклепки', 'Тип анкера', 'Тип
болта',
                   'Тип Шплинта',
                   'В комплекте', 'Форма', 'Вид декорации', 'Вид зажима', 'Тип цепи', 'Звено', 'Тип
крепления',
                   'Размер',
                   'Модель', 'Стандарт', 'Бита', 'Тип скоб',
                   'Цвет', 'Полоса',
                   'Длина, мм', 'Длина', 'Высота', 'Длина, м', 'Длина шины, см', 'Минимальный
размер, мм',
                    'Посадочный квадрат', 'Максимальный захват, мм', 'Размер скобы', 'Размер, мм',
                    'Диаметр, мм', 'Диаметр резьбы, мм', 'Диаметр резьбы, М', 'Диаметр',
                    'Минимальный диаметр, мм', 'Максимальный диаметр, мм', 'Диаметр стержня, мм',
                    'Глубина строгания, мм', 'Глубина выборки четверти, макс, мм',
                    'Максимальная рабочая высота', 'Высота в сложенном состоянии',
                    'Ширина', 'Ширина, мм', 'Размер гайки, М', 'Ширина ножа, мм',
                    'Максимальный Размер, мм', 'Ширина режущей части, мм',
                   'Вес', 'Объем',
        'Грузоподъемность', 'Мощность', 'Число оборотов, макс', 'Рабочая температура, макс, град.',
        'Расход воздуха, л/мин', 'Обороты, макс', 'Производительность, г/мин', 'Время нагрева, мин',
        'Максимальное усилие, Нм', 'Мах рабочая нагрузка, кг', 'Максимальная нагрузка',
        'Грузоподъёмность на 1 единицу, кг', 'Грузоподъёмность на пару, кг', 'Допустимый пусковой
ток, А',
        'Количество', 'Количество звеньев цепи', 'Количество ступеней', 'Количество ступеней в
секции',
         'Количество секций', 'Количество клавиш',
        'Производитель', 'Страна производитель', 'Применение', 'Назначение']

ATTRIBUTES = ['С наружными зубцами', 'Для бетона', 'Универсальный', 'Левая резьба',
                   'Серьга', 'Набор', 'Мелкий шаг резьбы', 'Клипса', 'Изоляции',
                   'Дюймовый крепеж', 'Со скобой', 'Оборудование', 'Высокопрочные',
                   'Хомут-стяжка', 'Антивандальные', 'Внутренняя резьба', 'Бортик',
                   'Для высоких нагрузок', 'Держатель труб', 'Насечка', 'Соединитель цепи',
                   'Устойчивость к УФ', 'С предохраняющей гайкой', 'Спиральная',
                   'Карабин', 'Асимметричный', 'С коушем', 'Открытый',
                   'С роликом', 'Защита от перегрева', 'Регулируемая скорость',
                   'Регулировка глубины строгания', 'Выборка четверти', 'Насадки в комплекте',
                   'Сменные сопла', 'Регулируемая температура', 'Клинья для молотка', 'Трещотка']

YES_VALUES = ['да']

NONE_CHAR = 'NONE_CHAR'
```

Файл utils/extract_result_utils.py, набор методов для выделения характеристик из названий товаров:

```
from allennlp.common import Params
from allennlp.data import DatasetReader
from allennlp.models import Archive, CrfTagger, Model
from allennlp.predictors import SentenceTaggerPredictor, Predictor

from layers.utils.data_reader import CustomDataReader
from layers.utils.splitter import CustomWordSplitter
from utils.dataset_normalizer import parse_prelabeled
import pandas as pd
import numpy as np
from utils.dataset_normalizer import get_title_token_positions
from utils.dataset_normalizer import get_labeled_tittle
from layers.utils.split_utils import prepare_and_split
from layers.utils.split_utils import prepare_stable
from utils.exists_ru.exists_ru_columns import CHARACTERISTICS, ATTRIBUTES
from layers.taggers import *
from layers.taggers.crf_with_f1 import *

params = Params.from_file('models/lstm.jsonnet')
model = Model.load(params, 'output/models/crf_lstm_characters_only_additional_split/exists_ru')

reader_params = params.pop('dataset_reader')
reader_type = reader_params.pop('type')
reader = DatasetReader.by_name(reader_type).from_params(reader_params)

predictor = SentenceTaggerPredictor(model, dataset_reader = reader)
predictor._tokenizer = CustomWordSplitter()

def get_tags(title):
    tokens = prepare_and_split(title)

    tag_logits = predictor.predict(title)['logits']
    tag_ids = np.argmax(tag_logits, axis=-1)
    tags = [model.vocab.get_token_from_index(i, 'labels') for i in tag_ids]

    return tokens, tags
```

Файл utils/extract_result.py, скрипт выделения характеристик из названий товаров:

```
from allennlp.common import Params
from allennlp.data import DatasetReader
from allennlp.models import Archive, CrfTagger, Model
from allennlp.predictors import SentenceTaggerPredictor, Predictor
```

```
from layers.utils.data_reader import CustomDataReader
from layers.utils.splitter import CustomWordSplitter
from utils.dataset_normalizer import parse_prelabeled
import pandas as pd
import numpy as np
from utils.dataset_normalizer import get_title_token_positions
from utils.dataset_normalizer import get_labeled_tittle
from layers.utils.split_utils import prepare_and_split
from layers.utils.split_utils import prepare_stable
from utils.exists_ru.exists_ru_columns import CHARACTERISTICS, ATTRIBUTES
from layers.taggers import *
from layers.taggers.crf_with_f1 import *

params = Params.from_file('models/lstm.jsonnet')
model = Model.load(params, 'output/models/crf_lstm_characters_only/exists_ru')

reader_params = params.pop('dataset_reader')
reader_type = reader_params.pop('type')
reader = DatasetReader.by_name(reader_type).from_params(reader_params)

predictor = SentenceTaggerPredictor(model, dataset_reader = reader)
predictor._tokenizer = CustomWordSplitter()

test_df = pd.read_excel('temp/datasets/exists_ru/exists_ru_test.xlsx')

def get_tags(title):
    tokens = prepare_and_split(title)
    token_positions = get_title_token_positions(prepare_stable(title), tokens)

    tag_logits = predictor.predict(title)['logits']
    tag_ids = np.argmax(tag_logits, axis=-1)
    tags = [model.vocab.get_token_from_index(i, 'labels') for i in tag_ids]

    return tags

parsing_result_df = pd.DataFrame(columns=['title', 'title_labeled_correct', 'title_labeled'] +
CHARACTERISTICS + ATTRIBUTES)
for index, row in test_df.iterrows():
    title = str(row['title'])
    title_labeled_correct = str(row['title_labeled'])
    result_row = {'title': title, 'title_labeled_correct': title_labeled_correct}

    tokens = prepare_and_split(title)
    token_positions = get_title_token_positions(prepare_stable(title), tokens)

    tag_logits = predictor.predict(title)['logits']
    tag_ids = np.argmax(tag_logits, axis=-1)
```

```
    tags = [model.vocab.get_token_from_index(i, 'labels') for i in tag_ids]

    positions = {}
    for i in range(len(tags)):
        tag = tags[i]
        if tag != 'NONE_CHAR':
            if tag not in positions:
                positions[tag] = []
            positions[tag].append((token_positions[i], token_positions[i] + len(tokens[i])))
            if tag not in result_row:
                result_row[tag] = ''
            result_row[tag] += ' ' + tokens[i]

    title_labeled = get_labeled_tittle(title, positions)
    result_row['title_labeled'] = title_labeled

    parsing_result_df = parsing_result_df.append(result_row, ignore_index=True)

parsing_result_df.to_excel('temp/datasets/exists_ru/exists_ru_result.xlsx')
```

Файл layers/utils/split_utils.py, набор методов для разделения названий товаров на слова:

```
import re

def replace_comma_in_numbers(s):
    return re.sub(r'(\d),(\d)', r'\1.\2', s)


def replace_dot_in_words(s):
    return re.sub(r'([^(\W|\d)] ?)\.( ?[^(\W|\d)])', r'\1 \2', s)


print(replace_comma_in_numbers('22,18'))
print(replace_dot_in_words('22.18'))
print(replace_dot_in_words('дер .полн'))


def replace_x_in_numbers(s):
    return re.sub(r'(\d ?)[xx]( ?\d)', r'\1*\2', s)


replace_x_in_numbers('22 x18')


def replace_slash_in_words(s):
    return re.sub(r'([^(\W|\d)] ?)/( ?[^(\W|\d)])', r'\1 \2', s)
```

```
def is_number_with_slash(s):
    return re.match(r'\d/\d', s) is not None


def replace_slash_in_numbers(s):
    return re.sub(r'(\d ?)/( ?\d)', r'\1 \2', s)


def surround_with_spaces(s, regex, char):
    s = re.sub(regex + r'(\w)', char + r' \1', s)
    s = re.sub(r'(\w)' + regex, r'\1 ' + char, s)
    return s

def surround_with_spaces_words(s, regex, char):
    return re.sub(r'([^(\W|\d)] ?)' + regex + r'( ?[^(\W|\d)])', r'\1 ' + char + r' \2', s)


def surround_with_spaces_numbers(s, regex, char):
    return re.sub(r'(\d ?)' + regex + r'( ?\d)', r'\1 ' + char + r' \2', s)

def separate_words_and_numbers(s):
    s = re.sub(r'([^(\W|\d)])(\d)', r'\1 \2', s)
    s = re.sub(r'(\d)([^(\W|\d)])', r'\1 \2', s)
    return s

def prepare_stable(s):
    s = s.lower()
    s = replace_comma_in_numbers(s)
    return s

def prepare_value(s):
    s = prepare_stable(s)

    s = surround_with_spaces_numbers(s, 'x', 'x')
    s = surround_with_spaces_numbers(s, 'x', 'x')
    s = surround_with_spaces(s, '/', '/')
    s = surround_with_spaces_words(s, r'\.', '.')

    s = surround_with_spaces(s, ';', ';')
    s = surround_with_spaces(s, ',', ',')
    #s = surround_with_spaces(s, r'\*', '*')
    #s = surround_with_spaces(s, '"', '"')
    #s = surround_with_spaces(s, '-', '–')
    s = separate_words_and_numbers(s)

    return s
```

```
def split(s):
    tokens = re.split(r'\s|"', s)
    return [t for t in tokens if t is not None and t != '']

def prepare_and_split(s):
    return split(prepare_value(s))

def parse_prelabeled(s):
    find_from = 0

    parts = []
    tags = []

    first_tag_start_index = 0
    while first_tag_start_index != -1 and first_tag_start_index < len(s):
        first_tag_start_index = s.find('<', find_from)
        if first_tag_start_index != -1:
            first_tag_end_index = s.find('>', first_tag_start_index)
            tag_name = s[first_tag_start_index + 1:first_tag_end_index]
            second_tag_start_index = s.find('</', first_tag_end_index + 1)
            second_tag_end_index = s.find('>', second_tag_start_index)

            value = s[first_tag_end_index + 1:second_tag_start_index]

            before_tag = s[find_from:first_tag_start_index]
            if len(before_tag) > 0:
                parts.append(before_tag)
                tags.append('NONE_CHAR')

            if len(value) > 0:
                parts.append(value)
                tags.append(tag_name)

            find_from = second_tag_end_index + 1

    if find_from < len(s):
        parts.append(s[find_from:])
        tags.append('NONE_CHAR')

    return parts, tags


def split_and_get_tags(parts, parts_tags):
    tokens = []
    tags = []

    for i in range(len(parts)):
```

```
        part_tokens = split(prepare_value(parts[i]))
        tag = parts_tags[i]
        for t in part_tokens:
            tokens.append(t)
            tags.append(tag)

    return tokens, tags
```

Файл layers/utils/splitter.py, класс разделения названий товаров на слова:

```
from typing import List
from allennlp.data.tokenizers.word_splitter import WordSplitter
from allennlp.data.tokenizers.token import Token

from layers.utils.split_utils import *


@WordSplitter.register('custom-word-splitter')
class CustomWordSplitter(WordSplitter):
    def split_words(self, sentence: str) -> List[Token]:
        return [Token(w) for w in split(prepare_value(sentence))]
```

Файл layers/utils/data_reader.py, описание класс объекта, читающего набор данных:

```
import pandas as pd
import numpy as np
from typing import List, Dict, Iterator
from allennlp.data import DatasetReader, TokenIndexer, Token, Instance
from allennlp.data.fields import TextField, SequenceLabelField, ArrayField
from allennlp.data.token_indexers import SingleIdTokenIndexer

from layers.utils.split_utils import *

@DatasetReader.register('custom-dataset-reader')
class CustomDataReader(DatasetReader):
    def __init__(self, token_indexers: Dict[str, TokenIndexer] = None) -> None:
        super().__init__(lazy=True)
        self.token_indexers = token_indexers or {"tokens": SingleIdTokenIndexer()}

    def text_to_instance(self, tokens: List[Token], tags: List[str] = None) -> Instance:
        sentence_field = TextField(tokens, self.token_indexers)
        fields = {"tokens": sentence_field}

        if tags:
            label_field = SequenceLabelField(labels=tags, sequence_field=sentence_field)
            fields["tags"] = label_field
```

```
        return Instance(fields)

    def _read(self, file_path: str) -> Iterator[Instance]:
        df = pd.read_excel(file_path)
        for index, row in df.iterrows():
            if pd.isna(row['title_labeled']):
                continue

            parts, parts_tags = parse_prelabeled(str(row['title_labeled']))
            sentence, tags = split_and_get_tags(parts, parts_tags)

            yield self.text_to_instance([Token(w) for w in sentence], tags)
```

Файл layers/utils/f1_support.py, набор методов для добавления макро-f1 метрики в стандартный модуль из библиотеки AllenNLP:

```
from typing import Dict, Optional, List, Any

from allennlp.data import Vocabulary
from allennlp.models import CrfTagger
from allennlp.modules import TextFieldEmbedder
from allennlp.common.checks import check_dimensions_match, ConfigurationError
from allennlp.data import Vocabulary
from allennlp.modules import Seq2SeqEncoder, TimeDistributed, TextFieldEmbedder
from allennlp.modules import ConditionalRandomField, FeedForward
from allennlp.modules.conditional_random_field import allowed_transitions
from allennlp.models.model import Model
from allennlp.nn import InitializerApplicator, RegularizerApplicator
import allennlp.nn.util as util
from allennlp.training.metrics import CategoricalAccuracy, SpanBasedF1Measure, F1Measure
import layers.utils.f1_support as f1_support

@Model.register("crf_tagger_f1")
class CrfTaggerWithMetric(CrfTagger):
    def __init__(self, vocab: Vocabulary,
                 text_field_embedder: TextFieldEmbedder,
                 encoder: Seq2SeqEncoder,
                 label_namespace: str = "labels",
                 feedforward: Optional[FeedForward] = None,
                 label_encoding: Optional[str] = None,
                 include_start_end_transitions: bool = True,
                 constrain_crf_decoding: bool = None,
                 calculate_span_f1: bool = None,
                 dropout: Optional[float] = None,
                 verbose_metrics: bool = False,
                 initializer: InitializerApplicator = InitializerApplicator(),
```

```
                 regularizer: Optional[RegularizerApplicator] = None) -> None:
        super().__init__(vocab,
                         text_field_embedder,
                         encoder,
                         label_namespace,
                         feedforward,
                         label_encoding,
                         include_start_end_transitions,
                         constrain_crf_decoding,
                         calculate_span_f1,
                         dropout,
                         verbose_metrics,
                         initializer,
                         regularizer)
        f1_support.add_f1(self)

    def get_metrics(self, reset: bool = False):
        return f1_support.get_metrics(self, reset)
```

Файл layers/taggers/simple_tagger_f1.py, класс, в котором для модуля предсказания классов элементов последовательности из библиотеки AllenNLP добавляется метрика макро F1:

```
from typing import Dict, Optional, List, Any

import numpy
from allennlp.models import SimpleTagger
from overrides import overrides
import torch
from torch.nn.modules.linear import Linear
import torch.nn.functional as F

from allennlp.common.checks import check_dimensions_match, ConfigurationError
from allennlp.data import Vocabulary
from allennlp.modules import Seq2SeqEncoder, TimeDistributed, TextFieldEmbedder
from allennlp.models.model import Model
from allennlp.nn import InitializerApplicator, RegularizerApplicator

import layers.utils.f1_support as f1_support

@Model.register("simple_tagger_f1")
class SimpleTaggerF1(SimpleTagger):
    def __init__(self, vocab: Vocabulary,
                 text_field_embedder: TextFieldEmbedder,
                 encoder: Seq2SeqEncoder,
```

```
                 initializer: InitializerApplicator = InitializerApplicator(),
                 regularizer: Optional[RegularizerApplicator] = None,
                 verbose_metrics = False) -> None:
        super().__init__(vocab, text_field_embedder, encoder, initializer, regularizer)

        self._verbose_metrics = verbose_metrics
        f1_support.add_f1(self)

    def get_metrics(self, reset: bool = False):
        return f1_support.get_metrics(self, reset)
```

Файл layers/taggers/crf_with_f1.py, класс, в котором для модуля предсказания классов элементов последовательности с использованием Марковских случайных полей из библиотеки AllenNLP добавляется метрика макро-F1:

```
from typing import Dict, Optional, List, Any

from allennlp.data import Vocabulary
from allennlp.models import CrfTagger
from allennlp.modules import TextFieldEmbedder
from allennlp.common.checks import check_dimensions_match, ConfigurationError
from allennlp.data import Vocabulary
from allennlp.modules import Seq2SeqEncoder, TimeDistributed, TextFieldEmbedder
from allennlp.modules import ConditionalRandomField, FeedForward
from allennlp.modules.conditional_random_field import allowed_transitions
from allennlp.models.model import Model
from allennlp.nn import InitializerApplicator, RegularizerApplicator
import allennlp.nn.util as util
from allennlp.training.metrics import CategoricalAccuracy, SpanBasedF1Measure, F1Measure
import layers.utils.f1_support as f1_support

@Model.register("crf_tagger_f1")
class CrfTaggerWithMetric(CrfTagger):
    def __init__(self, vocab: Vocabulary,
                 text_field_embedder: TextFieldEmbedder,
                 encoder: Seq2SeqEncoder,
                 label_namespace: str = "labels",
                 feedforward: Optional[FeedForward] = None,
                 label_encoding: Optional[str] = None,
                 include_start_end_transitions: bool = True,
                 constrain_crf_decoding: bool = None,
                 calculate_span_f1: bool = None,
                 dropout: Optional[float] = None,
                 verbose_metrics: bool = False,
                 initializer: InitializerApplicator = InitializerApplicator(),
```

```
                 regularizer: Optional[RegularizerApplicator] = None) -> None:
        super().__init__(vocab,
                         text_field_embedder,
                         encoder,
                         label_namespace,
                         feedforward,
                         label_encoding,
                         include_start_end_transitions,
                         constrain_crf_decoding,
                         calculate_span_f1,
                         dropout,
                         verbose_metrics,
                         initializer,
                         regularizer)
        f1_support.add_f1(self)

    def get_metrics(self, reset: bool = False):
        return f1_support.get_metrics(self, reset)
```

Файл cnn_ff_proj.jsonnet, описание конфигурации модели CNN-FeedForward:

```
local char_embedding_dim = 20;
local word_embedding_dim = 40;
local encoder_input_dim = word_embedding_dim;
local hidden_dim = 100;

local num_epochs = 40;
local patience = 10;
local batch_size = 100;
local learning_rate = 0.01;

{
    "dataset_reader": {
        "type": "custom-dataset-reader",
        "token_indexers": {
            "token_characters": { "type": "characters" }
        }
    },
    "model": {
        "type": "simple_tagger_f1",
        "text_field_embedder": {
            "token_characters": {
                "type": "character_encoding",
                "embedding": {
                    "embedding_dim": char_embedding_dim,
                },
```

```
            "encoder": {
                "type": "cnn",
                "embedding_dim": char_embedding_dim,
                "num_filters": 20,
                "output_dim": word_embedding_dim,
                "ngram_filter_sizes": [2, 3]
            }
        }
    },
    "encoder": {
        "type": "feedforward",
         "feedforward": {
            "input_dim": encoder_input_dim,
            "num_layers": 3,
            "hidden_dims": [hidden_dim, hidden_dim, hidden_dim],
            "activations": "relu"
        }
    }
},
"iterator": {
    "type": "basic",
    "batch_size": batch_size
},
"trainer": {
    "num_epochs": num_epochs,
    "optimizer": {
        "type": "adam",
        "lr": learning_rate
    },
    "patience": patience
}
}
```

Файл models/cnn_lstm_crf.jsonnet, описание конфигурации модели CNN-biLSTM-CRF:

```
local char_embedding_dim = 20;
local word_embedding_dim = 40;
local encoder_input_dim = word_embedding_dim;
local hidden_dim = 100;

local num_epochs = 40;
local patience = 10;
local batch_size = 100;
local learning_rate = 0.01;
```

```
{
    "dataset_reader": {
        "type": "custom-dataset-reader",
        "token_indexers": {
            "token_characters": { "type": "characters" }
        }
    },
    "model": {
        "type": "crf_tagger_f1",
        "text_field_embedder": {
            "token_characters": {
                "type": "character_encoding",
                "embedding": {
                    "embedding_dim": char_embedding_dim,
                },
                "encoder": {
                    "type": "cnn",
                    "embedding_dim": char_embedding_dim,
                    "num_filters": 20,
                    "output_dim": word_embedding_dim,
                    "ngram_filter_sizes": [2, 3]
                }
            }
        },
        "encoder": {
            "type": "lstm",
            "input_size": encoder_input_dim,
            "hidden_size": hidden_dim,
            "bidirectional": true
        }
    },
    "iterator": {
        "type": "basic",
        "batch_size": batch_size
    },
    "trainer": {
        "num_epochs": num_epochs,
        "optimizer": {
            "type": "adam",
            "lr": learning_rate
        },
        "patience": patience
    }
}
```

Файл models/cnn_lstm_proj.jsonnet, описание конфигурации модели CNN-biLSTM:

```
local char_embedding_dim = 20;
local word_embedding_dim = 40; # As cnn generate vector of dimension 40, this double lstm should
produce the same
local encoder_input_dim = word_embedding_dim;
local hidden_dim = 100;

local num_epochs = 40;
local patience = 10;
local batch_size = 100;
local learning_rate = 0.01;

{
    "dataset_reader": {
        "type": "custom-dataset-reader",
        "token_indexers": {
            "token_characters": { "type": "characters" }
        }
    },
    "model": {
        "type": "simple_tagger_f1",
        "text_field_embedder": {
            "token_characters": {
                "type": "character_encoding",
                "embedding": {
                    "embedding_dim": char_embedding_dim,
                },
                "encoder": {
                    "type": "cnn",
                    "embedding_dim": char_embedding_dim,
                    "num_filters": 20,
                    "output_dim": word_embedding_dim,
                    "ngram_filter_sizes": [2, 3]
                }
            }
        },
        "encoder": {
            "type": "lstm",
            "input_size": encoder_input_dim,
            "hidden_size": hidden_dim,
            "bidirectional": true
        }
    },
    "iterator": {
        "type": "basic",
        "batch_size": batch_size
    },
    "trainer": {
```

```
        "num_epochs": num_epochs,
        "optimizer": {
            "type": "adam",
            "lr": learning_rate
        },
        "patience": patience
    }
}
```

Файл models/lstm_ff_proj.jsonnet, описание конфигурации модели biLSTM-FeedForward:

```
local char_embedding_dim = 20;
local word_embedding_dim = 20;
local encoder_input_dim = 2 * word_embedding_dim;
local hidden_dim = 100;

local num_epochs = 40;
local patience = 10;
local batch_size = 100;
local learning_rate = 0.01;

{
    "dataset_reader": {
        "type": "custom-dataset-reader",
        "token_indexers": {
            "token_characters": { "type": "characters" }
        }
    },
    "model": {
        "type": "simple_tagger_f1",
        "text_field_embedder": {
            "token_characters": {
                "type": "character_encoding",
                "embedding": {
                    "embedding_dim": char_embedding_dim,
                },
                "encoder": {
                    "type": "lstm",
                    "input_size": char_embedding_dim,
                    "hidden_size": word_embedding_dim,
                    "bidirectional": true
                }
            }
        },
        "encoder": {
            "type": "feedforward",
```

```
            "feedforward": {
               "input_dim": encoder_input_dim,
               "num_layers": 3,
               "hidden_dims": [hidden_dim, hidden_dim, hidden_dim],
               "activations": "relu"
            }
        }
    },
    "iterator": {
        "type": "basic",
        "batch_size": batch_size
    },
    "trainer": {
        "num_epochs": num_epochs,
        "optimizer": {
            "type": "adam",
            "lr": learning_rate
        },
        "patience": patience
    }
}
```

Файл models/lstm_lstm_crf.jsonnet, описание конфигурации модели biLSTM-biLSTM-CRF:

```
// jsonnet allows local variables like this
local char_embedding_dim = 20;
local word_embedding_dim = 20;
local encoder_input_dim = word_embedding_dim * 2;
local hidden_dim = 100;

local num_epochs = 40;
local patience = 10;
local batch_size = 100;
local learning_rate = 0.01;

{
    "dataset_reader": {
        "type": "custom-dataset-reader",
        "token_indexers": {
            "token_characters": { "type": "characters" }
        }
    },
    "model": {
        "type": "crf_tagger_f1",
        "text_field_embedder": {
```

```
            "token_characters": {
                "type": "character_encoding",
                "embedding": {
                    "embedding_dim": char_embedding_dim,
                },
                "encoder": {
                    "type": "lstm",
                    "input_size": char_embedding_dim,
                    "hidden_size": word_embedding_dim,
                    "bidirectional": true
                }
            }
        },
        "encoder": {
            "type": "lstm",
            "input_size": encoder_input_dim,
            "hidden_size": hidden_dim,
            "bidirectional": true
        }
    },
    "iterator": {
        "type": "basic",
        "batch_size": batch_size
    },
    "trainer": {
        "num_epochs": num_epochs,
        "optimizer": {
            "type": "adam",
            "lr": learning_rate
        },
        "patience": patience
    }
}
```

Файл models/lstm_lstm_proj.jsonnet, описание конфигурации модели biLSTM-biLSTM:

```
// jsonnet allows local variables like this
local char_embedding_dim = 20;
local word_embedding_dim = 20; # As cnn generate vector of dimension 40, this double lstm should
produce the same
local encoder_input_dim = word_embedding_dim * 2;
local hidden_dim = 100;

local num_epochs = 40;
local patience = 10;
local batch_size = 100;
local learning_rate = 0.01;
```

```
{
    "dataset_reader": {
        "type": "custom-dataset-reader",
        "token_indexers": {
            "token_characters": { "type": "characters" }
        }
    },
    "model": {
        "type": "simple_tagger_f1",
        "text_field_embedder": {
            "token_characters": {
                "type": "character_encoding",
                "embedding": {
                    "embedding_dim": char_embedding_dim,
                },
                "encoder": {
                    "type": "lstm",
                    "input_size": char_embedding_dim,
                    "hidden_size": word_embedding_dim,
                    "bidirectional": true
                }
            }
        },
        "encoder": {
            "type": "lstm",
            "input_size": encoder_input_dim,
            "hidden_size": hidden_dim,
            "bidirectional": true
        }
    },
    "iterator": {
        "type": "basic",
        "batch_size": batch_size
    },
    "trainer": {
        "num_epochs": num_epochs,
        "optimizer": {
            "type": "adam",
            "lr": learning_rate
        },
        "patience": patience
    }
}
```

Файл models/lstm_pt_crf.jsonnet, описание конфигурации модели biLSTM-CRF:

```
// jsonnet allows local variables like this
local char_embedding_dim = 20;
local word_embedding_dim = 20;
local encoder_input_dim = word_embedding_dim * 2;
local hidden_dim = 100;

local num_epochs = 40;
local patience = 10;
local batch_size = 100;
local learning_rate = 0.01;

{
    "dataset_reader": {
        "type": "custom-dataset-reader",
        "token_indexers": {
            "token_characters": { "type": "characters" }
        }
    },
    "model": {
        "type": "crf_tagger_f1",
        "text_field_embedder": {
            "token_characters": {
                "type": "character_encoding",
                "embedding": {
                    "embedding_dim": char_embedding_dim,
                },
                "encoder": {
                    "type": "lstm",
                    "input_size": char_embedding_dim,
                    "hidden_size": word_embedding_dim,
                    "bidirectional": true
                }
            }
        },
    "encoder": {
        "type": "pass_through",
        "input_dim": encoder_input_dim
    }

    },
    "iterator": {
        "type": "basic",
        "batch_size": batch_size
    },
    "trainer": {
        "num_epochs": num_epochs,
        "optimizer": {
            "type": "adam",
```

```
            "lr": learning_rate
        },
        "patience": patience
    }
}
```

Файл models/train.sh – скрипт запуска обучения модели:

```
#!/usr/bin/env bash
```

#!/usr/bin/env bash

```
# 1 - model name
# 2 - dataset name
# 3 - model config path

# Train
rm -rf output/models/$2/$1/*
allennlp train $3 -s output/models/$2/$1 --include-package layers -o
'{"train_data_path":'"'"'temp/datasets/'$2'/train.xlsx'"'"',"validation_data_path":'"'"'temp/dataset
s/'$2'/test.xlsx'"'"'}'
```

Файл models/train_all.sh – скрипт запуска обучения всех используемых моделей:

```
#!/usr/bin/env bash

models/train.sh cnn_ff_proj exists_ru/067 models/cnn_ff_proj.jsonnet
models/train.sh cnn_pt_crf exists_ru/067 models/cnn_pt_crf.jsonnet
models/train.sh cnn_lstm_proj exists_ru/067 models/cnn_lstm_proj.jsonnet
models/train.sh cnn_lstm_crf exists_ru/067 models/cnn_lstm_crf.jsonnet

models/train.sh lstm_ff_proj exists_ru/067 models/lstm_ff_proj.jsonnet
models/train.sh lstm_pt_crf exists_ru/067 models/lstm_pt_crf.jsonnet
models/train.sh lstm_lstm_proj exists_ru/067 models/lstm_lstm_proj.jsonnet
models/train.sh lstm_lstm_crf exists_ru/067 models/lstm_lstm_crf.jsonnet

models/train.sh cnn_ff_proj krepmarket/067 models/cnn_ff_proj.jsonnet
models/train.sh cnn_pt_crf krepmarket/067 models/cnn_pt_crf.jsonnet
models/train.sh cnn_lstm_proj krepmarket/067 models/cnn_lstm_proj.jsonnet
models/train.sh cnn_lstm_crf krepmarket/067 models/cnn_lstm_crf.jsonnet

models/train.sh lstm_ff_proj krepmarket/067 models/lstm_ff_proj.jsonnet
models/train.sh lstm_pt_crf krepmarket/067 models/lstm_pt_crf.jsonnet
models/train.sh lstm_lstm_proj krepmarket/067 models/lstm_lstm_proj.jsonnet
models/train.sh lstm_lstm_crf krepmarket/067 models/lstm_lstm_crf.jsonnet
```

Файл models/eval.sh – скрипт оценки качества модели:

```
#!/usr/bin/env bash

# 1 - model name
# 2 - dataset name
# 3 -output file

mkdir -p $3/$2
allennlp evaluate output/models/$1/$2/ --include-package layers -o '{"model":{"verbose_metrics":
true}}' temp/datasets/$2/$2_train.xlsx --output-file $3/$2/$1_train.txt
```

```
allennlp evaluate output/models/$1/$2/ --include-package layers -o '{"model":{"verbose_metrics":
true}}' temp/datasets/$2/$2_test.xlsx --output-file $3/$2/$1_test.txt
allennlp evaluate output/models/$1/$2/ --include-package layers -o '{"model":{"verbose_metrics":
true}}' temp/datasets/$2/$2_validate_test.xlsx --output-file $3/$2/$1_validate_test.txt
```

Файл models/eval_all.sh – скрипт оценки проверки качества всех моделей:

```
#!/usr/bin/env bash

models/eval.sh cnn_ff_proj exists_ru output/eval
models/eval.sh cnn_pt_crf exists_ru output/eval
models/eval.sh cnn_lstm_proj exists_ru output/eval
models/eval.sh cnn_lstm_crf exists_ru output/eval

models/eval.sh lstm_ff_proj exists_ru output/eval
models/eval.sh lstm_pt_crf exists_ru output/eval
models/eval.sh lstm_lstm_proj exists_ru output/eval
models/eval.sh lstm_lstm_crf exists_ru output/eval
```