Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский
Академический университет Российской академии наук»
Центр высшего образования

Кафедра математических и информационных технологий

Геллер Марк Аркадьевич

Предсказание и анализ решения суда при помощи методов машинного обучения

Магистерская диссертация

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Омельченко А. В.

Научный руководитель:
Шпильман А. А.

Рецензент:
Тузова Е. А.

Санкт-Петербург
2017

SAINT-PETERSBURG ACADEMIC UNIVERSITY
Higher education centre

Department of Mathematics and Information Technology

Geller Mark

Applications of machine learning in court decision prediction and analysis

Graduation Thesis

Admitted for defence.
Head of the chair:
professor Alexander Omelchenko

Scientific supervisor:
Alexey Shpilman

Reviewer:
Ekaterina Tuzova

Saint-Petersburg
2017

Оглавление

Введение

В настоящее время всё больше систем принятия решений в нашей жизни подвергается автоматизации. Так, например, банки и страховые компании используют методы машинного обучения для оценки рисков при выдаче кредитов или страховок. Другим примером могут послужить букмекерские конторы, использующие подобные приемы для определения вероятностей победы той или иной команды.

Однако есть области, где решение все еще принимается человеком. Примером является судебная система, автоматизация которой в ближайшие годы кажется маловероятной ввиду важности выносимых ею вердиктов. Однако анализ принимаемых судом решений является достаточно актуальной задачей, потому что он может помочь юристам производить оценку рисков еще до начала судебного разбирательства, и даже давать рекомендации на основании имеющихся материалов дела. Но это в будущем, а в данной работе проводился анализ решений суда по уже завершенным процессам.

Решение суда состоит из четырех частей [18]: вводной, описательной, мотивировочной и резолютивной. Вводная и описательная части содержат общую информацию о сути иска и порядке разбирательства (место, время и т.д.). Резолютивная часть - содержит непосредственное решение суда. Наконец, в мотивировочной части решения суда описываются обстоятельства и доказательства выявленные тем или иным образом в ходе разбирательства, а также мотивы суда, согласно которым он признал их относящимися к делу. Есть основания полагать, что именно в мотивировочной части содержится большинство факторов, влияющих на финальный вердикт суда, поэтому она и была выбрана в качестве объекта исследования данной работы.

Таким образом, целью данной работы является разработка системы, позволяющей определять вклад слов и предложений мотивировочной части в финальное решение суда.

Для достижения описанной выше цели необходимо решить следующие задачи:

- Произвести сбор данных. Это является необходимым шагом для решения любых задач машинного обучения.
- Выбрать алгоритм способный определять вклад отдельных слов и предложений документа.
- Произвести обучение и оценку качества выбранного на предыдущем шаге алгоритма.
- Построить механизм постобработки данных, позволяющий при помощи обученной ранее модели определять вклады слов и предложений.
- Произвести визуализацию результатов.

В главе 1 описываются возможные подходы к решению задачи с точки зрения обработки естественных языков, а также рассматриваются их преимущества и недостатки. В главе 2 происходит описание процесса сбора и обработки данных, а также создания обучающей выборки. Глава 3 посвящена описанию и мотивации предлагаемой архитектуры, а также ее сравнению с другими методами. Наконец, глава 4 описывает механизм использования модели, построенной в главе 3, для определения и визуализации вкладов отдельных слов и предложений мотивировочной в финальный вердикт суда.

1. Обзор предметной области

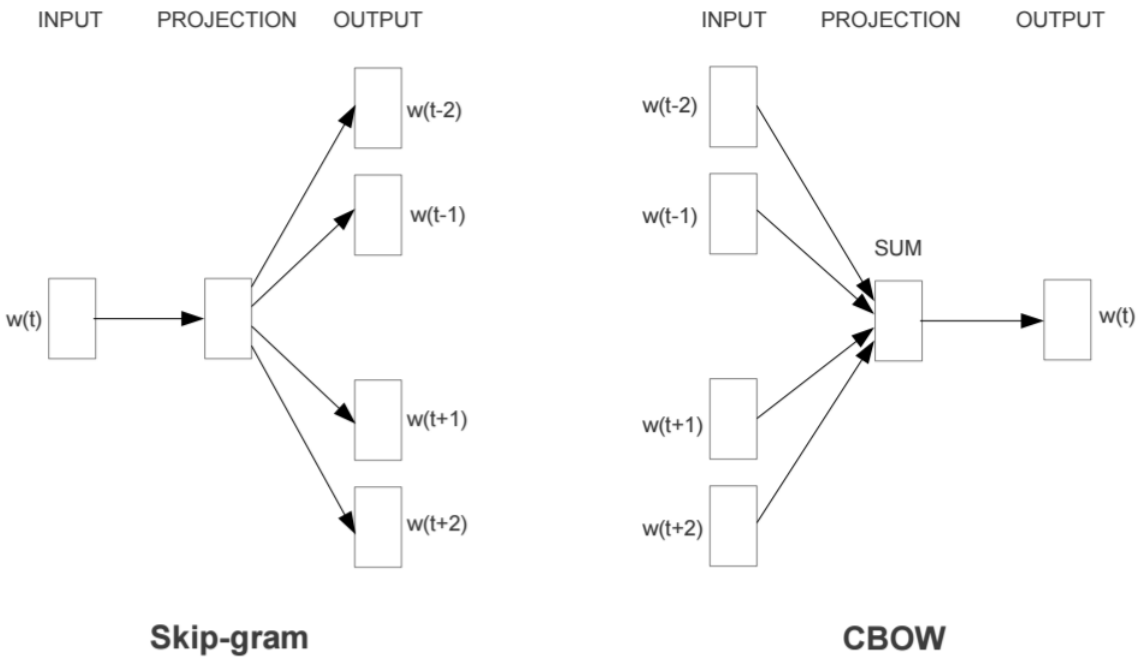
Существует ряд работ, в том или ином виде решающих задачу предсказания решения суда:

Первые работы фокусировались на построении детерминированных, легко интерпретируемых алгоритмов предсказания вердикта суда. Так, например, была продемонстрирована [3] эффективность использования решающих деревьев для предсказания вердикта Верховного суда США. При этом, в качестве признаков использовалась лишь общая информация о деле: Сведения о расположении и вердикте судов первой инстанции, гражданстве подсудимого и т.д. Важным достижением данной работы можно назвать экспериментальную демонстрацию факта улучшения качества принятия решений, по сравнению с экспертной группой.

Еще одной из самых известных работ в данной области является работа Н. Алетраса [1] в которой исследуется возможность предсказания решения суда основываясь на его содержании. В рамках этой статьи, в качестве источника данных использовались материалы дел Европейского Суда по Правам Человека: все части решения суда, а также текст статьи являющейся предметом разбирательства. Одним из основных достижений этой работы можно назвать демонстрацию применимости методов обработки естественных языков в применении к решению задачи предсказания вердикта суда.

С другой стороны, можно заявлять, что задача определения вероятного решения суда на основе мотивировочной части решения схожа по сути с задачей определения настроения документа на основании его содержания (Sentiment Analysis из области обработки естественных языков), ведь в обоих случаях суть заключается в бинарной классификации текста на основании его текста. Поэтому стоит обратить внимание на обширное число методов решения данной задачи.

Классическое обучение с учителем. При обучении с учителем алгоритмы машинного обучения определяют зависимости в размеченном наборе данных. Для этого создается большой набор специально сконструированных из предложений признаков, который затем используется для обучения одного из алгоритмов классификации (например SVM, или Наивной Байесовской модели). Большинство работ по этим методам фокусируются на построении вышеупомянутых признаков. Так, например, было показано [9], что использование последовательностей из заданного количества подряд идущих слов (N-gram) в качестве признаков позволяет добиться хороших результатов. Именно такой подход использовался в упомянутой ранее работе о предсказании решений ЕСПЧ [1].



(a) Структура модели Skip-gram (b) Структура модели CBOW

Рис. 1: Модели Word2Vec

Словари окраски. При решении задачи Sentiment Analysis многие алгоритмы используют так называемые словари окраски (Sentiment Lexicons [11]). Эти словари, по сути представляют собой таблицы отображения слов в некоторую численную величину описывающую окраску слова. Есть, также, работы показывающие эффективность расширения словарей коллокациями, или даже биграммами [12]. Часто эти словари создаются вручную, однако есть способы их построения на основе различной метаинформации. Однако данный подход плохо применим к решению задач юриспруденции, т.к. в формальных текстах не принято использовать слова, явно отображающие позицию автора.

Word2Vec [16]. В 2013 году был предложен способ векторизации слов основанный на нейронных сетях. В отличие от описанных ранее методов, данный подход строит семантическое представление слова. Для этого используется предположение о том, что контекст использования слов содержит информацию об их смысле. Для обучения модели, вокруг каждого слова выбирается контекст – набор окружающих его слов в окне заданной ширины (размер окна является гиперпараметром модели). Есть два основных способа использования контекста для построения отображения:

- модель Skip-gram (рисунок 1.a). Перед обучениям всем словам корпуса присваиваются порядковые номера. На стадии обучения на вход нейронной сети пода-

ется onehot encoding слова w(t) – вектор, все компоненты которого равны нулю, кроме той, чей номер совпадает с номером слова: она равна единице. Данное представление является входными данными для полносвязного слоя с линейной активацией. Его задачей является построение искомого скрытого представления слов. Наконец, упомянутый полносвязный слой связан с выходным слоем, которой при помощи softmax активации определяет вероятности принадлежности слов контексту w(t). После обучения, можно использовать матрицу весов скрытого слоя для получения векторизации слов.

- модель непрерывного мешка слов (CBOW), рисунок 1.б. Данный подход обладает архитектурой, схожей с моделью Skip-gram, однако противоположен по сути: он ставит своей задачей предсказание самого слова w(t) на основании onehot encoding контекста

Таким образом Word2Vec создает некоторое пространство семантического описания языка. При этом интересным является тот факт, что данное пространство обладает некоторой внутренней структурой: так, например, близкие по смыслу слова, зачастую, отображаются в близкие точки пространства. Более того, в данном пространстве наблюдаются некоторые метрические соотношения: вектор разности слов "Россия" и "Москва" близок к вектору разности слов "Франция" и "Париж". Данный факт позволяет говорить об осмысленности использования средних Word2Vec векторов предложений в качестве их семантического описания.

Используя модель Word2Vec можно построить классификатор, решающий задачу Sentiment Analysis при помощи простого мешка слов: документ представляется средним значением Word2Vec его слов, после чего это представление можно использовать для применения классического обучения с учителем.

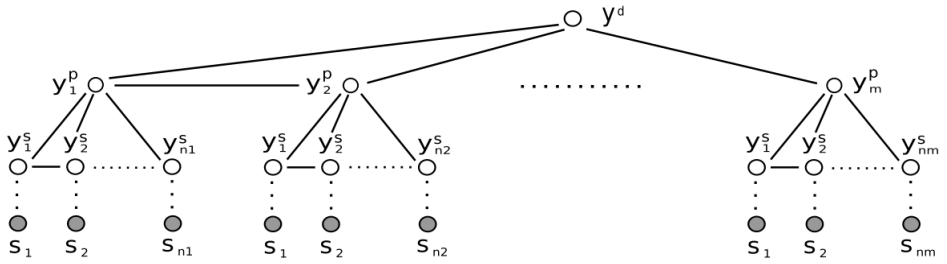


Рис. 2: Классические рекуррентные сети

Структурные модели . Описанные выше подходы к решению задачи Sentiment Analysis используют лишь один из уровеней абстракции: слов, предложений или документов. Однако одно и то же слово в разных контекстах может иметь разную окраску.

Поэтому основной целью структурных моделей[14] является создание многоуровневой модели, способной хотя бы частично отражать сложные семантические связи в тексте. Для достижения этой цели документ представляется графом, где решение об окрасе каждого предложения зависит от окраса его соседей, а также окраса всего документа. Т.е. выходом классификатора является набор решений $(y^d, y_1^s, y_2^s, ..., y_n^s)$, тут y^d — это окрас документа,а y_i^s — это окрас соответствующего предложения. По аналогии, данные модели могут быть расширены до уровня анализа слов, путем добавления в граф соответствующих им вершин и ребер (рис. 2).

Deep Learning. Нейронные сети показали выдающиеся результаты в ряде задач обработки естественных языков(NLP). Например они лежат в основе лучших на сегодняшний день решений не только задачи Sentiment Analysis, но и таких задач, как машинный перевод и семантический анализ текста. В частности, это стало возможно благодаря использованию рекуррентных слоев (рисунок 3) в архитектуре нейронных сетей (RNN) [10]. Данный метод позволяет учесть информацию о временной зависимости данных. В простейшем случае об RNN слое можно думать как о нейроне, который хранит в своем внутреннем состоянии информацию о всех увиденных им данных. Для этого его внутреннее состояние в момент времени t добавляется ко входным данным в следующий момент времени t + 1. Данный механизм крайне важен при решении задач NLP, так как наличие внутреннего состояния позволяет учитывать последовательности слов в предложениях и предложений в документе.

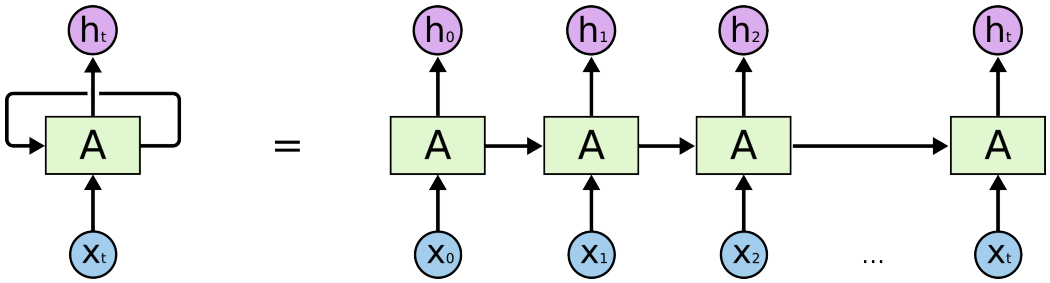


Рис. 3: Классические рекуррентные сети

С другой стороны, сам факт того, что RNN ячейки хранят в себе информацию обо всех предыдущих моментах времени, может оказаться не очень полезен: так, в длинных последовательностях данных внутренние представление смешивается и информация о ранних точках теряется. И чем длиннее цепочка данных, тем сильнее зашумление внутреннего состояния. Чтобы решить эту проблемы был предложен механизм LSTM ячеек [7], рисунок 4. Они построены таким образом, чтобы не было проблем с хранением информации, встреченной значительное число шагов назад. По

сути – это схожая с RNN ячейка, передающая во времени свое внутреннее состояние, однако механизм этой передачи более сложен. Основными структурными деталями архитектуры LSTM ячейки являются внутреннее состояние, которое хранится отдельно от входных данных, а также механизмы добавления и удаления информации в это внутреннее состояние. Альтернативой механизму LSTM является схожий по сути механизм GRU, являющийся более современным аналогом.

Также стоит упомянуть такое понятие, как двунаправленные рекуррентные слои: RNN и LSTM в качестве внутреннего состояния хранят информацию о предыдущих данных. Однако при анализе текста стоит учитывать контекст, окружающий конкретное слово с обеих сторон. Чтобы преодолеть данное ограничение, можно использовать две LSTM ячейки: в одну подавать данные в прямом порядке, в другую – в обратном, а затем использовать конкатенацию их выходов в качестве конечного результата. Данный подход хорошо проявил себя в различных задачах NLP и применяется практически при любой обработке текста с помощью рекуррентных нейронных сетей.

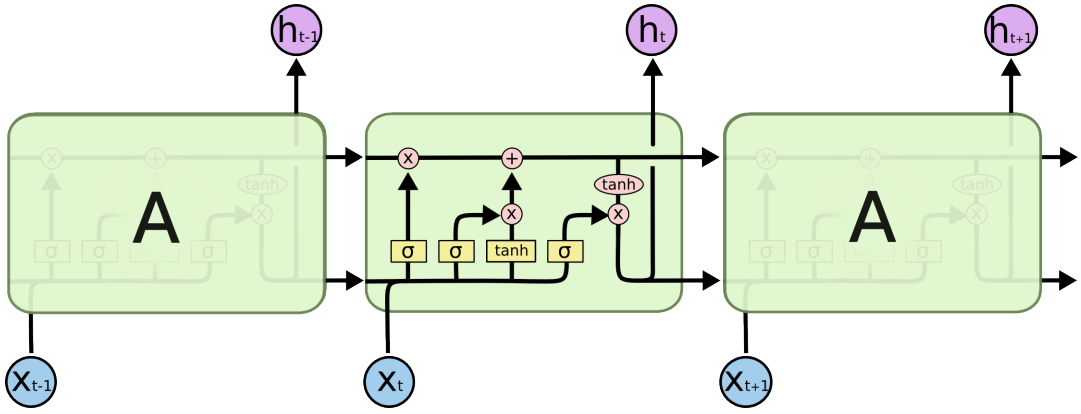


Рис. 4: LSTM ячейка

С учетом цели данной работы стоит отметить, что внутренние механизмы принятия решений нейронными сетями плохо поддаются интерпретации, что затрудняет определение вкладов отдельных слов и предложений. Однако в 2014 году был предложен механизм "внимания" для рекуррентных нейронных сетей[4] (если точнее - для решения конкретной задачи: машинного перевода). Он позволяет обойти данное ограничение, о чем будет подробнее рассказано в главе 3. Также, отметим, что в большинстве случаев, при работе с нейронными сетями в области обработки естественных языков в качестве стандартного подхода к векторизации слов используется Word2Vec, однако есть и другие способы, которые, опять же, будут рассмотрены в главе 3.

2. Данные

Как правило, большинство задач машинного обучения требуют наличия данных для первоначального обучения алгоритма. Этот набор данных — обучающая выборка.

Процесс построения обучающей выборки можно условно разбить на несколько этапов:

- Использование первоисточников с целью получения исходных "сырых" данных
- Предварительная обработка полученной информации
- Балансировка выборки и разбиение на обучающие множества

2.1. Первоначальный сбор данных

В качестве источника сырых данных была выбрана картотека арбитражных дел kad.arbitr.ru — единственный ресурс, на котором предоставлены дела из всех судов РФ. Данный портал предоставляет пользователям возможность поиска документов по ряду параметров, однако он плохо приспособлен для машинного сбора информации.

Для автоматизации извлечения данных была написана программа на языке Python имитирующая действия пользователя на вышеупомянутом ресурсе. В рамках этой задачи было необходимо учесть ряд ограничений:

- Результатом поисковой выдачи являются ссылки на карточки дел, содержащих все релевантные документы, а также дополнительную метаинформацию. Потому сбор данных проводился в два этапа: на первом этапе, при помощи поисковых запросов, составлялся список дел, а на втором происходил сбор документов по конкретному делу. Также стоит отметить, что данный сервис не предоставляет возможности определить, какой из документов является мотивировочной частью решения суда.
- Размер документов доступных по одному поисковому запросу сравнительно мал: 10 страниц по 20 документов. Чтобы обойти данное ограничение необходимо пользоваться непрерывным уточнением поискового запроса.
- Регулярно случаются сбои сервиса, что приводит к недоступности части дел.
- Частота запросов с одного ip-адреса сильно ограничена. Этот факт приводит к существенному замедлению сбора данных.

Таким образом была собрана информация примерно о 10600 дел, решения по которым были приняты в промежуток с 1 октября по 31 декабря 2016 года. Для каждого дела извлекались все относящиеся к нему документы, финальный вердикт суда, а

также информация о паре вид/категории спора — по сути описание объекта спора. Например вид – "экономические споры по административным правонарушениям", категория – "споры о неисполнении или ненадлежащем исполнении обязательств по договорам страхования". Всего в собранных данных представлено 4 пары вида/категории.

2.2. Предобработка данных

В первую очередь стоит отметить, что все материалы дел находились в формате PDF, поэтому первым необходимым шагом стала их конвертация в текстовый формат. Для этого использовалась утилита pdftotext, которая показала наилучшие результаты при распознавании текста кириллического алфавита.

Следующим шагом стало выделение мотивировочных частей решения среди всех имеющихся документов. Для решения это задачи было проведено изучение формата составления мотивировочных частей, что позволило составить набор правил, определяющий подходящие документы.

Далее, необходимо было разбить документ на предложения, а их, в свою очередь, разбить на слова. Для этого использовались соответствующие механизмы платформы NLTK.

Исследование содержания мотивировочных частей решения суда показало, что иногда в последних предложениях документа, в дополнение к фактам разбирательства, судьи указывают еще и их финальный вердикт. Чтобы учесть этот факт, в качестве входных данных алгоритма использовалась только часть документа . Так, средняя длина мотивировочной части решения составляла 72 предложения, в то время как в документы обучающей выборки добавлялись лишь первые 40.

Наконец, многие алгоритмы машинного обучения требуют для корректной работы фиксированную размерность входных данных. Поэтому необходимо унифицировать длину предложений. Было приняте решения использовать в качестве представления предложений последовательности из 75 слов (что в полтора раза больше средней длины предложения). Более короткие предложения были дополнены пустыми словами.

2.3. Балансировка и разбиение обучающих множеств

Наконец, для решения задач машинного обучения необходимо было провести разбиение данных на обучающую (Train), проверочную (Validate), и тестовую(Test) выборки. Из п. 2.1 известно, что данные были представлены 4 парами вид/категория. Из них в обучающую выборку попали лишь 3, а четвертая была выделена в отдельное тестовое множество, для демонстрации качества работы алгоритма на документах, отличающихся по сути от обучающей выборки.

Некоторые метрики оценки качества работы алгоритмов машинного обучения плохо поддаются сравнению в случае несбалансированных выборок, поэтому на этапе сбора данных было необходимо уравновесить количества удовлетворенных и отклоненных исков в рамках каждой пары вид/категория.

Т.о было произведено разбиение данных на 4 группы:

- Обучающая выборка — 8000 документов из 3 пар вида/категории
- Проверяющая выборка — 1000 документов аналогичных материалам обучающей выборки
- Тестовая выборка **same** — 1000 документов аналогичных материалам обучающей выборки
- Тестовая выборка **different** — 600 документов документов из четвертой пары вида/категории

3. Обучение модели

В рамках данной работы в качестве основного подхода к решению задачи предсказания вердикта суда было выбрано глубинное обучение. Если точнее, нас интересовали подходы, способные определять вклад отдельных слов/предложений. Для этого предлагается использовать механизм внимания.

3.1. Механизм Внимания

Суть механизма Attention заключается в том, чтобы научить нейронную сеть "обращать внимание" на различные участки входных данных в зависимости от контекста.

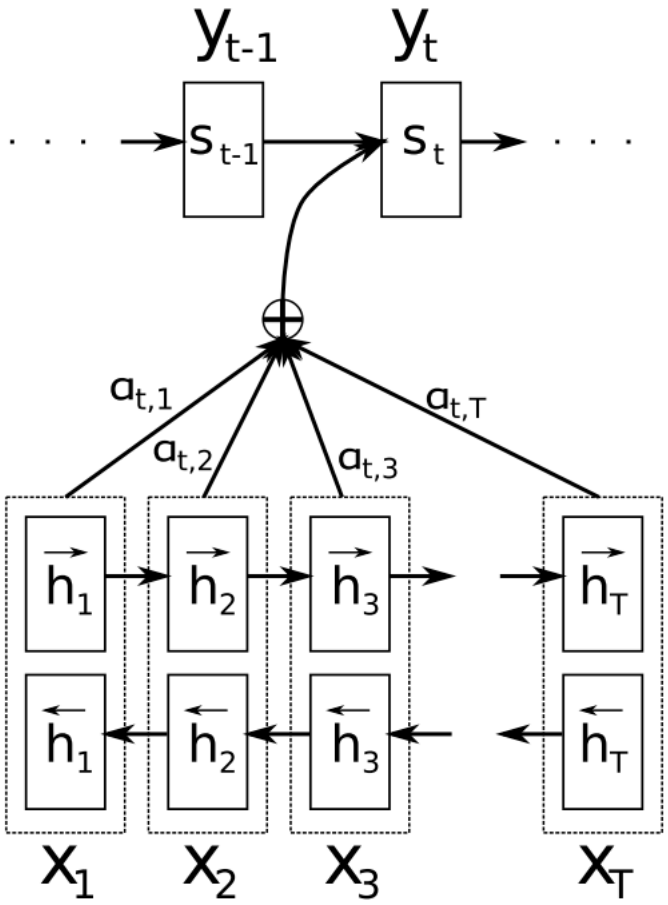


Рис. 5: схема механизма внимания

Идея состоит в том, чтобы добавить в нейронную сеть отдельный блок (Attention unit), задачей которого является генерация распределения весов входных данных на основании последовательности исходных точек(а также контекста).

В общем случае архитектура Attention unit может быть сколь угодно сложной [15]. Однако в большинстве случаев(особенно при малом количестве данных) можно

пользоваться простейшей моделью:
Пусть у нас есть некоторая последовательность точек h_i: это может быть, например, результат применения двунаправленного рекуррентного слоя ко входным данным сети. Тогда их комбинированное представление s_t можно получить с помощью следующих формул:

$$u_i = \tanh(Wh_i + b)$$
$$\alpha_{ti} = \frac{\exp(u_i^T u_c)}{\sum_i \exp(u_i^T u_c)}$$
$$s_t = \sum_i \alpha_{ti} h_i$$

Здесь:

- W, b - внутренняя матрица весов и смещение внутри механизма внимания.
- α_{ti} - веса входных данных, с точки зрения Attention unit.
- u_c - вектор контекста. По сути - высокоуровневое представление фиксированного запроса "какие точки входных данных обычно несут полезную информацию". Он является одним из обучаемых параметров сети.
- s_t - полученное представление входных данных. По-сути — их взвешенная сумма .

Стоит подчеркнуть тот факт, что из построения механизма внимания следует семантика величины α_{ti} - это мера важности отдельных элементов последовательности данных.

Также важно отметить, что в области обработки естественных языков, добавление к рекуррентной сети механизма внимания приводит к улучшению качества ее работы, что свидетельствует об осмысленности весов α_{ti}.

Код простейшей ячейки внимания приведен в приложении А.

3.2. Hierarhical Attention Networks

В 2016 году была предложена архитектура Hierarhical Attention Network [6] (в дальнейшем HAN), которая показала наилучшие результаты в решении задачи Sentiment Analysis на нескольких наборах данных.

На рисунке выше показана архитектура данной сети, представляющая из себя двухуровневую систему:

- Первый уровень состоит из пары word encoder + word attention. Его задачей стоит создание внутреннего представления предложений на основе их слов. Для этого, внутри word encoder слова векторизуются при помощи word2vec ($w_i j$) и

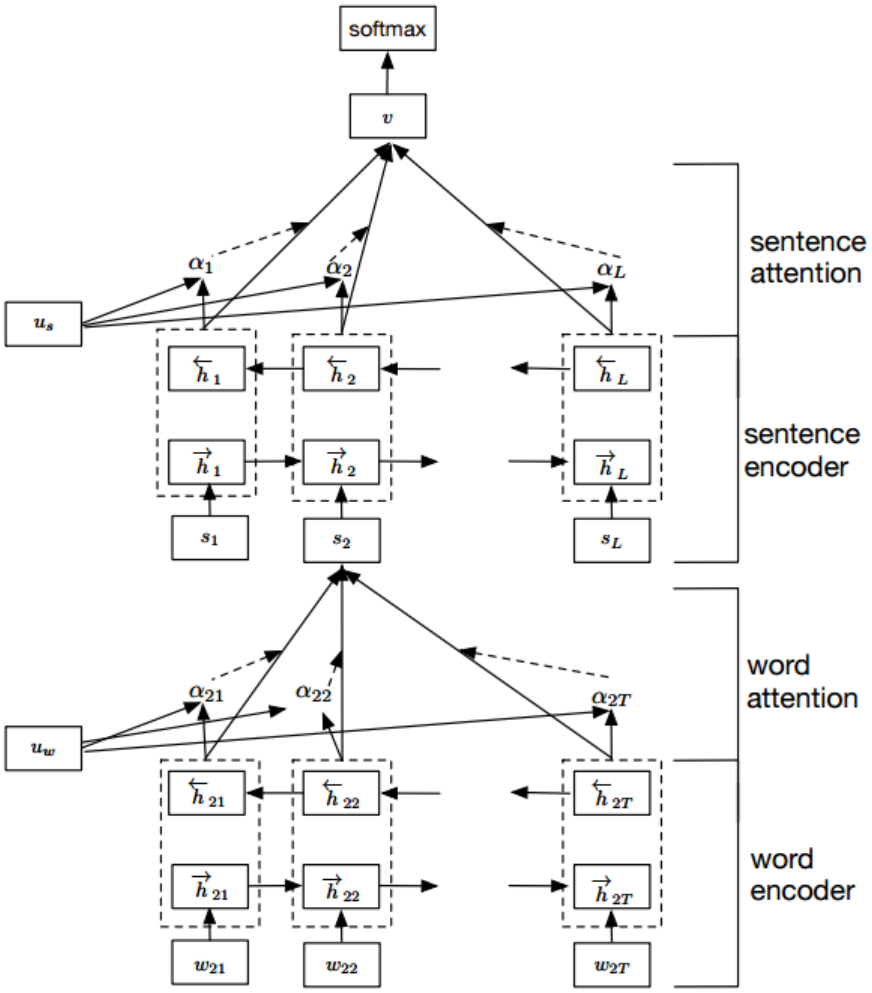


Рис. 6: архитектура Hierarhical Attention Network

подаются на вход двунаправленному GRU слою, который создает представление контекста каждого слова (h_ij). После чего Word Attention unit — реализующий механизм внимания — генерирует вклады α_{ij} для подсчета взешеной суммы s_i — внутреннего представления предложения i.

- Структура второго уровня практически идентична первому: на входе он имеет последовательность весов предложений s_i, которые подаются в sentence encoder принцип работы которого в точности эквивалентен word encoder из первого уровня. Наконец, Sentence Attention unit использует представления предложений из sentence encoder для подсчета их взвешенной суммы — то есть представления документа.

Внутреннее представление документа связано с финальным слоем, описывающий механизм принятия решения нейронной сетью. В простейшем варианте он представляет из себя полносвязный слой с двумя нейронами и функцией softmax в качестве

активационной: значение на выходе каждого из нейронов соответствует принадлежности документа одному из двух классов.

3.3. Character level embedding

Из за специфики стилистики исследуемых документов (решения суда), использование предобученной модели word2vec не представляется возможным. Поэтому в качестве word2vec необходимо использовать модель обученную на описанных ранее данных. В то же время, и при всех достоинствах оригинальной модели HAN [6] стоит отметить, что ее обучение производилось на достаточно большом количестве данных на английском языке, в то время как данных, описанных в главе 2 достаточно мало, и они на русском языке (морфологически более богатом). Из совокупности этих обстоятельств следует, что векторизация слов при помощи word2vec может быть не оптимальной в данном случае. Чтобы учесть этот факт мы предлагаем воспользоваться подходами к векторизации слов на уровне символов (character level embedding)

Char-CNN [19] — модель векторизации текста при помощи сверточных нейронных сетей. Изначально данный подход использовался для векторизации предложений, однако в 2016 году была продемонстрирована [2] его эффективности и для получения представления слов.

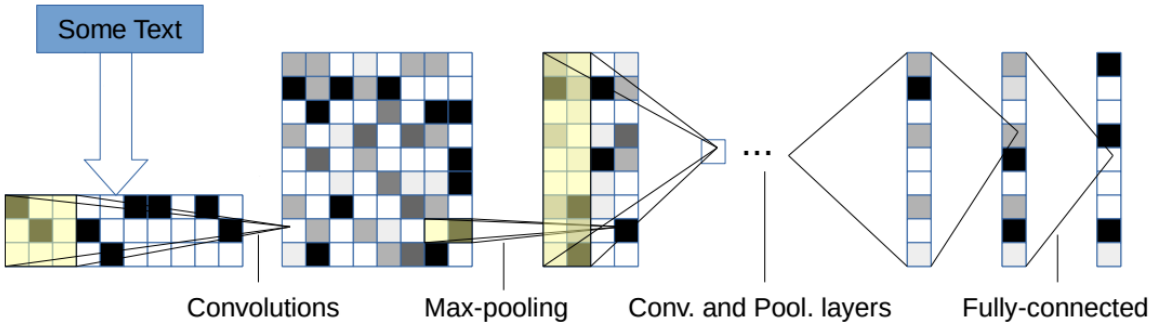


Рис. 7: архитектура Char-CNN

По своей структуре Char-CNN схожа со сверточными сетями из области обработки изображений: каждый символ представляется в виде onehot encoding, что преобразует слово в разреженную матрицу к которой, впоследствии, применяется последовательность из двух сверточных и Max-Pool слоев соединенных с полносвязным слоем для получения необходимой векторизации.

Код модели Char-CNN можно найти в приложении B

Char2Word [5] — альтернативная модель векторизации слов на основании их символов. В отличие от Char-CNN данный подход рассматривает слово как последовательность символов, в следствие чего предлагается использовать рекуррентные нейронные сети для построения представлений. В оригинальной статье архитектура Char2Word представляла из себя просто двухуровневую нейронную сеть:

- Character Lookup table - обучаемая таблица векторизации символов. Она позволяет получить векторное представление одного символа (на практике это работает лучше, чем onehot encoding. Это можно обосновать тем, что похожие в использовании символы отображаются в похожие вектора). Внутренний механизм работы этого слоя может быть описан полносвязным слоем, которому на вход подается onehot encoding символов.

- Bidirectional LSTM слой, суть которого состоит в составлении представления о слове используя последовательность его символов.

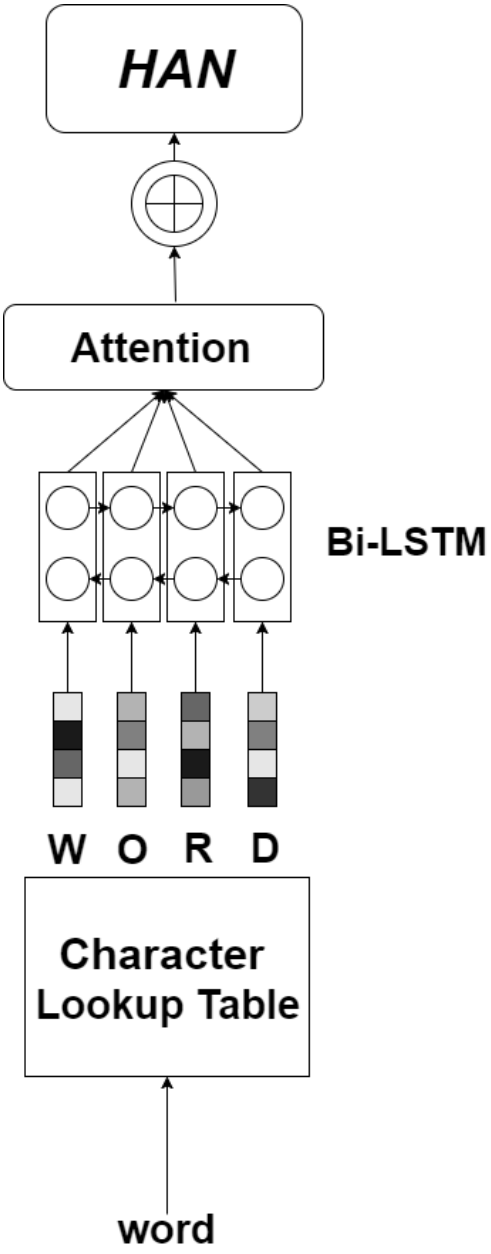


Рис. 8: Модифицированная архитектура Character2Word

Данный подход хорошо показал себя для решения задачи определения частей речи в морфологически богатых языках. В рамках данной работы мы предлагаем дополнить эту архитектуру с дополнительным механизмом внимания на уровне символов. Модифицированная таким образом модель нейронной сети (назовем ее AChar2Word) показана на рисунке 8.

Было произведено сравнение качества работы нейронной сети при использовании различных способов векторизации слов. В том числе исследовались комбинации различных подходов, где конечное представление получалось при помощи конкатенации различных векторизаций. Ниже приведена таблица сравнения полученных точностей

работы алгоритма на обеих тестовых выборках:

Таблица 1: Сравнение качества работы векторизаций слов

	Same	Different
Word2Vec	95.69	91.31
Char-CNN	91.29	86.12
Char2Word	92.12	87.67
AChar2Word	92.37	87.49
Word2Vec+Char-CNN	96.05	92.03
Word2Vec+Char2Word	96.81	92.86
Word2Vec+AChar2Word	97.17	**93.92**
Word2Vec+AChar2Word+Char-CNN	**97.22**	89.15

Стоит отметить, что для каждого из методов векторизации приведены наилучшие результаты, полученные настройкой гиперпараметров и внутренней архитектуры. Так, например, настраивалась модель сверточной сети в Char-CNN. Лучший результат был достигнут при использовании двух уровней свертки + Max-Pool. На первом уровне свертка велась с окном 5, на втором — с окном 3. В обоих слоях число фильтров равнялось 64. Выбор маленьких окон свертки в данном случае вполне разумен, т.к. длина слов равнялась 15 (механизм дополнения коротких слов аналогичен построению последовательностей слов в предложениях из главы 2).

Из таблицы сравнения качества работы различных способов векторизации можно сделать следующие выводы:

- Модель Char-CNN плохо подходит для решения данной задачи. Это следует из худшей ее производительности по сравнению с другими методами. Это вполне коррелирует с результатами оригинальной статьи[19], где прирост к эффективности проявлялся только на больших объемах данных.

- Использование только модели векторизации на основании символов хуже подходит для решения задачи Sentiment Analysis, чем Word2Vec. Это кажется логичным, т.к. для определения настроения контекст слова важнее его содержания.

- Добавление механизма внимания в модель Char2Word не ухудшает качества ее работы, а в совокупности с Word2Vec только улучшает.

- Использование комбинированного представления Word2Vec и AChar2Word является более эффективным по отношению к каждому из методов по отдельности. Это можно объяснить тем, что Word2Vec представляет собой семантическое описание слова, а AChar2Word - синтаксическое. Поэтому совместное описание является более полным. Так, например, из-за малого объема обучающей выборки Word2Vec может неправильно передавать смысл слова, но синтаксической

информации (о части речи) из AChar2Word может оказать достаточно для определения эмоционального окраса.

- Представление, объединяющее все три модели показывает хорошие результаты на данных того же рода, что и обучающая выборка, однако сильно проигрывает в производительности на другом типе данных. Это можно объяснить наличием переобучения.

Таким образом в качестве финально архитектуры векторизации слов была выбрана комбинация подходов Word2Vec и AChar2Word, продемонстрированная на рисунке 9. Такую модификацию архитектуры обозначим как WC2V-HAN. Ее реализацию можно найти в приложении C

3.4. Оценка качества

Необходимо произвести оценку качества работы описанной выше архитектуры. Так же, как и в случае сравнения способов векторизации слов, в роле метрики была выбрана точность. Приведем краткие описания сравниваемых алгоритмов:

- **Naive Bayes (NB)** – классический алгоритм классификации, использующий вероятности вероятности принадлежности слов одному из классов для определяющая настроения документа [13]. В данном случае в качестве векторизации слов был использован TF-IDF.

- **Bag of Words (Bow)** – модель, суть которой заключается в построении семантического вектора документа, описываемого средним значением Word2Vec его слов, с последующей классификации при помощи SVM. Выбор данного алгоритма, так как и NB, обусловлен тем, что обученной моделью можно пользоваться для определения настроений/вкладов отдельных слов и предложений.

- **SVM + N-gram (SVM)** – еще одна модель, использующая SVM в качестве классификатора, но теперь с механизмом N-gram в качестве признаков. Данный алгоритм использовался в работе о предсказании решений Европейского Суда по Правам Человека [1]

- **LSTM** – нейронная сеть состоящая из двух двунаправленных LSTM слоев. Это отображает HAN без механизмов внимания.

- **HAN** – оригинальная архитектура HAN, описанная в п 3.2.

- **WC2V-HAN** - предлагаемая в данной работе архитектура.

В таблице 2 приведено сравнение точностей работы описанных выше алгоритмов на обеих тестовых выборках. Из этих результатов можно сделать следующие выводы:

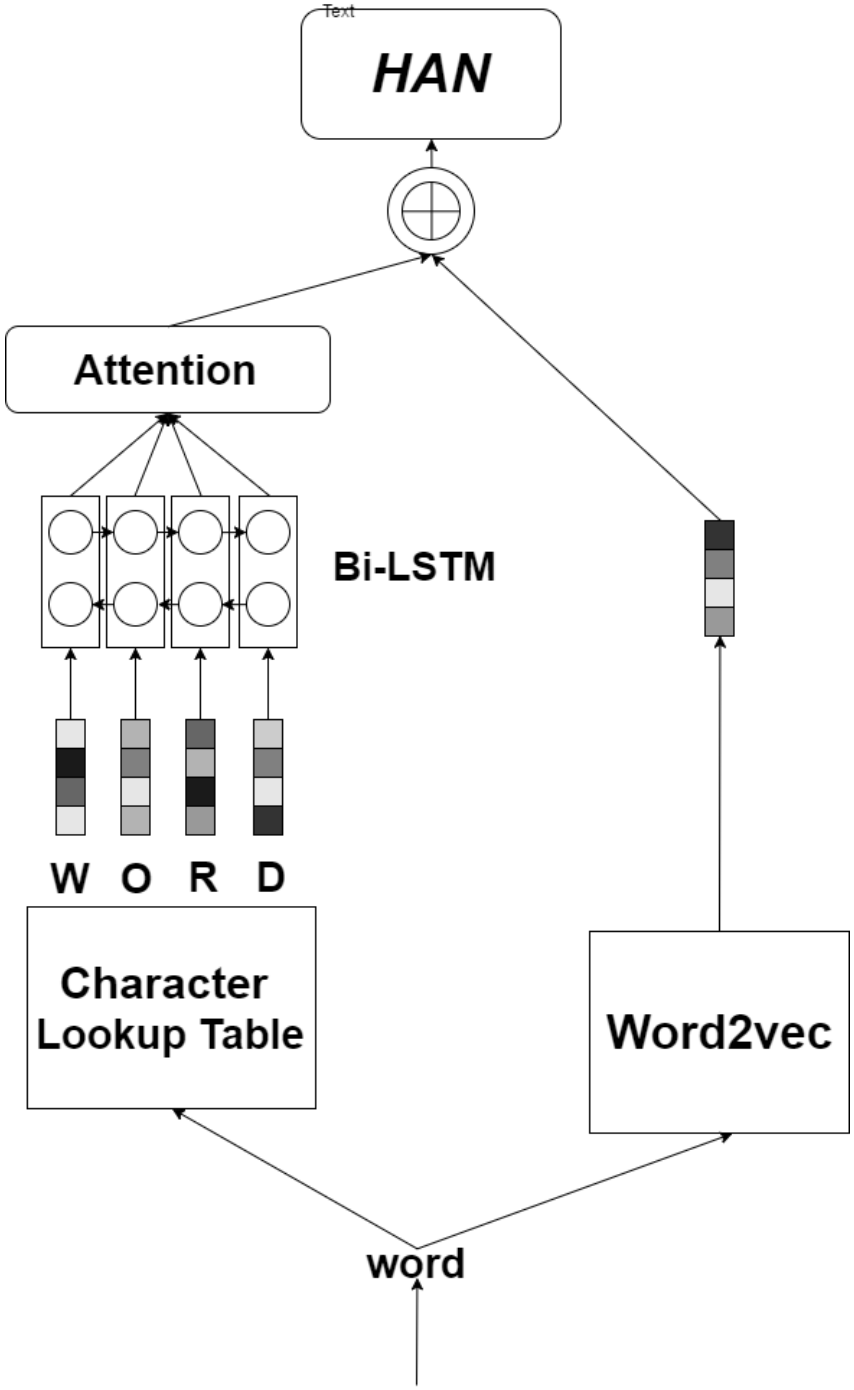


Рис. 9: финальная архитектура векторизации слов

- Добавление посимвольной векторизации действительно улучшает качество работы сети на данном наборе юридических документов. Это следует из сравнения WC2V-HAN и HAN.
- Методы, основанные на нейронных сетях превосходят классические по качеству предсказания решения суда. Это следует из сравнения LSTM/HAN/WC2V-HAN с остальными методами.

- Из сравнения LSTM и HAN следует, что добавление механизма внимания к рекуррентной нейронной сети действительно улучшает качество ее работы. Это подтверждает гипотезу об осмысленности весов генерируемых при помощи Attention unit .

Таблица 2: Сравнение качества работы различных алгоритмов

	NB	BoW	SVM	LSTM	HAN	WC2V-HAN
Same	74.46	84.12	82.50	92.61	95.69	**97.17**
Different	66.12	73.95	77.28	88.87	91.31	**93.92**

Для оценки качества бинарной классификации была построена ROC-кривая. Она показана на рисунке 10. Большое значение площади под кривой (0.98) свидетельствует о хорошем качестве классификатора.

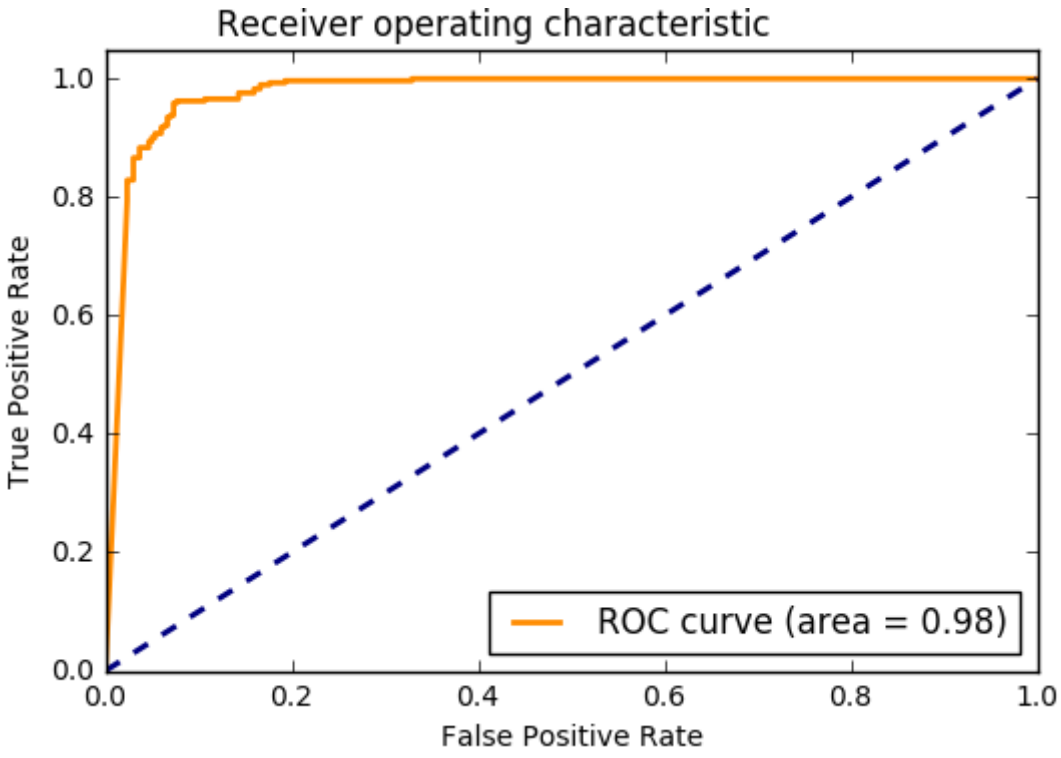


Рис. 10: финальная архитектура векторизации слов

3.5. О процессе обучения

Можно выделить ряд технических особенностей, использованных в процессе обучения:

- Все архитектуры нейронных сетей, описанные в данной главе были реализованы на языке Python, при помощи библиотеки Keras.

- В качестве функции потерь использовалась биномиальная кросс-энтропия. В качестве оптимизатора - rmsprop.
- В виду малого количества обучающих данных, с целью предотвращения переобучения, во все нейронные сети была добавлена Dropout регуляризация [17].
- При обучении сверточных моделей использовался механизм Batch Normalizatoin [8].

Стоит отметить, что WC2V-HAN существенно проигрывает HAN в скорости работы. Это вызвано наличием еще одного рекуррентного слоя.

4. Анализ результатов и визуализация

В главе 4 была описана нейронная сеть, определяющая вероятный вердикт суда на основании мотивировочной части решения. Теперь необходимо описать технику извлечения весов отдельных слов и предложений, а также произвести визуализацию полученных результатов.

4.1. Определение весов

Определим понятия весов предложений:

- **полный вес** предложения — это вклад предложения в финальное решение нейронной сети. По сути он отображает важность конкретного предложения в документе.
- **положительный вес** предложения — это вклад предложения в принятие удовлетворительного решения суда по конкретному делу.
- **отрицательный вес** предложения — это вклад предложения в принятие отказательного решения суда по конкретному делу.

Как уже говорилось в главе 2, средняя длина предложения в обучающей выборке составляет 48 слов, поэтому имеет смысл обратить внимание на распределение весов внутри предложения. Для этого дадим несколько определений: **полные, положительные, отрицательные веса слов** — это вклады слов в соответствующие веса предложений.

Также можно ввести понятие **абсолютного веса** слова — это произведение полных весов слова и его предложения. По сути он отображает вклад конкретного слова в финальное решение сети. Наконец, можно определить **ключевые слова** — слова с большим значением абсолютного веса.

Для шага визуализации необходимо определить положительные и отрицательные веса слов и предложений, а также выделить ключевые слова.

Полные веса слов и предложений — это прямая интерпретация весов внутри Attention unit первого и второго уровня HAN соответственно. Поэтому для их получения достаточно произвести вычисления α_{ij} и α_i соответственно (в теримнах с рис. 6).

Положительные/отрицательные веса — не имеют прямой интерпретации внутри описанной ранее архитектуры. Как правило, в статьях посвященных решению задачи Sentiment Analysis при помощи механизмов внимания основной задачей стоит предсказание классов документов, потому исследуются лишь полные веса. В данной работе мы предлагаем два подхода к решению данной задачи:

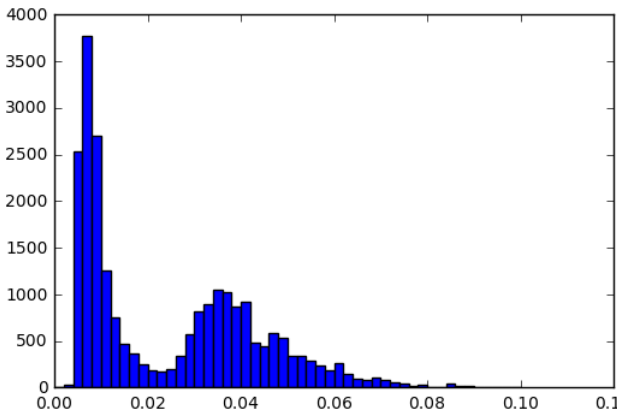
Output Gradient. Пусть на выходе сети есть два нейрона, выходы которых является вероятностями P_i соответствующих классов (удовлетворения/отклонения иска). Тогда для получения положительных/отрицательных весов предложений мы можем вычислить градиент $\frac{\partial P_i}{\partial h_j}$, где h_j является скрытым представлением j-го предложения. Поскольку градиенты могут быть отрицательными, для получения весов необходимо применить к ним softmax для преобразование в веса. Аналогичным образом можно получить веса слов, воспользовавшись градиентом $\frac{\partial P_i}{\partial h_{jk}}$, где h_jk - скрытое представление k-го слова j-го предложения.

Две сети. Альтернативный подход заключается в модификации последнего слоя нейронной сети. Можно произвести обучение двух экземпляров сети, у каждой из которых будет лишь один нейрон в качестве последнего слоя, активация которого будет соответствовать распознаванию класса документа. Т.е. у одной сети выходной нейрон будет активироваться в случае принятия удовлетворительного решения по делу, а у другой сети - в случае отказательного. благодаря этому трюку, вклады слов и предложений, полученные из механизмов внимания, теперь соответствуют положительным/отрицательным весам. Плюсом такого подхода является его эффективность при наличии обученной модели. В связи с этим он был выбран для дальнейшего применения в визуализации.

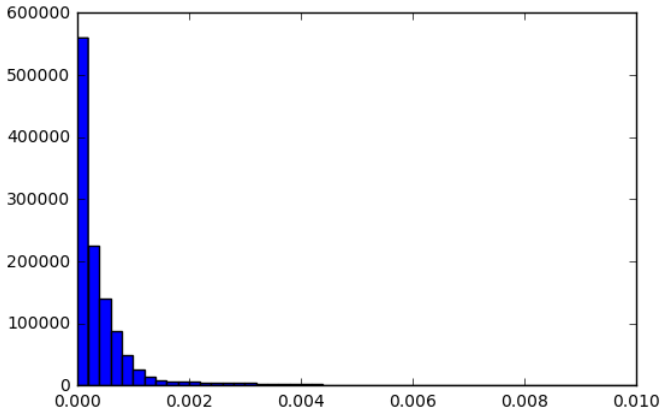
4.2. Визуализация

Для дальнейшего использования и анализа результатов необходимо визуализировать веса, полученные в п. 4.1. Были выделены следующие объекты визуализаиции:

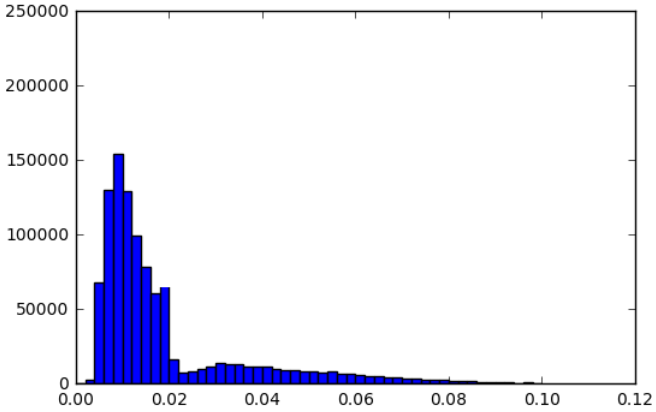
- Важные предложения – предложения, наиболее важные для принятие удовлетворительного/отказательного решения по делу. В связи с большим объемом документов, большая часть предложений практически игнорируются нейронной сетью, поэтому их разметка лишь ухудшит читаемость текста.
- Веса слов – внутри важных предложений будем отображать соотвествующие веса слов (положительные/отрицательные в зависимости от того, за счет какого из весов предложение стало важным).
- Ключевые слова – как правило, они встречаются внутри важных предлоежний,



(a) Распределение весов предложений



(b) Распределение абсолютных весов слов



(c) Распределение полных слов

Рис. 11: Анализ распределений весов

но иногда встречаются и вне них.

Сформулируем критерии определения важных предложений и ключевых слов.

На рисунке 11.a показано распределение весов предложений в тестовой выборке **different**, на котором наблюдается двухмодальное распределение, поэтому в качестве порогового значения важности было выбрано значение минимума между модами: 0.025. Предложения, положительные или отрицательные веса которых превосходят данное значения признаются важными.

С ключевыми словами ситуация сложнее. На рисунке 11.b изображено распределение абсолютных весов слов тестовой выборки. Данное распределение не позволяет определить разумных границ определения ключевых слов, поэтому обратимся к распределению полных весов слов с рисунка 11.c . По аналогии с рисунком 1.a здесь наблюдаются две моды распределения, что дает возможность определить пороговое значение весов слов равным 0.03. Тогда, используя определение абсолютного веса можно ввести пороговое значение равным $0.025 \cdot 0.03 = 0.00075$. Слова с абсолютным весом больше данного признаются ключевыми.

При помощи библиотеки PyDocX была реализована утилита, позволяющая произ-

Рис. 12: Примеры работы алгоритма визуализации

водить визуализацию и экспорт документов в формат совместимый с Microsoft Word. В качестве примера на рисунке 12 показаны отрывки отображаемого текста:

- Зеленым и красным выделены предложения у которых лишь один вес (положительный/отрицательный) превосходит пороговое значение. Под каждым словом указан его соответствующий весов.
- Желтым выделены предложения, у которые являются важным согласно значениям обоих весов. По сути – это предложения, которые важны для сути дела в целом, а не для принятия конкретного решения. Для таких предложений отображены как положительные, так и отрицательные веса словом.
- Жирным отображены ключевые слова, выбранные согласно описаным ранее критериям.

Заключение

В рамках данной работы можно выделить следующие основные результаты:

- Собран корпус текстов мотивировочных частей решений судов РФ. Эти данные могут быть использованы для решения других будущих задач: например, генерации выжимки материалов дела на основании его документов.
- Предложены улучшения модели HAN с учетом предметной области. Сравнение различных способов векторизации показало, что лучшее качество работы алгоритма достигается при комбинации семантического и синтаксического представлений слов.
- Предложены новые подходы к определению положительно/отрицательно настроенных частей текста. Это является скорее общим результатом в области Sentiment Analysis, который может быть использован в любых нейронных сетях с механизмом внимания.
- Произведена визуализация вкладов отдельных слов и предложений мотивировочной части в финальное решение суда. Данные материалы позволяют сделать выводы о том, что же влияет на конечный вердикт суда, поэтому размеченные документы переданы на дальнейшее исследование группе юристов.

Существует два способа улучшения качества предсказания и разметки документов:

- Использование более сложной архитектуры. Так, например, можно использовать более сложную(многоуровневую) модель узла внимания с использованием механизмов Deep Memory [15]. Это может привести к улучшению точности работы нейронной сети.
- Использование дополнительных данных. Например, можно использовать отзывы группы юристов о размеченных документах для реализации обратной связи.

A. Механизм Внимания

Listing 1: Attention Unit

```
class AttLayer(Layer):
    def __init__(self, **kwargs):
        super(AttLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3
        self.W = self.add_weight(shape=(input_shape[-1],),
            ↪ initializer='normal', trainable=True, name='W_att')
        super(AttLayer, self).build(input_shape)  # be sure you
            ↪ call this somewhere!

    def call(self, x, mask=None):
        eij = K.tanh(K.dot(x, self.W))
        ai = K.exp(eij)
        weights = ai / K.sum(ai, axis=1).dimshuffle(0, 'x')

        weighted_input = x * weights.dimshuffle(0, 1, 'x')
        return weighted_input.sum(axis=1)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[-1])
```

B. Сверточная векторизация

Listing 2: векторизация Char-CNN

```
sentence_input = Input(shape=(MAX_SENT_LENGTH,), dtype='int32')


filter_length = [5, 3]
nb_filter = [128, 128]
pool_length = 2

word_input = Input(shape=(MAX_WORD_LEN, N_CHARS))

embedded = Conv1D(filters=nb_filter[0],
                  kernel_size=filter_length[0],
                  padding='valid',
                  activation='relu',
                  kernel_initializer='glorot_normal',
                  strides=1)(word_input)

embedded = Dropout(0.2)(embedded)
embedded = MaxPooling1D(pool_size=pool_length)(embedded)

embedded = Conv1D(filters=nb_filter[1],
                  kernel_size=filter_length[1],
                  padding='valid',
                  activation='relu',
                  kernel_initializer='glorot_normal',
                  strides=1)(embedded)

embedded = Dropout(0.2)(embedded)
embedded = MaxPooling1D(pool_size=pool_length)(embedded)

embedded = Dense(EMBEDDING_DIM)(embedded)
embedded = Dropout(0.2)(embedded)
wordEncoder = Model(word_input, embedded)


raw_embedded = word_embedding_layer(sentence_input)
embedded = Reshape((MAX_SENT_LENGTH, MAX_WORD_LEN, N_CHARS),
```

```
                    input_shape=(MAX_SENT_LENGTH,
                        ↪ CHAR_EMBEDDING_DIM))(raw_embedded)
word_embedded = TimeDistributed(wordEncoder)(embedded)
word_embedded = TimeDistributed(Reshape((EMBEDDING_DIM,),
                                input_shape=(1, EMBEDDING_DIM)))(
                                    ↪ word_embedded)


embedded_sequences = w2v_embedding_layer(sentence_input)
total_embedded = Concatenate(axis=-1)([word_embedded,
    ↪ embedded_sequences])
```

C. Финальная архитектура

Listing 3: Модель WC2V-HAN

```
w2v = Word2Vec(sents, size=EMBEDDING_DIM, window = 5, min_count =
    ↪  5)

data = np.zeros((len(X), MAX_SENTS, MAX_SENT_LENGTH), dtype='
    ↪ int32')
word2idx = {}
idx2word = {}
distinct_words = 0
for i, sentences in enumerate(X):
    for j, sent in enumerate(sentences):
        if j< MAX_SENTS:
            k = 0
            for _, word in enumerate(sent):
                if k<MAX_SENT_LENGTH and word in w2v.vocab:
                    if word not in word2idx:
                        distinct_words += 1
                        word2idx[word] = distinct_words
                        idx2word[distinct_words] = word
                    data[i,j,k] = word2idx[word]
                    k=k+1

chars = {c for w in set(all_words) for c in w}
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))

MAX_WORD_LEN = 15
N_CHARS = len(chars)


CHAR_EMBEDDING_DIM = MAX_WORD_LEN * N_CHARS

word2char = np.random.random((len(word2idx) + 1,
    ↪ CHAR_EMBEDDING_DIM))

for word, i in word2idx.items():
```

```
    embedding_vector = np.zeros(CHAR_EMBEDDING_DIM)
    n = min(MAX_WORD_LEN, len(word))
    for j, c in enumerate(word[:n]):
        embedding_vector[(n - 1 - j) * N_CHARS + char_indices[c]]
            ↪ = 1
    word2char[i] = embedding_vector

word_embedding_layer = Embedding(len(word2idx) + 1,
                                 CHAR_EMBEDDING_DIM,
                                 weights=[word2char],
                                 input_length=MAX_SENT_LENGTH,
                                 trainable=True)

w2v_embedding_matrix = np.random.random((len(word2idx) + 1,
    ↪ EMBEDDING_DIM))
for word, i in word2idx.items():
    embedding_vector = w2v[word]
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        w2v_embedding_matrix[i] = embedding_vector

w2v_embedding_layer = Embedding(len(word2idx) + 1,
                                EMBEDDING_DIM,
                                weights=[w2v_embedding_matrix],
                                input_length=MAX_SENT_LENGTH,
                                trainable=True)

LING_DIM = 50
word_input = Input(shape=(MAX_WORD_LEN, N_CHARS))
embedded = TimeDistributed(Dense(LING_DIM))(word_input)
l_lstm_emb = Bidirectional(GRU(100, return_sequences=True,
    ↪ dropout=0.2))(embedded)
l_dense_emb = TimeDistributed(Dense(EMBEDDING_DIM))(l_lstm_emb)
l_att_emb = AttLayer()(l_dense_emb)
wordEncoder = Model(word_input, l_att_emb)

sentence_input = Input(shape=(MAX_SENT_LENGTH,), dtype='int32')

raw_embedded = word_embedding_layer(sentence_input)
```

```
embedded = Reshape((MAX_SENT_LENGTH, MAX_WORD_LEN, N_CHARS),
                   input_shape=(MAX_SENT_LENGTH,
                       ↪ CHAR_EMBEDDING_DIM))
                   (raw_embedded)
word_embedded = TimeDistributed(wordEncoder)(embedded)
word_embedded = TimeDistributed(Reshape((EMBEDDING_DIM,),
                                input_shape=(1, EMBEDDING_DIM)))
                                (word_embedded)

embedded_sequences = w2v_embedding_layer(sentence_input)
total_embedded = Concatenate(axis=-1)([word_embedded,
    ↪ embedded_sequences])

l_lstm = Bidirectional(GRU(100, return_sequences=True, dropout
    ↪ =0.2))(total_embedded)
l_dense = TimeDistributed(Dense(200))(l_lstm)
l_att = AttLayer()(l_dense)
sentEncoder = Model(sentence_input, l_att)

case_input = Input(shape=(MAX_SENTS,MAX_SENT_LENGTH), dtype='
    ↪ int32')
case_encoder = TimeDistributed(sentEncoder)(review_input)
l_lstm_sent = Bidirectional(GRU(100, return_sequences=True,
    ↪ dropout=0.2))(review_encoder)
l_dense_sent = TimeDistributed(Dense(200))(l_lstm_sent)
l_att_sent = AttLayer()(l_dense_sent)
preds = Dense(2, activation='softmax')(l_att_sent)
model = Model(review_input, preds)

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])
```

Список литературы

[1] Aletras N Tsarapatsanis D Preoţiuc-Pietro D Lampos V. Predicting judicial decisions of the European Court of Human Rights: A Natural Language Processing perspective. — 2016. — URL: http://www.lampos.net/publications/inferring-judicial-decisions-with-stat-nlp.

[2] Character-aware neural language models / Yoon Kim, Yacine Jernite, David Sontag, Alexander M Rush // Thirtieth AAAI Conference on Artificial Intelligence. — 2016.

[3] Competing approaches to predicting supreme court decision making / Andrew D Martin, Kevin M Quinn, Theodore W Ruger, Pauline T Kim // Perspectives on Politics. — 2004. — Vol. 2, no. 04. — P. 761–767.

[4] Dzmitry Bahdanau Kyunghyun Cho Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. — 2015. — URL: https://arxiv.org/abs/1409.0473.

[5] Finding function in form: Compositional character models for open vocabulary word representation / Wang Ling, Tiago Luís, Luís Marujo et al. // arXiv preprint arXiv:1508.02096. — 2015.

[6] Hierarchical attention networks for document classification / Zichao Yang, Diyi Yang, Chris Dyer et al. // Proceedings of NAACL-HLT. — 2016. — P. 1480–1489.

[7] Hochreiter Sepp, Schmidhuber Jürgen. Long short-term memory // Neural computation. — 1997. — Vol. 9, no. 8. — P. 1735–1780.

[8] Ioffe Sergey, Szegedy Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift // arXiv preprint arXiv:1502.03167. — 2015.

[9] Kushal Dave Steve Lawrence David M. Pennock. Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. — 2003. — URL: http://dl.acm.org/citation.cfm?id=775226.

[10] Liu P. Joty S. R. Meng H. M. Fine-grained Opinion Mining with Recurrent Neural Networks and Word Embeddings. — 2015.

[11] Saif M. Mohammad Peter D. Turney. Emotions evoked by common words and phrases: using mechanical turk to create an emotion lexicon. — 2010. — URL: http://dl.acm.org/citation.cfm?id=1860635.

[12] Saif M Mohammad Svetlana Kiritchenko, Zhu. Xiaodan. NRCCanada: Building the State-of-the-Art in Sentiment Analysis of Tweets. — 2013. — URL: http://dl.acm.org/citation.cfm?id=1860635.

[13] Sentiment Analysis using Tf-Idf weighting: Python/Scikit-learn. — 2017. — URL: https://appliedmachinelearning.wordpress.com/2017/02/12/sentiment-analysis-using-tf-idf-weighting-pythonscikit-learn/ (дата обращения: 04.05.2017).

[14] Structured models for fine-to-coarse sentiment analysis / Ryan McDonald, Kerry Hannan, Tyler Neylon et al. // Annual Meeting-Association For Computational Linguistics / Citeseer. — Vol. 45. — 2007. — P. 432.

[15] Tang Duyu, Qin Bing, Liu Ting. Aspect level sentiment classification with deep memory network // arXiv preprint arXiv:1605.08900. — 2016.

[16] Tomas Mikolov Quoc V Le, Sutskever Ilya. Exploiting Similarities among Languages for Machine Translation. — 2013.

[17] Wager Stefan, Wang Sida, Liang Percy S. Dropout training as adaptive regularization // Advances in neural information processing systems. — 2013. — P. 351–359.

[18] Wiki. Судебное решение // Википедия, свободная энциклопедия. — 2012. — URL: goo.gl/B7Hjgs (дата обращения: 08.04.2013).

[19] Zhang Xiang, Zhao Junbo, LeCun Yann. Character-level convolutional networks for text classification // Advances in neural information processing systems. — 2015. — P. 649–657.