Федеральное государственное бюджетное образовательное
учреждение высшего образования
Московский авиационный институт
(национальный исследовательский университет)

Факультет №3
«Системы управления, информатика и электроэнергетика»

Кафедра 304
«Вычислительные машины, системы и сети»

Выпускная квалификационная работа
на тему «Классификация текстов на основе нейронных сетей»

Преподаватель: Чебатко М.И.

_____

Студент: Долгополов Н.И.

Группа: М3О-207М-17

_____

Москва, 2018

1 Оглавление

Введение

Данная работа посвящена анализу существующих решений для каждого этапа классификации текстов с целью построения классификатора, обладающего наилучшими характеристиками. В рамках работы будет написана программа, использующая сверточные нейронные сети и демонстрирующая преимущества данного подхода.

Объектом данного исследования является задача классификации текстов, то есть присвоения тексту одной из заданных категорий, основываясь исключительно на содержимом текста. Предметом работы является применение методов машинного обучения для решения этой задачи, а также анализ и сравнение этих методов.

Актуальность данной работы заключается в том, что, несмотря на существующее многообразие подходов для классификации текстов, до сих пор не найден оптимальный, который стал бы стандартом в этой сфере. Помимо этого, большинство методов были разработаны во времена, когда вычислительные мощности были несопоставимы с современными, и поэтому такие методы имеют существенные ограничения с целью оптимизации вычислений. Современная вычислительная техника дает возможность использовать более эффективные, но при этом более затратные методы. Новизна работы заключается в применении сверточных нейронных сетей к работе с текстами. Обычно этот метод используется для анализа изображений, но его применение к задаче классификации текстов дало результаты, превосходящие традиционные подходы к решению этой задачи.

Целью работы является создание приложения для анализа тональности текста, то есть такого, которое отличает тексты с «положительной интонацией» от текстов с «отрицательной». Предварительно приложение должно быть обучено на большом объеме текстов с заранее известной интонацией.

Для достижения такой цели должны быть выполнены следующие задачи:

1. Изучение подходов для каждого из этапов классификации текста
2. Сравнение и выбор оптимального подхода
3. Создание программы для классификации текстов на основе выбранных подходов

Научным результатом работы должно стать подтверждение возможности применения сверточных нейросетей к задаче классификации текстов.

1 Теория классификации текстов

1.1 Основные понятия и определения

1.1.1 Постановка задачи классификации текстов

Классификация документов — одна из задач информационного поиска, заключающаяся в отнесении документа к одной из нескольких категорий на основании содержания документа.

Если сформулировать задачу более формально, то существует множество документов $D = \{d_1, ..., d_n\}$, множество категорий (классов, меток) $C = \{c_1, ..., c_n|\}$ и неизвестная целевая функция $\Phi: C \times D \rightarrow \{0, 1\}$. Необходимо построить классификатор Ф', максимально близкий к Ф.

Имеется некоторая начальная коллекция размеченных документов $R \subset C \times D$, для которых известны значения Ф. Обычно её делят на «обучающую» и «проверочную» части. Первая используется для обучения классификатора, вторая — для независимой проверки качества его работы.

Классификатор может выдавать точный ответ $\Phi': C \times D \rightarrow \{0,1\}$ или степень подобия $\Phi': C \times D \rightarrow [0,1]$.

1.1.2 Определение анализа тональности

Анализ тональности (Sentiment analysis) — это область компьютерной лингвистики, которая занимается изучением мнений и эмоций в текстовых документах.

Целью анализа тональности является нахождение мнений в тексте и определение их свойств. В зависимости от поставленной задачи нас могут интересовать разные свойства, например:

автор — кому принадлежит это мнение

тема — о чем говорится во мнении

тональность — позиция автора относительно упомянутой темы (обычно «положительная» или «отрицательная»)

Пример: "Главный итог завершившихся Игр XXX Олимпиады в Лондоне – то чувство гордости за нашу страну, которое

испытывали болельщики благодаря выступлениям российских олимпийцев», — считает Александр Жуков"

автор: Александр Жуков

тема: "выступление российских олимпийцев"

тональность: "положительная"

В литературе встречаются разные способы формализации модели мнений. Также используется и разная терминология. В английском языке эту область исследования обычно называют opinion mining and sentiment analysis (дословно: «поиск мнений и анализ чувств»). В русских статьях обычно употребляется термин «анализ тональности». Несмотря на то, что тональность является лишь одной из характеристик мнения, именно задача классификации тональности является наиболее часто изучаемой в наши дни. Это можно объяснить несколькими причинами:

Во многих случаях нам достаточно лишь определить тональность, т.к. другие характеристики нам уже известны. Например, если мы собираем мнения из блогов, обычно авторами мнений являются авторы постов, т.е. определять автора нам не требуется. Также зачастую нам уже известна тема: например, если мы производим в Твиттере поиск по ключевому слову «Windows 8», то затем нам нужно лишь определить тональность найденных твитов. Конечно же, это работает не во всех случаях, а лишь в большинстве из них. Но эти допущения позволяют в значительной мере упростить и так нелегкую задачу.

Анализ тональности находит свое практическое применение в разных областях:

- **Социология** — сбор данных из социальных сетей (например, о религиозных взглядах)
- **Политология** — сбор данных из блогов о политических взглядах населения

- **Маркетинг** — анализ социальных сетей, чтобы узнать, какая модель ноутбуков пользуется наибольшим спросом

  •**Медицина и психология** — определение депрессии у пользователей социальных сетей

1.2 Методы решения

1.2.1 Подход на правилах (Rule based classification)

Первый подход решения задачи классификации состоит из набора правил, применяя которые система делает заключение о тональности текста. Например, для предложения «Я люблю кофе», можно применить следующее правило:

если сказуемое ("*люблю*") входит в положительный набор глаголов ("*люблю*", "*обожаю*", "*одобряю*" ...) и в предложении не имеется отрицаний, то классифицировать тональность как "**положительная**"

Многие коммерческие системы используют данный подход, несмотря на то что он требует больших затрат, т.к. для хорошей работы системы необходимо составить большое количество правил. Зачастую правила привязаны к определенному домену (например, «ресторанная тематика») и при смене домена («обзор фотоаппаратов») требуется заново составлять правила. Тем не менее, этот подход является наиболее точным при наличии хорошей базы правил, но совершенно неинтересным для исследования.

1.2.2 Подход на словарях (Weight based classification)

Подходы, основанные на словарях, используют так называемые тональные словари (affective lexicons) для анализа текста. В простом виде тональный словарь представляет из себя список слов со значением тональности для каждого слова. Вот пример из базы ANEW, переведенный на русский:

слово	валентность (1-9)
счастливый	8.21
хороший	7.47
скучный	2.95
сердитый	2.85
грустный	1.61

Чтобы проанализировать текст, можно воспользоваться следующим алгоритмом: сначала каждому слову в тексте присвоить его значение тональности из словаря (если оно присутствует в словаре), а затем вычислить общую тональность всего текста. Вычислять общую тональность можно разными способами. Самый простой из них — среднее арифметическое всех значений. Более сложный — обучить классификатор (напр. нейронная сеть).

Такой метод требует меньше усилий для обучения, чем классификация с помощью правил, но, тем не менее, является недостаточно гибким и универсальным.

1.2.3 Машинное обучение (Machine Learning)

Машинное обучение с учителем является наиболее распространенным методом, используемым в исследованиях. Его суть состоит в том, чтобы обучить машинный классификатор на коллекции заранее размеченных текстов, а затем использовать полученную модель для анализа новых документов.

В этом подходе набор правил или, более обще, критерий принятия решения текстового классификатора, вычисляется автоматически из обучающих данных (другими словами, производится обучение

классификатора). Обучающие данные — это некоторое количество хороших образцов документов из каждого класса. В машинном обучении сохраняется необходимость ручной разметки (термин разметка означает процесс приписывания класса документу). Но разметка является более простой задачей, чем написание правил. Кроме того, разметка может быть произведена в обычном режиме использования системы. Например, в программе электронной почты может существовать возможность помечать письма как спам, тем самым формируя обучающее множество для классификатора — фильтра нежелательных сообщений. Таким образом, классификация текстов, основанная на машинном обучении, является примером обучения с учителем, где в роли учителя выступает человек, задающий набор классов и размечающий обучающее множество.

Данный метод будет использован в работе, как представляющий наибольший интерес для исследования и позволяющий добиться наилучших результатов. Будет решена задача классификации текстов, при этом, возможно, будет применен анализ тональности.

1.3 Этапы классификации текстов

Полное решение задачи классификации текстов принято разбивать на следующие этапы:

1. Формирование базы (корпуса) текстов
2. Предобработка
3. Индексация
4. Выбор признаков
5. Построение и обучение классификатора
6. Оценка качества

К теоретическим этапам относятся индексация, выбор признаков и построение и обучение классификатора. Именно они будут проанализированы в данной главе. Практические этапы будут рассмотрены в главе №3, посвященной программной реализации задачи.

1.4 Индексация

Вычислительная сложность различных методов классификации напрямую зависит от размерности пространства признаков. Поэтому для эффективной работы классификатора часто прибегают к сокращению числа используемых признаков (терминов).

За счет уменьшения размерности пространства терминов можно снизить эффект переобучения – явление, при котором классификатор ориентируется на случайные или ошибочные характеристики обучающих данных, а не на важные и значимые. Переобученный классификатор хорошо работает на тех экземплярах, на которых он обучался, и значительно хуже на тестовых данных. Чтобы избежать переобучения, количество обучающих примеров должно быть соразмерно числу используемых терминов. В некоторых случаях сокращение размерности пространства признаков в 10 раз (и даже в 100) может приводить лишь к незначительному ухудшению работы классификатора.

1.4.1 N-граммы

Качество результатов напрямую зависят от того, как мы представим документ для классификатора, а именно, какой набор характеристик мы будем использовать для составления вектора признаков. Наиболее распространенный способ представления документа в задачах компьютерной лингвистики и поиска — это либо в виде набора слов (bag-of-words) либо в виде набора N-грамм. Так, например, предложение «Я люблю черный кофе» можно представить в виде набора униграмм (Я, люблю, черный, кофе) или биграмм (Я люблю, люблю черный, черный кофе).

Обычно униграммы и биграммы дают лучшие результаты чем N-граммы более высоких порядков (триграммы и выше), т.к. выборка обучения в большинстве случаев недостаточна большая для подсчета N-грамм высших порядков. Всегда имеет смысл протестировать результаты с применением униграмм, биграмм и их комбинации (Я, люблю, черный, кофе, Я люблю, люблю черный, черный кофе). В зависимости от типа данных униграммы

могут показать лучшие результаты чем биграммы, а могут и наоборот. Также иногда комбинация униграммов и биграммов позволяет улучшить результаты.

1.4.2 Bag of words

Чтобы предложение можно было подать на вход нейронной сети, надо решить несколько проблем. Во-первых, необходимо преобразовать слова в цифры. Первое предположение — сопоставить каждому слову из словаря свое число. Скажем (Абрикос — 1, Аппарат — 2, …. Яблоко — 53845). Но делать так нельзя, потому что таким образом мы неявно предполагаем, что абрикос гораздо больше похож на аппарат, чем на яблоко. Второй вариант — закодировать слова длинным вектором, в котором нужному слову соответствует 1, а всем остальным — 0 (Абрикос — 1 0 0 …, Аппарат — 0 1 0 0 …, … Яблоко — … 0 0 0 1). Здесь все слова равноудалены и не похожи друг на друга. Этот подход гораздо лучше и в ряде случаев работает хорошо (если есть достаточно много примеров).

Но если набор примеров маленький, то весьма вероятно, что какие-то слова (например, «абрикос») в нем будут отсутствовать, и в результате встретив такие слова в реальных примерах, алгоритм не будет знать, что с ними делать. Поэтому оптимально кодировать слова такими векторами, чтобы похожие по смыслу слова оказывались близко друг к другу — а далекие, соответственно — далеко. Есть несколько алгоритмов, которые «читают» большие объемы текстов, и на основании этого создают такие вектора (самый известный, но не всегда самый лучший — word2vec).

1.4.3 Word2vec

Word2vec — программный инструмент анализа семантики естественных языков, представляющий собой технологию, которая основана на дистрибутивной семантике и векторном представлении слов. Этот инструмент был разработан группой исследователей Google в 2013 году. Работа этой технологии осуществляется следующим образом: word2vec принимает

большой текстовый корпус в качестве входных данных и сопоставляет каждому слову вектор, выдавая координаты слов на выходе. Сначала он создает словарь, «обучаясь» на входных текстовых данных, а затем вычисляет векторное представление слов. Векторное представление основывается на контекстной близости: слова, встречающиеся в тексте рядом с одинаковыми словами (следовательно, имеющие схожий смысл), в векторном представлении будут иметь близкие координаты векторов-слов. Полученные векторы-слова могут быть использованы для обработки естественного языка и машинного обучения.

В word2vec существуют два основных алгоритма обучения : CBOW (Continuous Bag of Words) и Skip-gram. CBOW — «непрерывный мешок со словами» модельная архитектура, которая предсказывает текущее слово, исходя из окружающего его контекста. Архитектура типа Skip-gram действует иначе: она использует текущее слово, чтобы предугадывать окружающие его слова. Пользователь word2vec имеет возможность переключаться и выбирать между алгоритмами. Порядок слов контекста не оказывает влияния на результат ни в одном из этих алгоритмов.

Получаемые на выходе координатные представления векторов-слов позволяют вычислять «семантическое расстояние» между словами. И, именно основываясь на контекстной близости этих слов, технология word2vec совершает свои предсказания. Так как инструмент word2vec основан на обучении нейронной сети, чтобы добиться его наиболее эффективной работы, необходимо использовать большие корпусы для его обучения. Это позволяет повысить качество предсказаний.

1.5 Выбор признаков

1.5.1 TF-IDF

Существует несколько способов определения веса признаков документа. Наиболее распространенный – вычисление функции TF-IDF. Его основная идея состоит в том, чтобы больший вес получали слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

Вычисляется частота термина TF (term frequency) – оценка важности слова в пределах одного документа d по формуле $TF = n_{t,d} / n_d$, где $n_{t,d}$ – количество употреблений слова t в документе d; n_d – общее число слов в документе d. Обратная частота документа IDF (inverse document frequency) – инверсия частоты, с которой слово встречается в документах коллекции. IDF уменьшает вес общеупотребительных слов по формуле $IDF = \log(|D| / D_t)$, где $|D|$ – общее количество документов в коллекции; D_t – количество всех документов, в которых встречается слово t. Итоговый вес термина в документе относительно всей коллекции документов вычисляется по формуле $V_{t,d} = TF \times IDF$. Следует отметить, что по данной формуле оценивается значимость термина только с точки зрения частоты вхождения в документ, без учета порядка следования терминов в документе и их лексической сочетаемости.

1.5.2 Дельта TF-IDF

Идея метода дельта TF-IDF заключается в том, чтобы дать больший вес для слов, которые имеют не-нейтральную тональность, т.к. именно такие слова определяют тональность всего текста. Формула для расчета веса слова w следующая:

$$V_{t,d} = C_{t,d} \cdot \log\left(\frac{|N| \cdot P_t}{|P| \cdot N_t}\right)$$

где:

$V_{t,d}$ — вес слова t в документе d

$C_{t,d}$ — количество раз слово t встречается в документе d

|P| — количество документов с положительной тональностью

|N| — количество документов с отрицательной тональностью

Pt — количество положительных документов, где встречается слово t

N_t — количество отрицательных документов, где встречается слово t

Допустим, мы работаем с коллекцией отзывов фильмов. Рассмотрим три слова: «отличный», «нудный», «сценарий». Самое главное в формуле дельта TF-IDF — это второй множитель log(...). Именно он будет разный у этих трех слов:

Слово «отличный» скорее всего встречается в большинстве положительных (P_t) отзывов и почти не встречается в отрицательных (N_t), в итоге вес будет большим положительным числом, т.к. отношение P_t/N_t будет числом гораздо больше 1.

Слово «нудный» наоборот встречается в основном в отрицательных отзывах, поэтому отношение P_t/N_t будет меньше единицы и в итоге логарифм будет отрицательным. В итоге вес слова будет отрицательным числом, но большим по модулю.

Слово «сценарий» может встречаться с одинаковой вероятностью и в положительных, так и в отрицательных отзывах, поэтому отношение P_t/N_t будет очень близко к единице, и в итоге логарифм будет близок к нулю. Вес слова будет практически равен нулю.

В итоге вес слов с положительной тональностью будет большим положительным числом, вес слов с отрицательной тональностью будет отрицательным числом, вес нейтральных слов будет близок к нулю. Такое взвешивание вектора признаков в большинстве случаев позволяет улучшить точность классификации тональности.

1.6 Классификаторы

Можно выделить следующие методы классификации:

- Вероятностные
- Метрические
- Логические
- Линейные
- Методы на основе искусственных нейронных сетей

Далее обобщенно описываются эти методы, указываются преимущества и недостатки каждого из них.

Метод Байеса (Naive Bayes, NB)

Данный метод относится к вероятностным методам классификации.

Преимущества метода:

- высокая скорость работы;
- поддержка инкрементного обучения;
- относительно простая программная реализация алгоритма;
- легкая интерпретируемость результатов работы алгоритма.

Недостатки метода: относительно низкое качество классификации и неспособность учитывать зависимость результата классификации от сочетания признаков.

Метод k ближайших соседей (k Nearest Neighbors, KNN)

Данный метод относится к метрическим методам классификации. Чтобы найти категорию, соответствующую документу d, классификатор сравнивает d со всеми документами из обучающей выборки L, то есть для каждого $d_z \in L$ вычисляется расстояние $r(d_z, d)$. Далее из обучающей выборки выбираются k документов, ближайших к d. Согласно методу k ближайших соседей, документ d считается принадлежащим тому классу, который является наиболее распространенным среди соседей данного документа, то есть для каждого класса ci вычисляется функция ранжирования:

$$CSV(d) = \sum_{d_z \in L_k(d)} \rho(d_z, d) \cdot \Phi(d_z, c_i)$$

где L_k (d) – ближайшие k документов из L к d; $F(d_z , c_i)$ – известные величины, уже расклассифицированные по категориям документы обучающей выборки.

Преимущества метода:

- возможность обновления обучающей выборки без переобучения классификатора;
- устойчивость алгоритма к аномальным выбросам в исходных данных;
- относительно простая программная реализация алгоритма;
- легкая интерпретируемость результатов работы алгоритма;
- хорошее обучение в случае с линейно неразделимыми выборками.

Недостатки метода:

- репрезентативность набора данных, используемого для алгоритма;
- высокая зависимость результатов классификации от выбранной метрики;

- большая длительность работы из-за необходимости полного перебора обучающей выборки;

- невозможность решения задач большой размерности по количеству классов и документов.

Метод деревьев решений (Decision Trees, DT)

Данный метод относится к логическим методам классификации.

Деревом решений называют ациклический граф, по которому производится классификация объектов (в нашем случае текстовых документов), описанных набором признаков. Каждый узел дерева содержит условие ветвления по одному из признаков. У каждого узла столько ветвлений, сколько значений имеет выбранный признак. В процессе классификации осуществляются последовательные переходы от одного узла к другому в соответствии со значениями признаков объекта. Классификация считается завершенной, когда достигнут один из листьев (конечных узлов) дерева. Значение этого листа определит класс, которому принадлежит рассматриваемый объект. На практике обычно используют бинарные деревья решений, в которых принятие решения перехода по ребрам осуществляется простой проверкой наличия признака в документе. Если значение признака меньше определенного значения, выбирается одна ветвь, если больше или равно, другая.

В отличие от остальных подходов, представленных ранее, подход, использующий деревья решений, относится к символьным (то есть нечисловым) алгоритмам.

Алгоритм построения бинарного дерева решений состоит из следующих шагов.

Создается первый узел дерева, в который входят все документы, представленные всеми имеющимися признаками. Размер вектора признаков для каждого документа равен n, так как $d = (t_1, \ldots, t_n)$.

Для текущего узла дерева выбираются наиболее подходящий признак t_k и его наилучшее пограничное значение v_k.

На основе пограничного значения выбранного признака производится разделение обучающей выборки на две части. Далее выбранный признак не включается в описание фрагментов в этих частях, то есть фрагменты в частях представляются вектором с размерностью $n - 1$.

Образовавшиеся подмножества обрабатываются аналогично до тех пор, пока в каждом из них не останутся документы только одного класса или признаки для различения документов.

Когда говорят о выборе наиболее подходящего признака, как правило, подразумевают частотный признак, то есть любой признак текста, допускающий возможность нахождения частоты его появления в тексте. Лучшим для разделения является признак, дающий максимальную на данном шаге информацию о категориях. Таким признаком для текста может являться, например, ключевое слово. С этой точки зрения любой частотный признак можно считать переменной. Тогда выбор между двумя наиболее подходящими признаками сводится к оценке степени связанности двух переменных. Поэтому для выбора подходящего признака на практике применяют различные критерии проверки гипотез, то есть критерии количественной оценки степени связанности двух переменных, поставленных во взаимное соответствие, где 0 соответствует полной независимости переменных, а 1 – их максимальной зависимости.

Для исследования связи между двумя переменными удобно использовать представление совместного распределения этих переменных в виде таблицы сопряженности (факторной таблицы, или матрицы частот появления

признаков). Она является наиболее универсальным средством изучения статистических связей, так как в ней могут быть представлены переменные с любым уровнем измерения. Таблицы сопряженности часто используются для проверки гипотезы о наличии связи между двумя признаками при помощи различных статистических критериев: критерия Фишера (точного теста Фишера), критерия согласия Пирсона (критерия хи-квадрат), критерия Крамера, критерия Стьюдента (t-критерия Стьюдента) и пр.

Преимущества метода:

- относительно простая программная реализация алгоритма;
- легкая интерпретируемость результатов работы алгоритма.

Недостатки метода: неустойчивость алгоритма по отношению к выбросам в исходных данных и большой объем данных для получения точных результатов.

Метод опорных векторов (Support Vector Machine, SVM)

Данный метод является линейным методом классификации. В настоящее время этот метод считается одним из лучших. Рассмотрим множество документов, которые необходимо расклассифицировать. Сопоставим ему множество точек в пространстве размерности |D|.

Выборку точек называют линейно разделимой, если принадлежащие разным классам точки можно разделить с помощью гиперплоскости (в двухмерном случае гиперплоскостью является прямая линия). Очевидный способ решения задачи в таком случае – провести прямую так, чтобы по одну сторону от нее лежали все точки одного класса, а по другую – все точки другого класса. Тогда для классификации неизвестных точек достаточно

будет посмотреть, с какой стороны прямой они окажутся.

В общем случае можно провести бесконечное множество гиперплоскостей (прямых), удовлетворяющих нашему условию. Ясно, что лучше всего выбрать

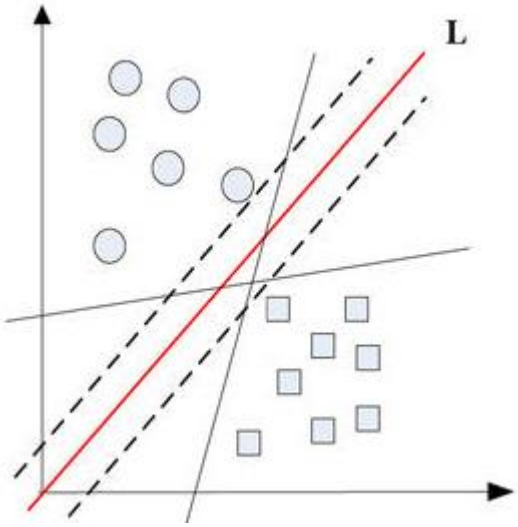

Рисунок 1.1. Разделяющая гиперплоскость в методе опорных векторов.

прямую, максимально удаленную от имеющихся точек. В методе опорных векторов расстоянием между прямой и множеством точек считается расстояние между прямой и ближайшей к ней точкой из множества. Именно такое расстояние и максимизируется в данном методе. Гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей, называется разделяющей. Ближайшие к параллельным гиперплоскостям точки называются опорными векторами, через них проходят пунктирные линии. Другими словами, алгоритм работает в предположении, что, чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора, так как максимизация зазора между классами способствует более уверенной классификации.

На практике структура данных зачастую бывает неизвестна и очень редко удается построить разделяющую гиперплоскость, а значит, невозможно гарантировать линейную разделимость выборки. Могут существовать такие документы, которые алгоритм отнесет к одному классу, а в действительности они должны относиться к противоположному. Такие данные называются выбросами, они создают погрешность метода, поэтому было бы лучше их игнорировать. В этом заключается суть проблемы линейной неразделимости.

Выборку называют линейно неразделимой, если точки, принадлежащие разным классам, нельзя разделить с помощью гиперплоскости. Когда такой разделяющей гиперплоскости не существует, необходимо перейти от исходного пространства признаков документов к новому, в котором обучающая выборка окажется линейно разделимой. Для этого каждое скалярное произведение необходимо заменить на некоторую функцию, отвечающую определенным требованиям. Например, можно назначать некий штраф за каждый неверно расклассифицированный документ. Эту функцию называют ядром. Замена скалярного произведения функцией-ядром позволяет перейти к другому пространству признаков, где данные уже будут разделимы.

В случае линейной неразделимости проблема поиска оптимальной разделяющей гиперплоскости сводится к задаче, эквивалентной поиску седловой точки функции Лагранжа с условиями дополняющей нежесткости. Полученная система уравнений решается методами квадратичного программирования. Это уже чисто вычислительная задача.

Этот вариант алгоритма называют алгоритмом с мягким зазором (soft-margin SVM), тогда как в линейно разделимом случае говорят о жестком зазоре (hard-margin SVM).

Преимущества метода:

- один из наиболее качественных методов;
- возможность работы с небольшим набором данных для обучения;
- сводимость к задаче выпуклой оптимизации, имеющей единственное решение.

Недостатки метода: сложная интерпретируемость параметров алгоритма и неустойчивость по отношению к выбросам в исходных данных.

1.6.1 Логистическая регрессия (logit model, logistic regression)

Данный метод также является линейным методом классификации. Этот метод используется для предсказания вероятности возникновения некоторого события по значениям множества признаков. Для этого вводятся так называемая зависимая переменная y, которая может принимать лишь одно из двух значений – как правило, это числа 0 (событие не произошло) и 1 (событие произошло), и множество независимых переменных (также называемых признаками, предикторами или регрессорами) – вещественных x_1, …, x_n, на основе значений которых требуется вычислить вероятность принятия того или иного значения зависимой переменной. В случае классификации документов роль зависимой переменной выполняет категория c_i, а роль независимых переменных – набор документов d_1, …, d_n.

Для улучшения обобщающей способности алгоритма, то есть для уменьшения эффекта переобучения, на практике часто рассматривается логистическая регрессия с регуляризацией. Регуляризация заключается в том, что вектор параметров q рассматривается как случайный вектор с некоторой заданной априорной плотностью распределения p(q). Для обучения модели вместо метода наибольшего правдоподобия при этом используется метод максимизации апостериорной оценки, то есть должны быть найдены параметры q, максимизирующие величину: $\prod_{i=1}^{n} P(c_i \mid d_i, \theta) \cdot p(\theta)$.

Мультиномиальная логистическая регрессия – это общий случай модели логистической регрессии, в которой зависимая переменная имеет более двух категорий. В модели мультиномиальной логистической регрессии для каждой категории зависимой переменной строится уравнение бинарной логистической регрессии. При этом одна из категорий зависимой переменной становится опорной, а все другие категории сравниваются с ней. Уравнение мультиномиальной логистической регрессии прогнозирует вероятность

принадлежности к каждой категории зависимой переменной по значениям независимых переменных.

Вообще говоря, логистическую регрессию можно представить в виде однослойной нейронной сети с сигмоидальной функцией активации, веса которой – коэффициенты логистической регрессии, а вес поляризации – константа регрессионного уравнения:

$P\{y = 1| x\} = f(z)$.

Преимущества метода:

- является одним из наиболее качественных;
- поддерживает инкрементное обучение;
- имеет относительно простую программную реализацию алгоритма.

Недостатки метода: сложная интерпретируемость параметров алгоритма и неустойчивость по отношению к выбросам в исходных данных.

1.6.2 Методы на основе искусственных нейронных сетей.

Существует большое количество разновидностей нейронных сетей, основные из них – сети прямого распространения, рекуррентные сети, радиально-базисные функции и самоорганизующиеся карты. Настройка весов может быть фиксированной или динамической.

В классических нейронных сетях прямого распространения (Feed Forward Back Propagation, FFBP) присутствуют входной слой, выходной слой и промежуточные слои: сигнал идет последовательно от входного слоя нейронов по промежуточным слоям к выходному. Примером такой структуры является многослойный перцептрон.

Для классификации документа d_i при помощи нейронной сети прямого распространения веса признаков документа подаются на соответствующие входы сети. Активация распространяется по сети; значения, получившиеся на

выходах, и есть результат классификации. Стандартный метод обучения такой сети – метод обратного распространения ошибки. Суть его в следующем: если на одном из выходов для одного из обучающих документов получен неправильный ответ, то ошибка распространяется обратно по сети и веса ребер меняются так, чтобы уменьшить ошибку.

Количество промежуточных слоев нейронной сети может быть не задано заранее, такую архитектуру называют динамической. В этом случае слои последовательно динамически генерируются до тех пор, пока не будет достигнут нужный уровень точности.

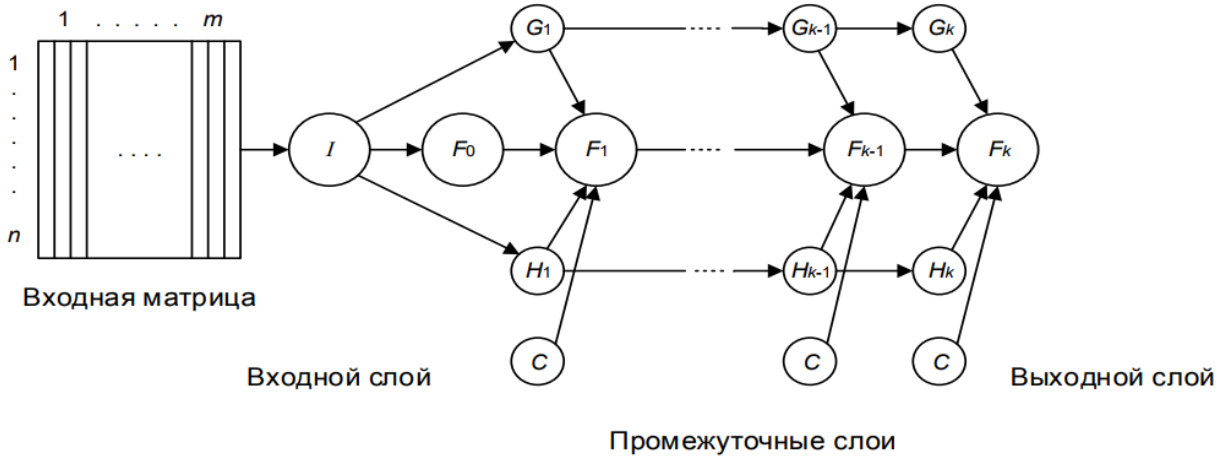

Рисунок 1.2 Структурная схема нейронной сети

Обобщенная схема DAN2 приведена на рисунке …. Каждый элемент F_k представляет собой функцию, которая содержит текущий элемент накопленных знаний (Current Accumulated Knowledge Element), полученный на предыдущем шаге обучения сети. С обозначают константы. Вершины G_k и H_k представляют собой текущие остаточные нелинейные компоненты процесса по передаточной функции взвешенной и нормализованной суммы входных переменных (Current Residual Nonlinear Element).

Сверточная нейронная сеть – однонаправленная многослойная сеть с применением операции свертки, при которой каждый фрагмент входных

данных умножается на матрицу (ядро) свертки поэлементно, а результат суммируется и записывается в аналогичную позицию выходных данных.

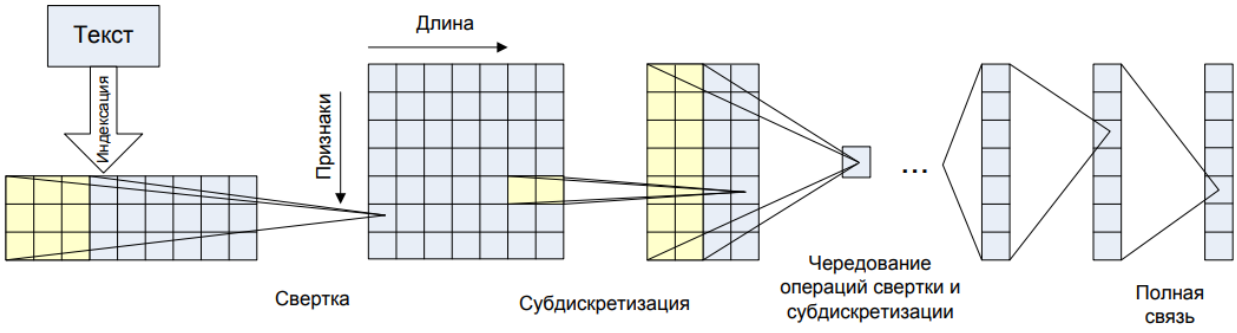

Рисунок 1.3. Сверточная нейронная сеть

Рекуррентная нейронная сеть получается из многослойного перцептрона введением обратных связей. Одна из широко распространенных разновидностей рекуррентных нейронных сетей – сеть Элмана – изображена на рисунке.

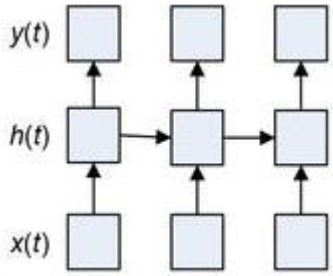

Рисунок 1.4. Нейронная сеть Элмана (разновидность рекуррентной сети)

В ней обратные связи идут не от выхода сети, а от выходов внутренних нейронов. Это позволяет учесть предысторию наблюдаемых процессов и накопить информацию для выработки правильной стратегии обучения. Главной особенностью рекуррентных нейронных сетей является запоминание последовательностей.

Скрытый слой h(t) в период времени t вычисляется путем преобразования текущего входного слоя x(t) и предыдущего скрытого слоя h(t – 1). Далее из скрытого слоя h(t) результат поступает на выходной слой y(t).

Преимущества метода:

- имеет очень высокое качество алгоритма при удачном подборе параметров;
- является универсальным аппроксиматором непрерывных функций;
- поддерживает инкрементное обучение.

Недостатки метода:

- вероятность возможной расходимости или медленной сходимости, поскольку для настройки сети используются градиентные методы;
- необходимость очень большого объема данных для обучения, чтобы достичь высокой точности;
- низкая скорость обучения;
- сложная интерпретируемость параметров алгоритма.

2 Сравнение алгоритмов

2.1 Методики оценки

Для сравнения различных алгоритмов классификации необходимо ввести какую-либо меру оценки, то есть некоторую функцию, которая описывает качество классификации. Существует несколько различных мер. Для того, чтобы их определить, введем понятие матрицы ошибок.

2.1.1 Матрица ошибок

Матрица ошибок для задачи классификации на два класса представляет собой таблицу 2x2:

	Y=1	Y=0
Y'=1	True Positive, истинно положительные (**TP**) – количество объектов, относящихся к классу 1 и классифицированных как 1	False Positive, ложноположительные (**FP**) – количество объектов, относящихся к классу 0 и классифицированных как 1 (ошибка)
Y'=0	False Negative, ложноотрицательные (**FN**) – количество объектов, относящихся к классу 1 и классифицированных как 0 (ошибка)	True Negative, истинно отрицательные (**TN**) – количество объектов, относящихся к классу 0 и классифицированных как 0

На основе такой матрицы можно вывести следующие меры:

2.1.2 Доля правильных ответов (accuracy)

Самой очевидной метрикой является доля правильных ответов, то есть отношение количества правильно классифицированных объектов к общему количеству объектов:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Эта метрика редко применяется на практике из-за того, что в случае, когда априорные вероятности классов заметно различаются, она перестает адекватно оценивать качество классификации. Тем не менее, в выборке, рассматриваемой в данной работе, априорные вероятности классов практически равны, поэтому данная метрика может быть применена.

Для сравнения различных моделей классификации текстов были проделаны следующие шаги. Задача классификации текстов разбивается на подзадачи векторизации текста (то есть приведения текстов к виду числовых векторов равной длины), трансформации векторов и собственно классификации. Для каждого шага было выбрано несколько различных моделей, решающих эти задачи, и проведено их сравнение (в данном разделе – по доле правильных ответов). Для сравнения шаги комбинировались во всех возможных сочетаниях, то есть для всех возможных троек *[векторизатор, трансформер, классификатор]*, составленных из элементов из нижеизложенного списка, измерялась точность классификации (по доле правильных ответов, а также по метрикам, описанным далее).

Сравнивались следующие алгоритмы:

1. Векторизация
   1.1. 1-граммы (отдельные слова)
   1.2. 1- и 2- граммы (отдельные слова и словосочетания длиной 2)
   1.3. 2-граммы (словосочетания длиной 2)
2. Трансформация
   2.1.Линейный TF
   2.2.Логарифмический TF
   2.3.Линейный TF-IDF
   2.4.Логарифмический TF-IDF

3. Классификация

3.1. Метод K ближайших соседей

3.2. Метод опорных векторов с линейным ядром

3.3. Метод опорных векторов с ядром RBF (радиальная базисная функция)

3.4. Дерево решений

3.5. Метод «случайного леса»

3.6. AdaBoost

3.7. Многомерный наивный Байесовский классификатор

3.8. Сверточные нейронные сети

Для сравнения моделей была написана вспомогательная программа на языке python (приложение 2). Она комбинирует алгоритмы во всех сочетаниях и фиксирует результаты работы каждой комбинации (то есть долю правильных ответов, точность, полноту и F-меру). Кроме этого, программа строит графики результатов, сгруппированные отдельно для каждого этапа классификации.

vectorizer	transformer	classifier	accuracy	f1_score	fscore_1	fscore_2	precision_1	precision_2	recall_1	recall_2
1-grams	TF l2 log	Nearest Neighbors	0,524	0,150	0,669	0,150	0,509	0,779	0,976	0,083
1-grams	TF l2 log	Linear SVM	0,675	0,677	0,672	0,677	0,668	0,680	0,675	0,674
1-grams	TF l2 log	RBF SVM	0,725	0,727	0,723	0,727	0,718	0,731	0,727	0,722
1-grams	TF l2 log	Decision Tree	0,615	0,619	0,611	0,619	0,609	0,620	0,612	0,617
1-grams	TF l2 log	Random Forest	0,495	0,009	0,661	0,009	0,494	0,727	0,998	0,005
1-grams	TF l2 log	AdaBoost	0,651	0,640	0,661	0,640	0,634	0,670	0,691	0,612
1-grams	TF l2 log	Naive Bayes	0,708	0,696	0,718	0,696	0,685	0,735	0,755	0,661
1-grams	TF l2 linear	Nearest Neighbors	0,526	0,160	0,670	0,160	0,510	0,777	0,974	0,089
1-grams	TF l2 linear	Linear SVM	0,673	0,675	0,671	0,675	0,666	0,679	0,675	0,671
1-grams	TF l2 linear	RBF SVM	0,722	0,724	0,719	0,724	0,716	0,727	0,723	0,721
1-grams	TF l2 linear	Decision Tree	0,615	0,619	0,610	0,619	0,609	0,620	0,611	0,618
1-grams	TF l2 linear	Random Forest	0,494	0,006	0,661	0,006	0,494	0,667	0,998	0,003
1-grams	TF l2 linear	AdaBoost	0,652	0,635	0,667	0,635	0,631	0,677	0,708	0,597
1-grams	TF l2 linear	Naive Bayes	0,707	0,695	0,718	0,695	0,684	0,734	0,755	0,660

1-grams	TF-IDF l2 log	Nearest Neighbors	0,555	0,369	0,656	0,369	0,530	0,653	0,860	0,257
1-grams	TF-IDF l2 log	Linear SVM	0,660	0,663	0,657	0,663	0,654	0,666	0,660	0,660
1-grams	TF-IDF l2 log	RBF SVM	0,717	0,718	0,716	0,718	0,709	0,726	0,724	0,710
1-grams	TF-IDF l2 log	Decision Tree	0,615	0,618	0,613	0,618	0,609	0,622	0,617	0,614
1-grams	TF-IDF l2 log	Random Forest	0,496	0,035	0,659	0,035	0,495	0,568	0,986	0,018
1-grams	TF-IDF l2 log	AdaBoost	0,650	0,632	0,666	0,632	0,629	0,675	0,707	0,594
1-grams	TF-IDF l2 log	Naive Bayes	0,711	0,699	0,721	0,699	0,687	0,739	0,759	0,664
1-grams	TF-IDF l2 linear	Nearest Neighbors	0,559	0,382	0,657	0,382	0,533	0,656	0,855	0,269
1-grams	TF-IDF l2 linear	Linear SVM	0,660	0,661	0,659	0,661	0,653	0,668	0,665	0,655
1-grams	TF-IDF l2 linear	RBF SVM	0,716	0,718	0,714	0,718	0,709	0,723	0,720	0,712
1-grams	TF-IDF l2 linear	Decision Tree	0,615	0,618	0,612	0,618	0,609	0,621	0,615	0,615
1-grams	TF-IDF l2 linear	Random Forest	0,493	0,002	0,660	0,002	0,493	0,385	0,998	0,001
1-grams	TF-IDF l2 linear	AdaBoost	0,649	0,639	0,660	0,639	0,633	0,668	0,688	0,612
1-grams	TF-IDF l2 linear	Naive Bayes	0,709	0,697	0,719	0,697	0,686	0,735	0,755	0,663
1- and 2-grams	TF l2 log	Nearest Neighbors	0,530	0,183	0,670	0,183	0,512	0,762	0,967	0,104
1- and 2-grams	TF l2 log	Linear SVM	0,669	0,640	0,694	0,640	0,638	0,713	0,760	0,580
1- and 2-grams	TF l2 log	RBF SVM	0,724	0,726	0,721	0,726	0,719	0,729	0,724	0,724
1- and 2-grams	TF l2 log	Decision Tree	0,615	0,609	0,621	0,609	0,604	0,627	0,639	0,592
1- and 2-grams	TF l2 log	Random Forest	0,498	0,022	0,662	0,022	0,496	0,789	0,997	0,011
1- and 2-grams	TF l2 log	AdaBoost	0,658	0,649	0,667	0,649	0,643	0,676	0,693	0,625
1- and 2-grams	TF l2 log	Naive Bayes	0,718	0,715	0,722	0,715	0,704	0,734	0,741	0,697
1- and 2-grams	TF l2 linear	Nearest Neighbors	0,531	0,190	0,670	0,190	0,513	0,762	0,965	0,109
1- and 2-grams	TF l2 linear	Linear SVM	0,669	0,640	0,693	0,640	0,638	0,712	0,759	0,580
1- and 2-grams	TF l2 linear	RBF SVM	0,722	0,725	0,719	0,725	0,718	0,727	0,721	0,724
1- and 2-grams	TF l2 linear	Decision Tree	0,616	0,617	0,614	0,617	0,609	0,623	0,620	0,612
1- and 2-grams	TF l2 linear	Random Forest	0,493	0,006	0,660	0,006	0,493	0,421	0,996	0,003
1- and 2-grams	TF l2 linear	AdaBoost	0,657	0,647	0,666	0,647	0,640	0,675	0,694	0,620
1- and 2-grams	TF l2 linear	Naive Bayes	0,717	0,714	0,720	0,714	0,704	0,732	0,738	0,697

1- and 2-grams	TF-IDF l2 log	Nearest Neighbors	0,562	0,367	0,665	0,367	0,534	0,685	0,882	0,250
1- and 2-grams	TF-IDF l2 log	Linear SVM	0,659	0,655	0,663	0,655	0,647	0,672	0,679	0,639
1- and 2-grams	TF-IDF l2 log	RBF SVM	0,722	0,725	0,720	0,725	0,717	0,727	0,722	0,722
1- and 2-grams	TF-IDF l2 log	Decision Tree	0,612	0,604	0,620	0,604	0,600	0,626	0,642	0,584
1- and 2-grams	TF-IDF l2 log	Random Forest	0,495	0,007	0,661	0,007	0,494	0,739	0,999	0,003
1- and 2-grams	TF-IDF l2 log	AdaBoost	0,656	0,643	0,668	0,643	0,637	0,677	0,701	0,612
1- and 2-grams	TF-IDF l2 log	Naive Bayes	0,715	0,715	0,716	0,715	0,705	0,726	0,727	0,704
1- and 2-grams	TF-IDF l2 linear	Nearest Neighbors	0,562	0,377	0,663	0,377	0,535	0,675	0,871	0,261
1- and 2-grams	TF-IDF l2 linear	Linear SVM	0,659	0,655	0,663	0,655	0,647	0,672	0,680	0,639
1- and 2-grams	TF-IDF l2 linear	RBF SVM	0,721	0,724	0,718	0,724	0,716	0,726	0,721	0,721
1- and 2-grams	TF-IDF l2 linear	Decision Tree	0,611	0,596	0,624	0,596	0,596	0,628	0,656	0,566
1- and 2-grams	TF-IDF l2 linear	Random Forest	0,496	0,022	0,661	0,022	0,495	0,659	0,994	0,011
1- and 2-grams	TF-IDF l2 linear	AdaBoost	0,656	0,643	0,667	0,643	0,638	0,677	0,700	0,612
1- and 2-grams	TF-IDF l2 linear	Naive Bayes	0,713	0,712	0,713	0,712	0,703	0,723	0,724	0,702
2-grams	TF l2 log	Nearest Neighbors	0,586	0,531	0,629	0,531	0,564	0,622	0,711	0,463
2-grams	TF l2 log	Linear SVM	0,529	0,218	0,662	0,218	0,512	0,681	0,937	0,130
2-grams	TF l2 log	RBF SVM	0,653	0,668	0,636	0,668	0,659	0,647	0,614	0,690
2-grams	TF l2 log	Decision Tree	0,532	0,347	0,635	0,347	0,516	0,592	0,827	0,245
2-grams	TF l2 log	Random Forest	0,494	0,003	0,661	0,003	0,494	0,778	1,000	0,001
2-grams	TF l2 log	AdaBoost	0,551	0,378	0,648	0,378	0,528	0,632	0,839	0,270
2-grams	TF l2 log	Naive Bayes	0,648	0,628	0,667	0,628	0,626	0,677	0,713	0,585
2-grams	TF l2 linear	Nearest Neighbors	0,587	0,534	0,630	0,534	0,565	0,624	0,711	0,467
2-grams	TF l2 linear	Linear SVM	0,529	0,221	0,662	0,221	0,512	0,680	0,936	0,132
2-grams	TF l2 linear	RBF SVM	0,653	0,669	0,636	0,669	0,660	0,648	0,614	0,692
2-grams	TF l2 linear	Decision Tree	0,532	0,344	0,636	0,344	0,516	0,591	0,828	0,243
2-grams	TF l2 linear	Random Forest	0,494	0,006	0,661	0,006	0,494	0,682	0,999	0,003
2-grams	TF l2 linear	AdaBoost	0,551	0,367	0,652	0,367	0,528	0,640	0,852	0,257
2-grams	TF l2 linear	Naive Bayes	0,647	0,627	0,666	0,627	0,625	0,675	0,711	0,585
2-grams	TF-IDF l2 log	Nearest Neighbors	0,578	0,507	0,631	0,507	0,555	0,621	0,732	0,429
2-grams	TF-IDF l2 log	Linear SVM	0,545	0,360	0,647	0,360	0,524	0,627	0,846	0,253

2-grams	TF-IDF l2 log	RBF SVM	0,646	0,663	0,626	0,663	0,653	0,639	0,602	0,688
2-grams	TF-IDF l2 log	Decision Tree	0,532	0,321	0,643	0,321	0,516	0,605	0,853	0,219
2-grams	TF-IDF l2 log	Random Forest	0,494	0,006	0,661	0,006	0,494	0,700	0,999	0,003
2-grams	TF-IDF l2 log	AdaBoost	0,549	0,369	0,649	0,369	0,527	0,632	0,844	0,261
2-grams	TF-IDF l2 log	Naive Bayes	0,648	0,629	0,665	0,629	0,627	0,674	0,708	0,590
2-grams	TF-IDF l2 linear	Nearest Neighbors	0,579	0,509	0,632	0,509	0,556	0,622	0,732	0,430
2-grams	TF-IDF l2 linear	Linear SVM	0,546	0,359	0,648	0,359	0,524	0,628	0,847	0,252
2-grams	TF-IDF l2 linear	RBF SVM	0,646	0,664	0,626	0,664	0,654	0,639	0,600	0,691
2-grams	TF-IDF l2 linear	Decision Tree	0,532	0,322	0,643	0,322	0,516	0,605	0,853	0,220
2-grams	TF-IDF l2 linear	Random Forest	0,495	0,008	0,661	0,008	0,494	0,833	0,999	0,004
2-grams	TF-IDF l2 linear	AdaBoost	0,550	0,377	0,647	0,377	0,528	0,630	0,837	0,269
2-grams	TF-IDF l2 linear	Naive Bayes	0,649	0,630	0,666	0,630	0,627	0,675	0,709	0,590
Word2Vec		CNN	0,772	0,766	0,775	0,766	0,763	0,779	0,788	0,753

Для каждой модели векторизации, трансформации и классификации было найдено среднее и максимальное значение точности в сочетаниях со всеми остальными параметрами.

Средние:

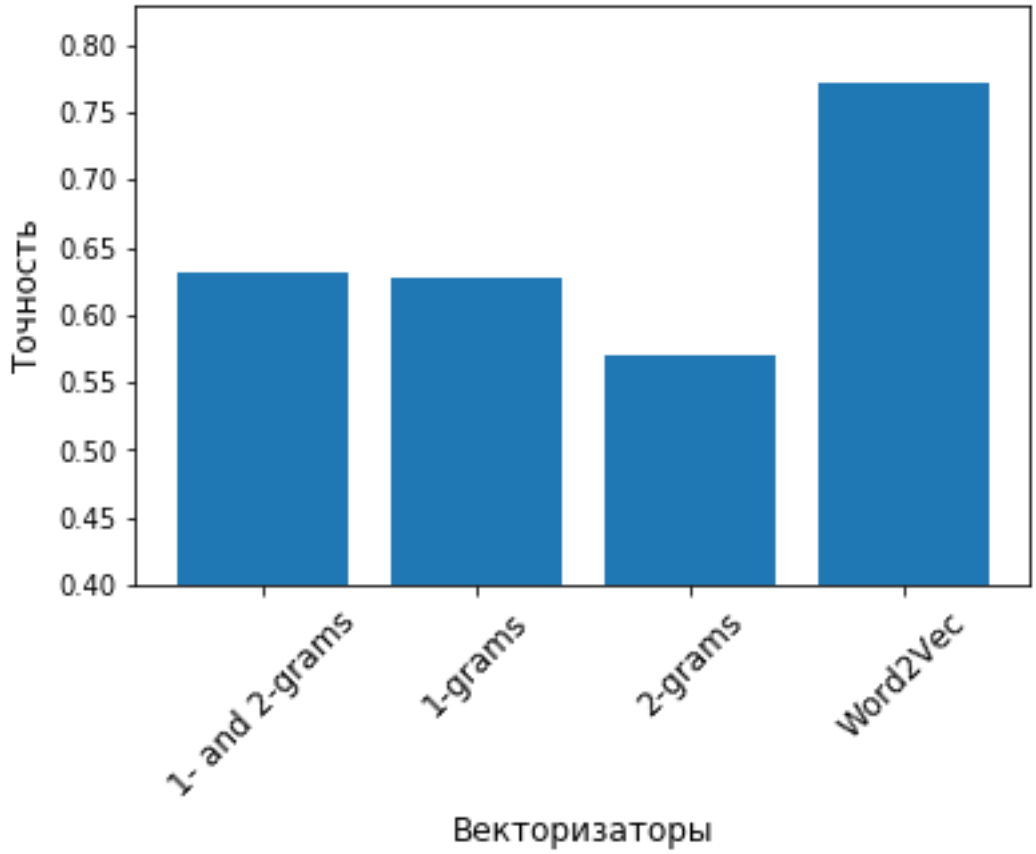

Рисунок 2.1 Средняя доля правильных ответов для векторизаторов

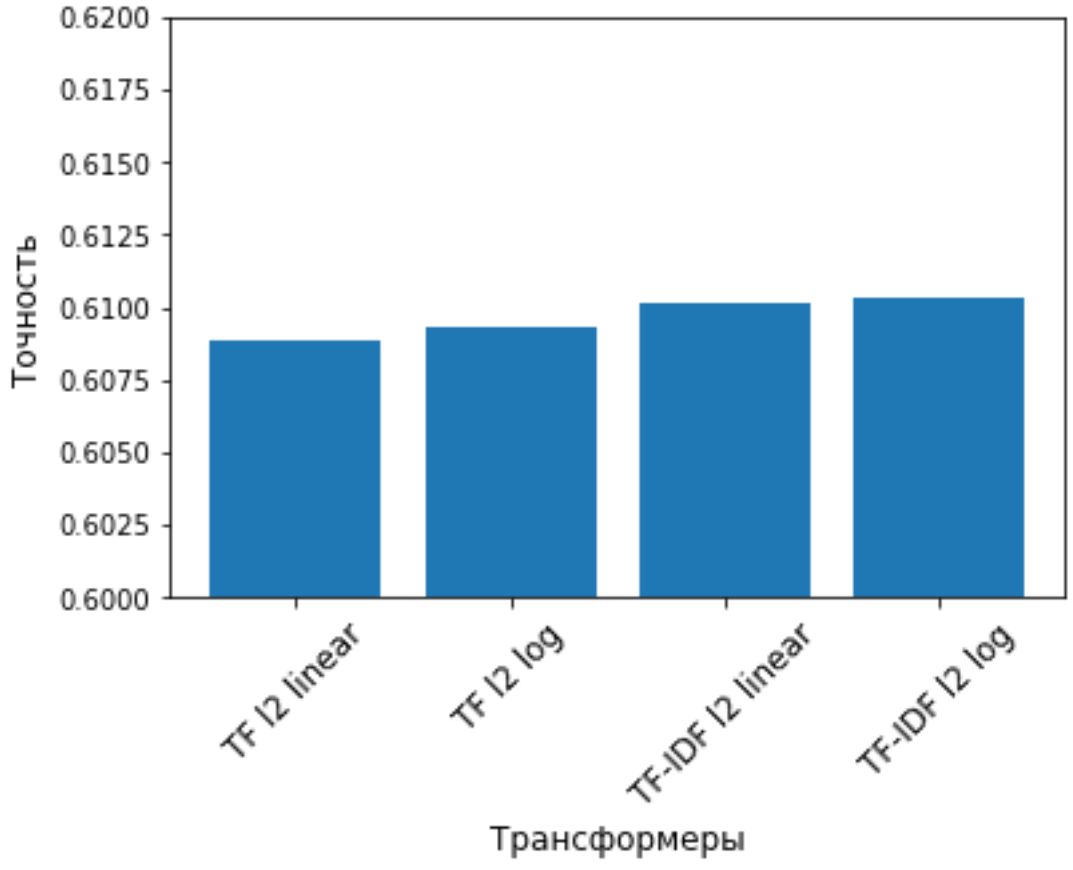

Рисунок 2.2 Средняя доля правильных ответов для трансформеров

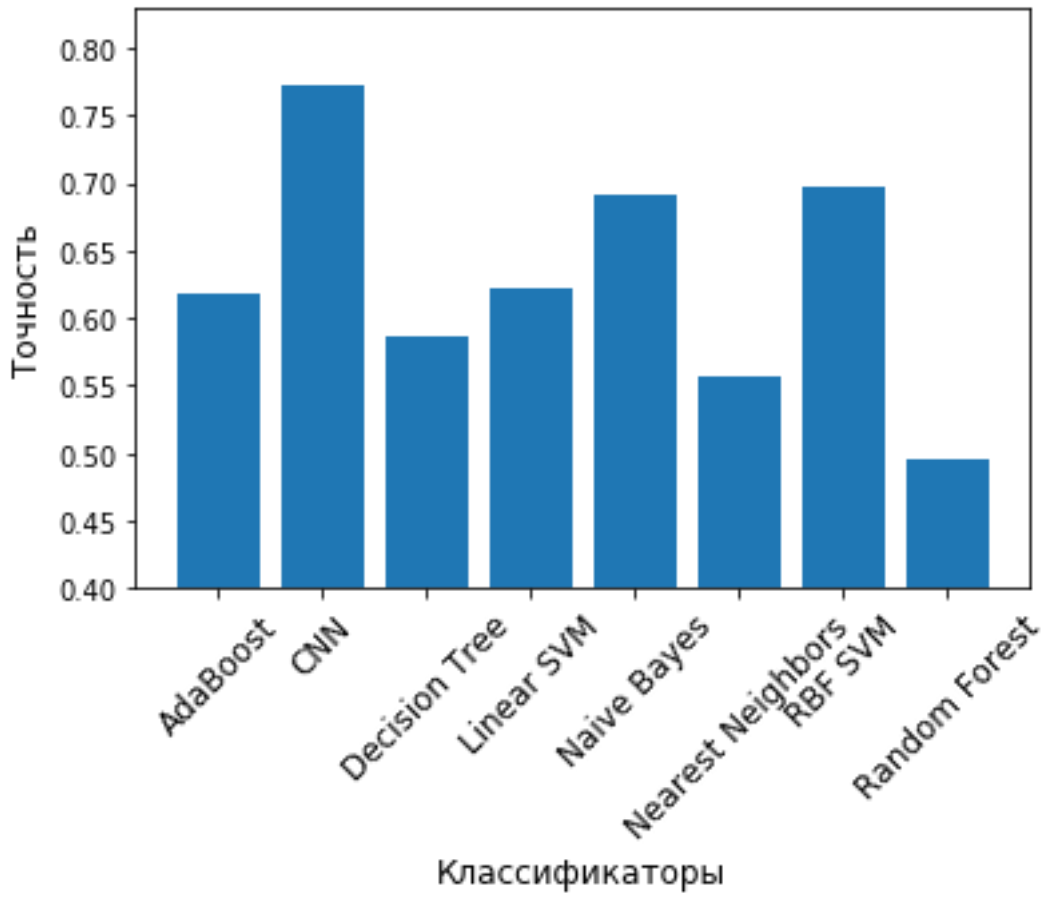

Рисунок 2.3 Средняя доля правильных ответов для классификаторов

Лучшие:

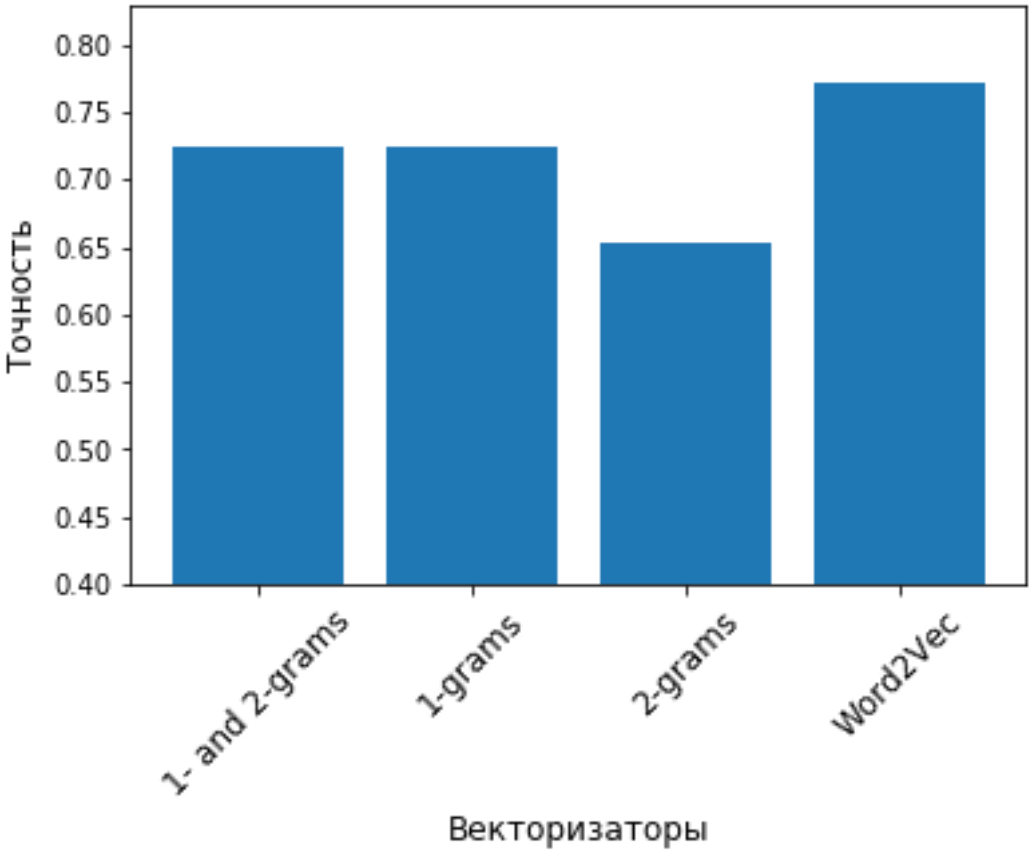

Рисунок 2.4 Наилучшая доля правильных ответов для векторизаторов

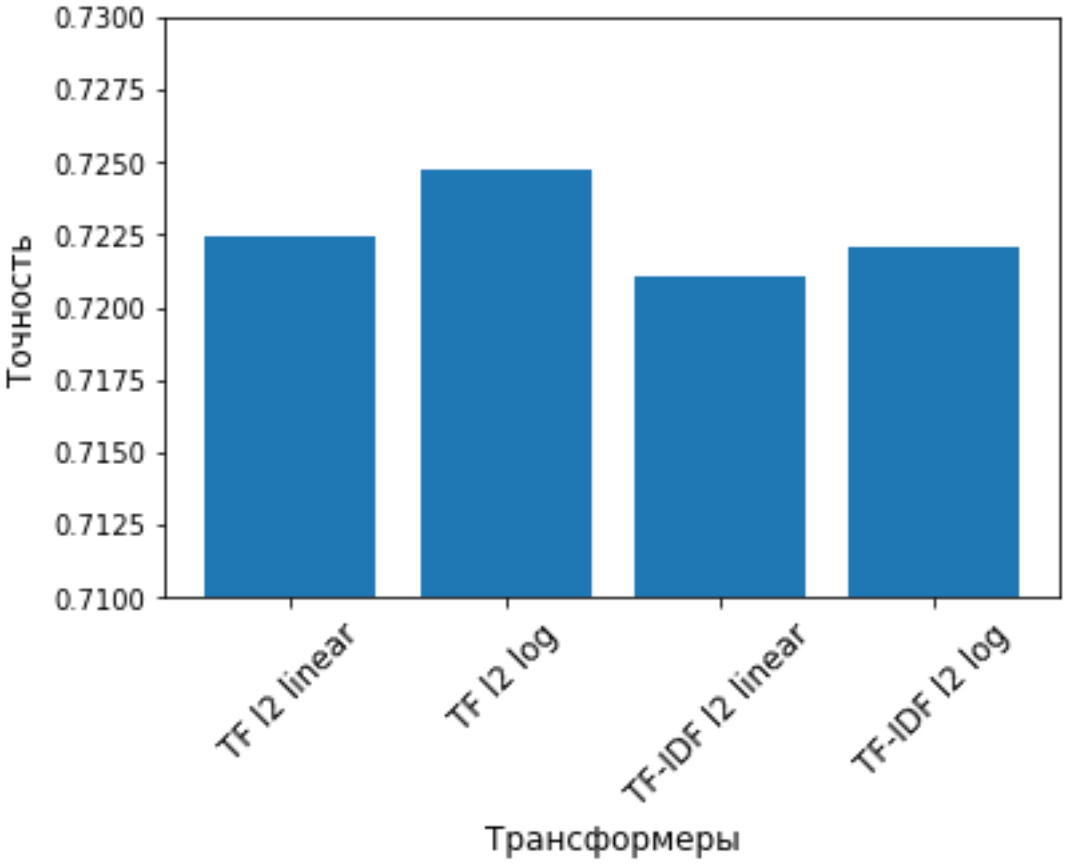

Рисунок 2.5Наилучшая доля правильных ответов для трансформеров

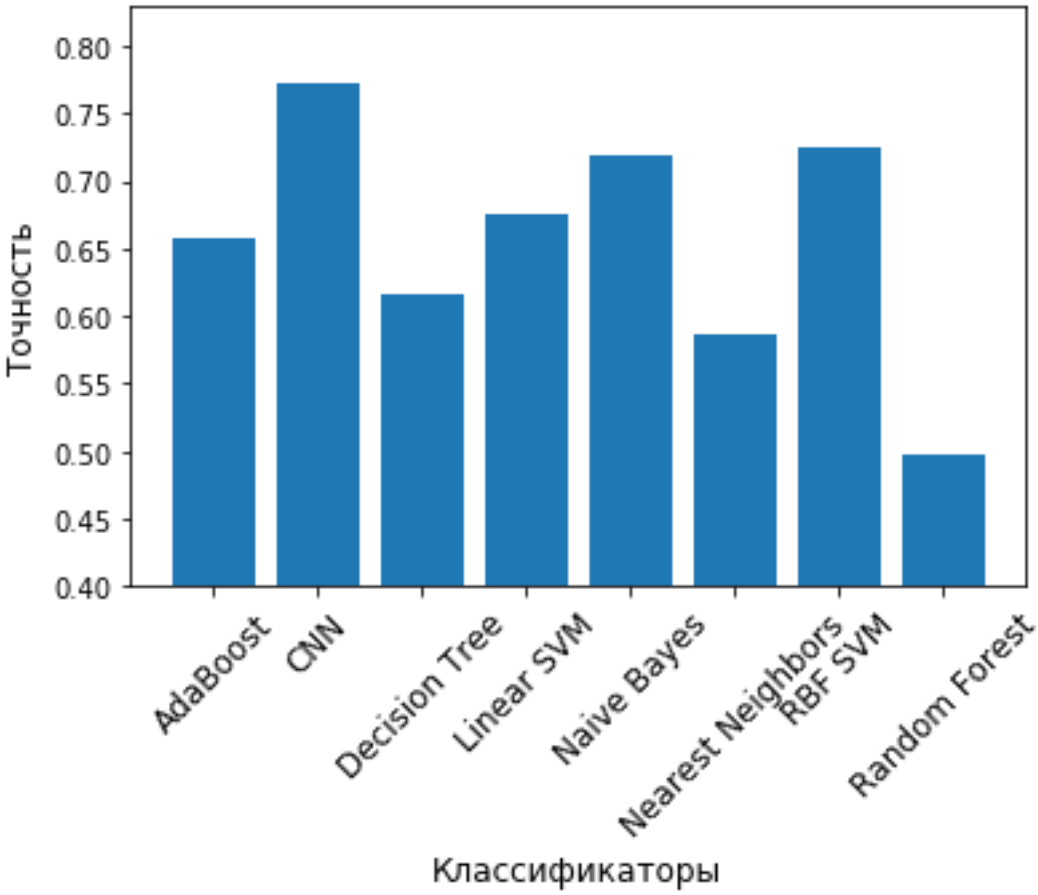

Рисунок 2.6 Наилучшая доля правильных ответов для классификаторов

2.1.3 Precision, recall и F-мера

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота).

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

Именно введение precision не позволяет записывать все объекты в один класс, так как это ведет к росту уровня False Positive. Recall демонстрирует способность алгоритма обнаруживать данный класс вообще, а precision — способность отличать этот класс от других классов.

Как отмечено ранее, ошибки классификации бывают двух видов: False Positive и False Negative. В статистике первый вид ошибок называют ошибкой I-го рода, а второй — ошибкой II-го рода.

Precision и recall не зависят, в отличие от accuracy, от соотношения классов и потому применимы в условиях несбалансированных выборок.

Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. F-мера (в общем случае F_β) — среднее гармоническое precision и recall :

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

β в данном случае определяет вес точности в метрике, и при $\beta = 1$ это среднее гармоническое (с множителем 2, чтобы в случае precision = 1 и recall = 1 иметь $F_1 = 1$)

F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю.

Таким образом, F-мера является хорошим критерием для оценки качества классификаторов. Как и в случае с оценкой доли верно классифицированных объектов, для каждого векторизатора, трансформера и классификатора была вычислена усредненная F-мера для всех сочетаний остальных параметров.

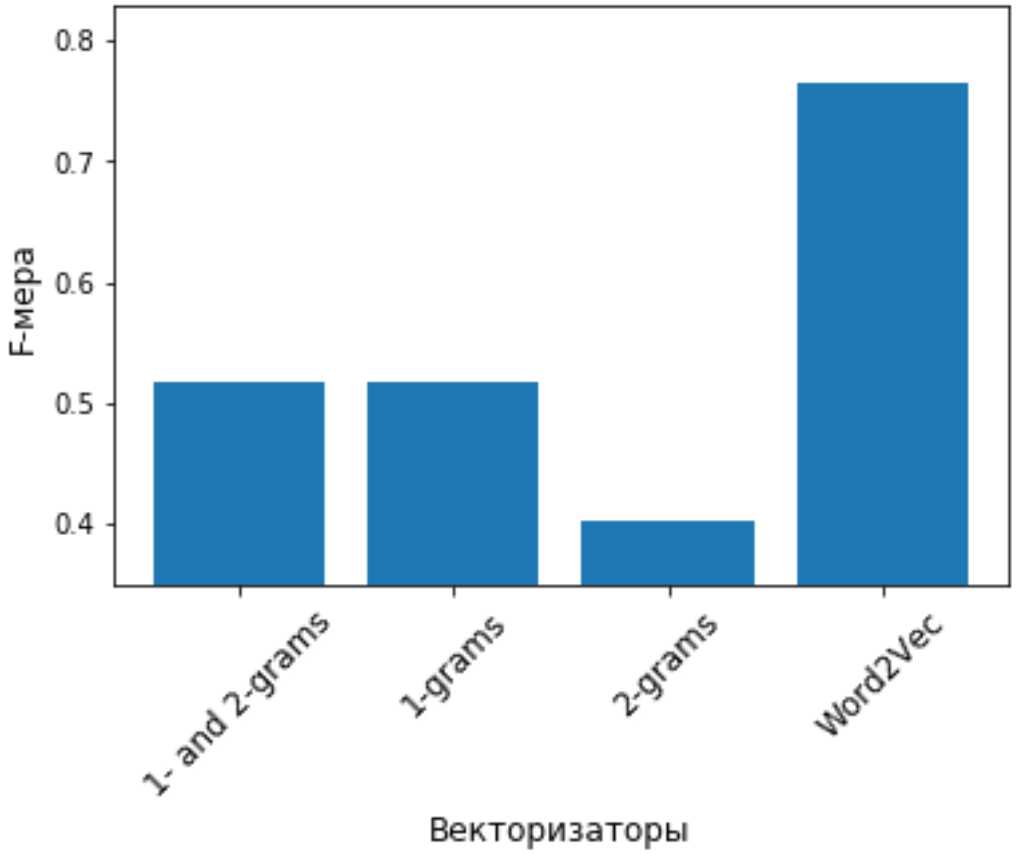

Рисунок 2.7 Средняя F-мера для векторизаторов

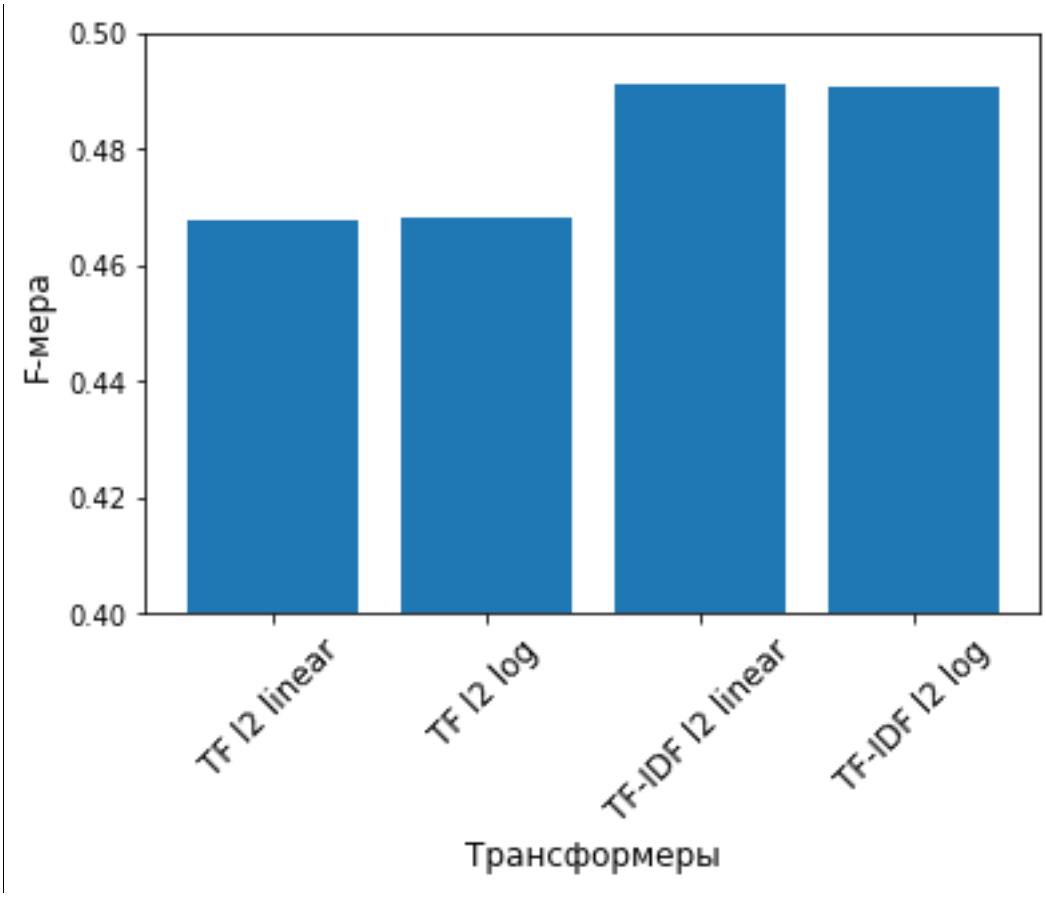

Рисунок 2.8 Средняя F-мера для трансформеров

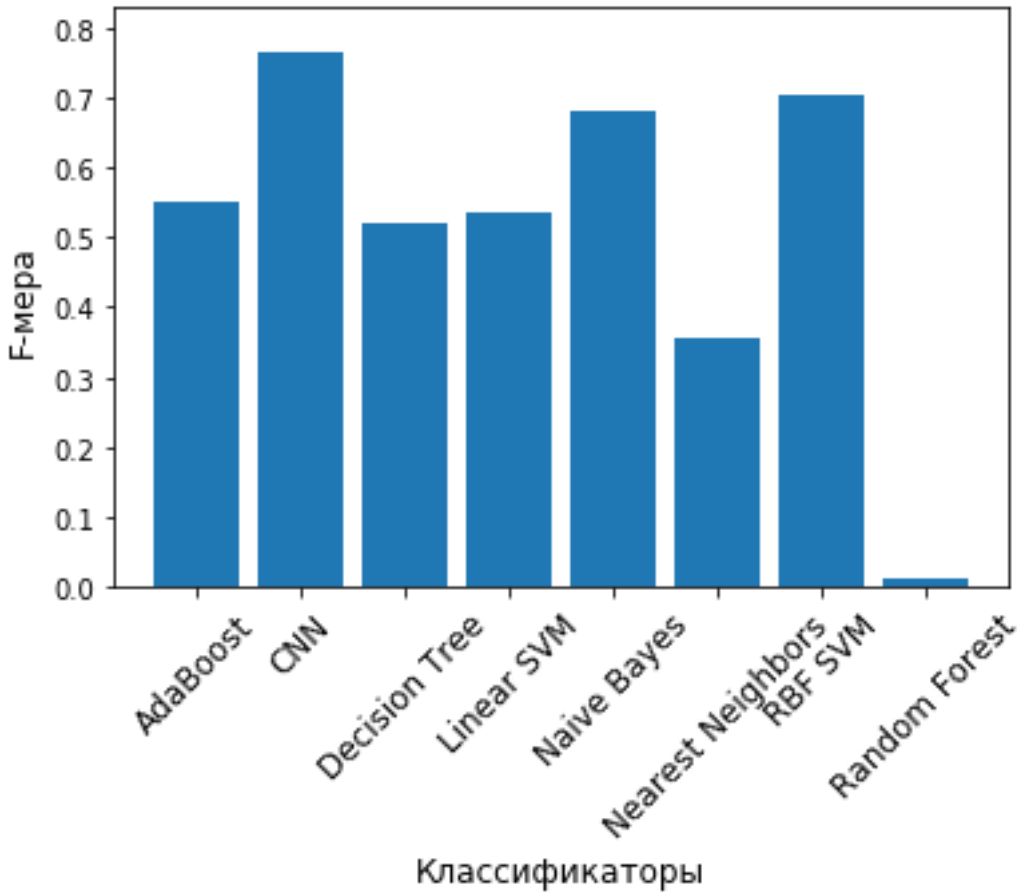

Рисунок 2.9 Средняя F-мера для классификаторов

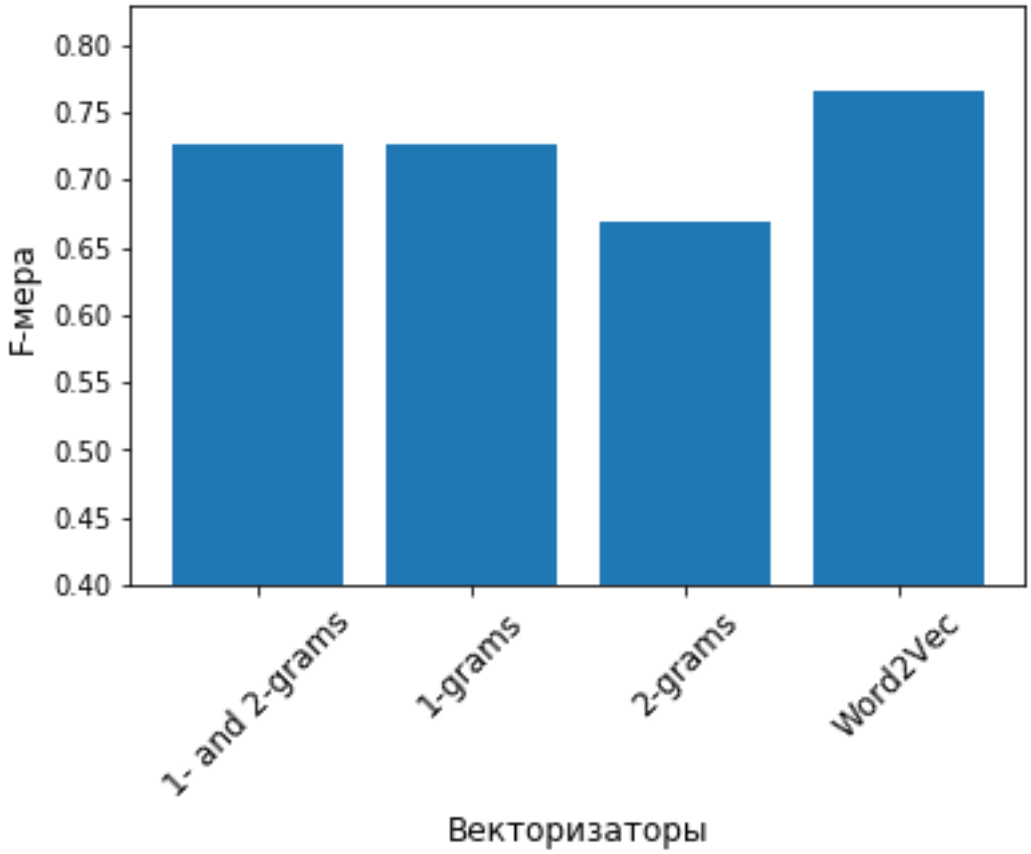

Рисунок 2.10 Наилучшая F-мера для векторизаторов

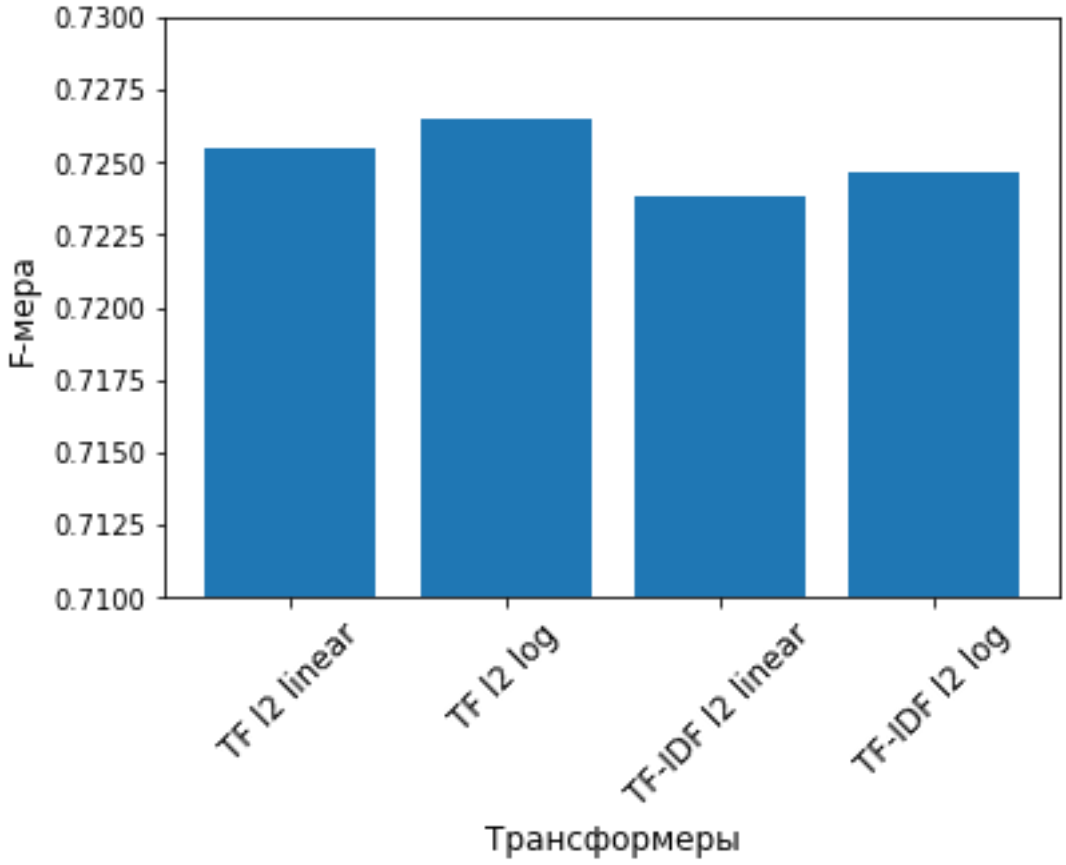

Рисунок 2.11 Наилучшая F-мера для трансформеров

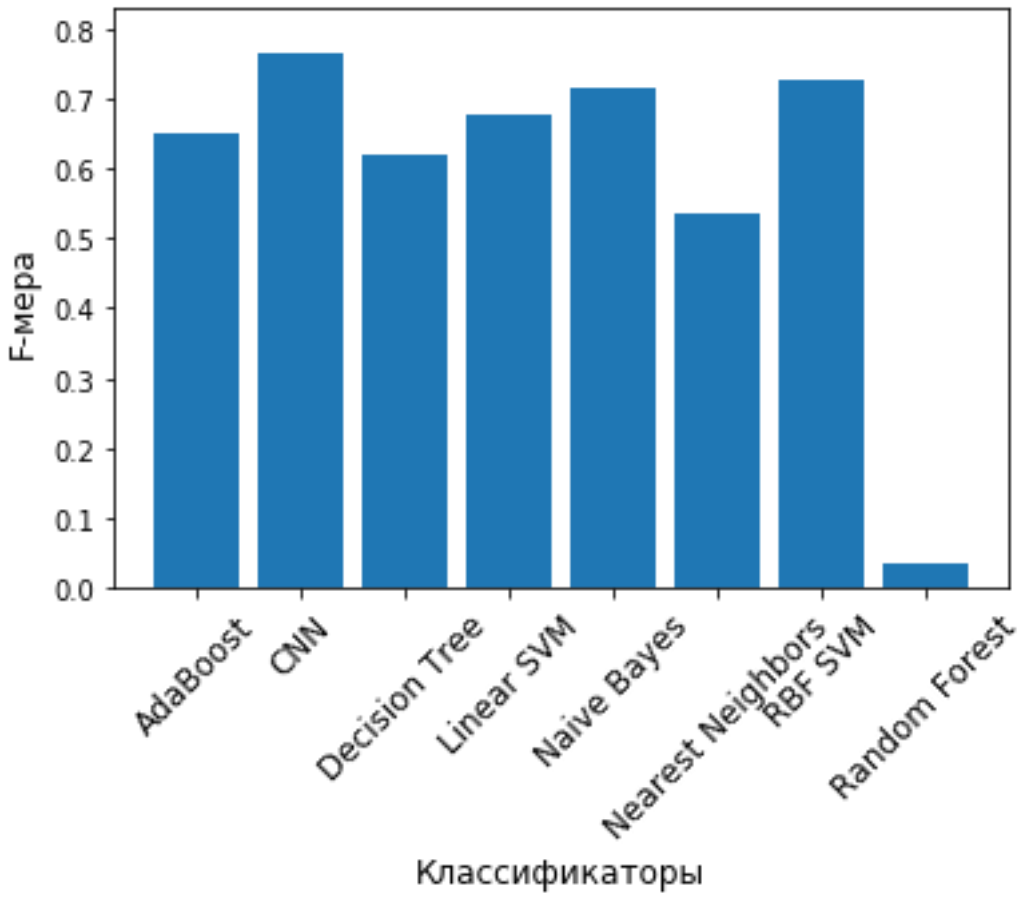

Рисунок 2.12 Наилучшая F-мера для векторизаторов

Таким образом, можно сделать следующие выводы:

1. Измерение доли верно классифицированных объектов не всегда позволяет верно оценить качество классификации. В отличие от нее, F-мера позволяет точнее выявлять некачественные классификаторы. Например, на графиках можно увидеть, что использование 2-грамм и «случайного леса» приводит к неудовлетворительным результатам.
2. Сверточные нейронные сети превосходят остальные алгоритмы и для средних, и для максимальных параметров. Ближайшие по точности – классификатор на опорных векторах с RBF-ядром и многомерный Байесовский классификатор

3 Архитектура нейронной сети

3.1 Основные определения

3.1.1 Нейрон

3.1.1.1 Биологический нейрон

Развитие искусственных нейронных сетей вдохновляется биологией. То есть рассматривая сетевые конфигурации и алгоритмы, исследователи мыслят их в терминах организации мозговой деятельности. Но наши знания о работе мозга столь ограничены, что мало бы нашлось руководящих ориентиров для тех, кто стал бы ему подражать. Поэтому разработчикам сетей приходится выходить за пределы современных биологических знаний в поисках структур, способных выполнять полезные функции. Во многих случаях это приводит к необходимости отказа от биологического правдоподобия, мозг становится просто метафорой, и создаются сети, невозможные в живой материи или требующие неправдоподобно больших допущений об анатомии и функционировании мозга.

Нервная система человека, построенная из элементов, называемых нейронами, имеет весьма высокую сложность. Около 1011 нейронов участвуют в примерно 1015 передающих связях, имеющих длину метр и более. Каждый нейрон обладает многими качествами, общими с другими элементами тела, но его уникальной способностью является прием, обработка и передача электрохимических сигналов по нервным путям, которые образуют коммуникационную систему мозга.

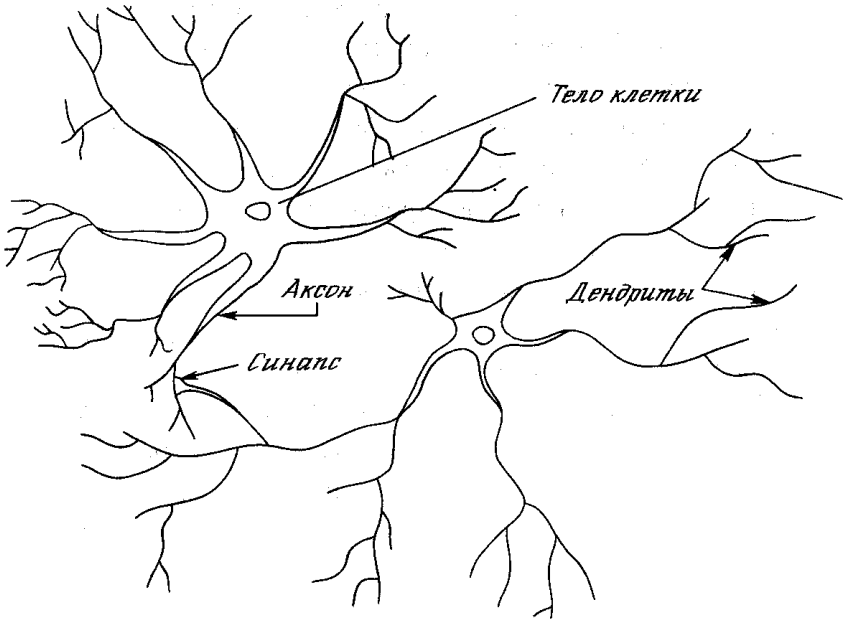

Рисунок 3.1 Биологический нейрон

На рисунке 3.1 показана структура пары типичных биологических нейронов. Дендриты идут от тела нервной клетки к другим нейронам, где они принимают сигналы в точках соединения, называемых синапсами. Принятые синапсом входные сигналы подводятся к телу нейрона. Здесь они суммируются, причем одни входы стремятся возбудить нейрон, другие – воспрепятствовать его возбуждению. Когда суммарное возбуждение в теле нейрона превышает некоторый порог, нейрон возбуждается, посылая по аксону сигнал другим нейронам. У этой основной функциональной схемы много усложнений и исключений, тем не менее большинство искусственных нейронных сетей моделируют лишь эти простые свойства.

Рассмотренная далее простая модель искусственного нейрона игнорирует многие свойства своего биологического двойника. Например, она не принимает во внимание задержки во времени, которые воздействуют на динамику системы. Входные сигналы сразу же порождают выходной сигнал. И, что более важно, она не учитывает воздействий функции частотной модуляции или синхронизирующей функции биологического нейрона, которые ряд исследователей считают решающими.

Несмотря на эти ограничения, сети, построенные из этих нейронов, обнаруживают свойства, сильно напоминающие биологическую систему.

3.1.1.2 Искусственный нейрон

Искусственный нейрон имитирует в первом приближении свойства биологического нейрона. На вход искусственного нейрона поступает некоторое множество сигналов, каждый из которых является выходом другого нейрона. Каждый вход умножается на соответствующий вес, аналогичный синаптической силе, и все произведения суммируются, определяя уровень активации нейрона. На рис. 3.2 представлена модель, реализующая эту идею. Хотя сетевые парадигмы весьма разнообразны, в основе почти всех их лежит эта конфигурация. Здесь множество входных сигналов, обозначенных x_1, x_2,…, x_n, поступает на искусственный нейрон. Эти входные сигналы, в совокупности обозначаемые вектором **X**, соответствуют сигналам, приходящим в синапсы биологического нейрона. Каждый сигнал умножается на соответствующий вес w_1, w_2,…, w_n, и поступает на суммирующий блок, обозначенный Σ. Каждый вес соответствует “силе” одной биологической синаптической связи. (Множество весов в совокупности обозначается вектором **W**.) Суммирующий блок, соответствующий телу биологического элемента, складывает взвешенные входы алгебраически, создавая выход, который мы будем называть NET. В векторных обозначениях это может быть компактно записано следующим образом:

NET = **XW**.

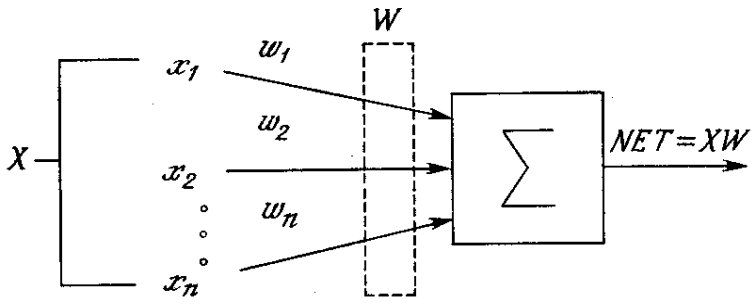

Рисунок 3.2 Искусственный нейрон

Сигнал NET далее, как правило, преобразуется активационной функцией F и дает выходной нейронный сигнал OUT. Активационная функция может быть обычной линейной функцией

OUT = K(NET),

где K – постоянная, пороговой функции

OUT = 1, если NET > T,

OUT = 0 в остальных случаях,

где T – некоторая постоянная пороговая величина, или же функцией, более точно моделирующей нелинейную передаточную характеристику биологического нейрона и представляющей нейронной сети большие возможности. О функциях активации будет подробно рассказано в следующем разделе.

3.1.2 Функции активации

Функция активации - функция, вычисляющая выходной сигнал искусственного нейрона. В качестве аргумента принимает сигнал Y, получаемый на выходе входного сумматора.

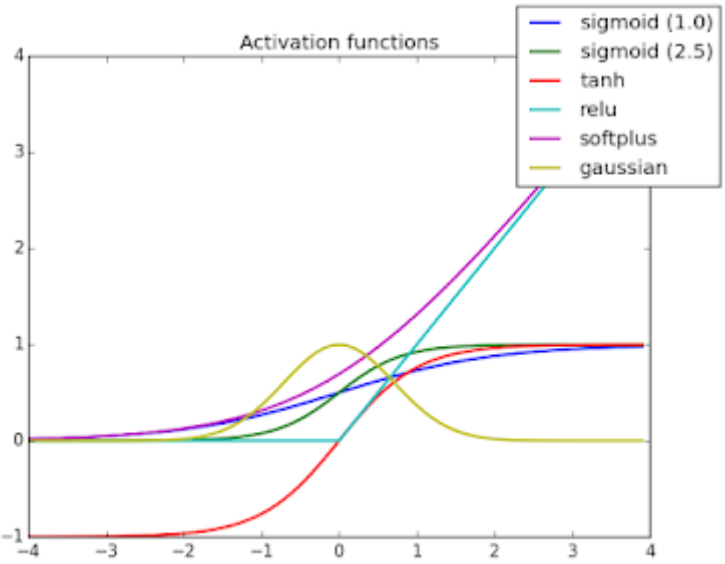

Рисунок 3.3 Графики основных функций активации

На рисунке 3.3 представлены графики основных функций активации:

1. Сигмоидальная с a=1
2. Сигмоидальная с a=2.5
3. Гиперболический тангенс
4. ReLU (выпрямитель)

5. Softplus
6. Гауссова

3.1.2.1 Логистическая (сигмоидальная) функция

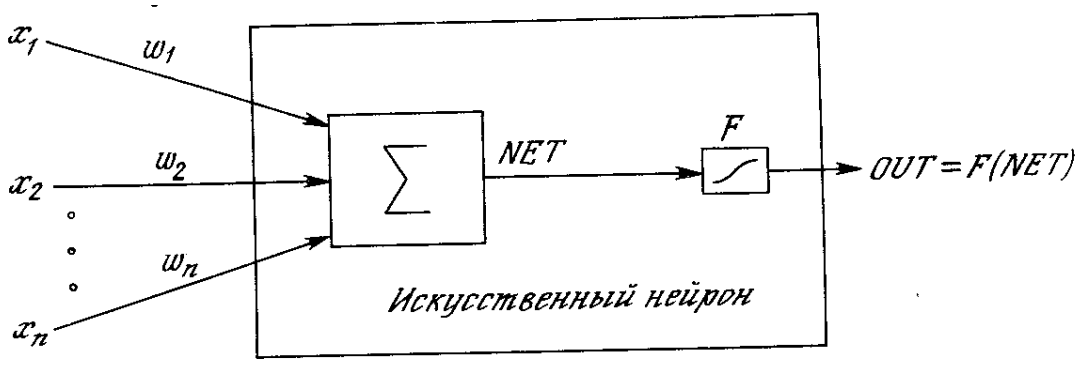

Рисунок 3.4 Искусственный нейрон с активационной функцией

На рис. 1.3 блок, обозначенный *F*, принимает сигнал NET и выдает сигнал OUT. Если блок *F* сужает диапазон изменения величины NET так, что при любых значениях NET значения OUT принадлежат некоторому конечному интервалу, то *F* называется *“сжимающей” функцией.* В качестве “сжимающей” функции часто используется логистическая или “сигмоидальная” (S-образная) функция, показанная на рис. 1.4а. Эта функция математически выражается как $F(x) = 1/(1 + e^{-x})$. Таким образом,

$$\mathrm{OUT} = \frac{1}{1 + e^{-\mathrm{NET}}}.$$

По аналогии с электронными системами активационную функцию можно считать нелинейной усилительной характеристикой искусственного нейрона. Коэффициент усиления вычисляется как отношение приращения величины OUT к вызвавшему его небольшому приращению величины NET. Он выражается наклоном кривой при определенном уровне возбуждения и изменяется от малых значений при больших отрицательных возбуждениях (кривая почти горизонтальна) до максимального значения при нулевом возбуждении и снова уменьшается, когда возбуждение становится большим положительным. Подобная нелинейная характеристика решает дилемму

шумового насыщения. Каким образом одна и та же сеть может обрабатывать как слабые, так и сильные сигналы? Слабые сигналы нуждаются в большом сетевом усилении, чтобы дать пригодный к использованию выходной сигнал. Однако усилительные каскады с большими коэффициентами усиления могут привести к насыщению выхода шумами усилителей (случайными флуктуациями), которые присутствуют в любой физически реализованной сети. Сильные входные сигналы в свою очередь также будут приводить к насыщению усилительных каскадов, исключая возможность полезного использования выхода. Центральная область логистической функции, имеющая большой коэффициент усиления, решает проблему обработки слабых сигналов, в то время как области с падающим усилением на положительном и отрицательном концах подходят для больших возбуждений. Таким образом, нейрон функционирует с большим усилением в широком диапазоне уровня входного сигнала.

$$OUT = \frac{1}{1 + e^{-NET}} = F(NET)$$

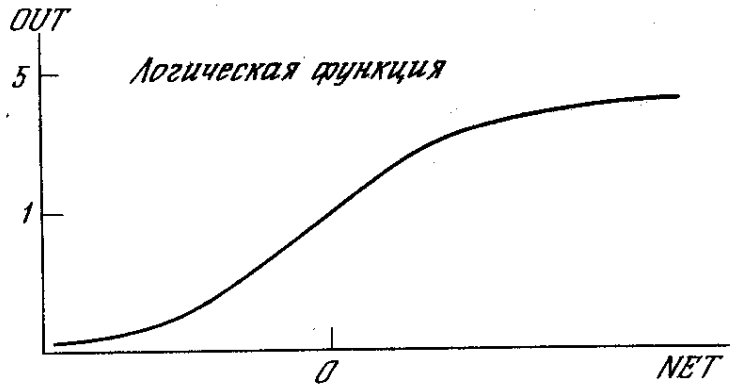

Рисунок 3.5 Сигмоидальная логисчтическая фунцкия

3.1.2.2 Приближенно-сигмоидальная функция (hard sigmoid)

Сигмоидальная функция обладает многими преимуществами, но относительно долго вычисляется, так как использует операцию возведения в степень. Для случаев, когда гладкостью и непрерывной дифференцироемостью можно пренебречь, используется приближенно-сигмоидальная функция – кусочно-линейная функция, напоминающая своей

формой сигмоиду. Ее форма близка к сигмоиде, но вычисление значительно быстрее, так как при вычислении используются только операции сложения и умножения.

В использованной в работе библиотеке Keras используется следующая реализация hard_sigmoid:

- `0` if `x < -2.5`
- `1` if `x > 2.5`
- `0.2 * x + 0.5` if `-2.5 <= x <= 2.5`.

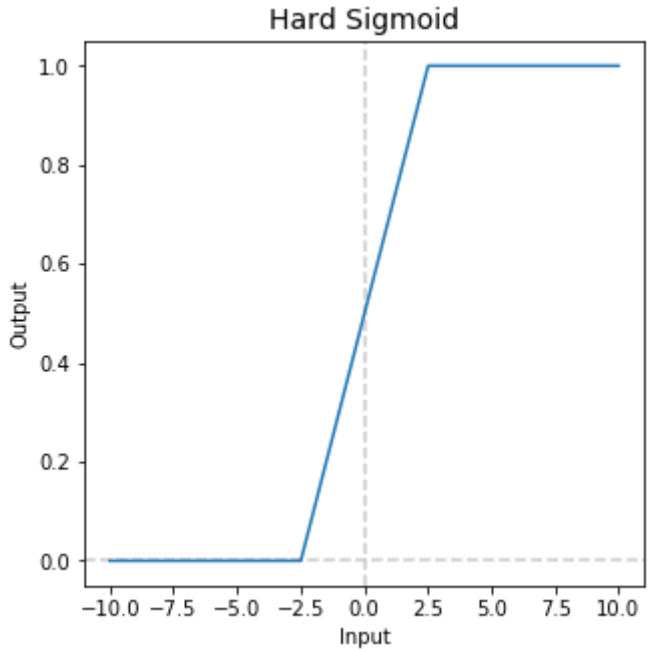

Рисунок 3.6 Приближенно-сигмоидальная фунцкия

3.1.2.3 Гиперболический тангенс

Другой широко используемой активационной функцией является гиперболический тангенс. По форме она сходна с логистической функцией и часто используется биологами в качестве математической модели активации нервной клетки. В качестве активационной функции искусственной нейронной сети она записывается следующим образом:

OUT = th(x).

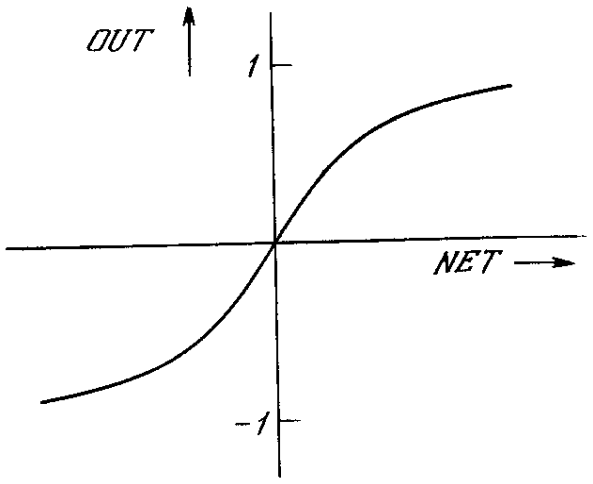

Рисунок 3.7 Функция гиперболического тангенса

Подобно логистической функции гиперболический тангенс является S-образной функцией, но он симметричен относительно начала координат, и в точке NET = 0 значение выходного сигнала OUT равно нулю (см. рис. 1.4б). В отличие от логистической функции гиперболический тангенс принимает значения различных знаков, что оказывается выгодным для ряда сетей (см. гл. 3).

3.1.2.4 ReLU (выпрямитель)

Известно, что нейронные сети способны приблизить сколь угодно сложную функцию, если в них достаточно слоев и функция активации является нелинейной. Функции активации вроде сигмоидной или тангенциальной являются нелинейными, но приводят к проблемам с затуханием или увеличением градиентов. Однако можно использовать и гораздо более простой вариант — выпрямленную линейную функцию активации (rectified linear unit, ReLU), которая выражается формулой:

$$f(s) = \max(0, s)$$

График функции ReLU в соответствии с рисунком ниже:

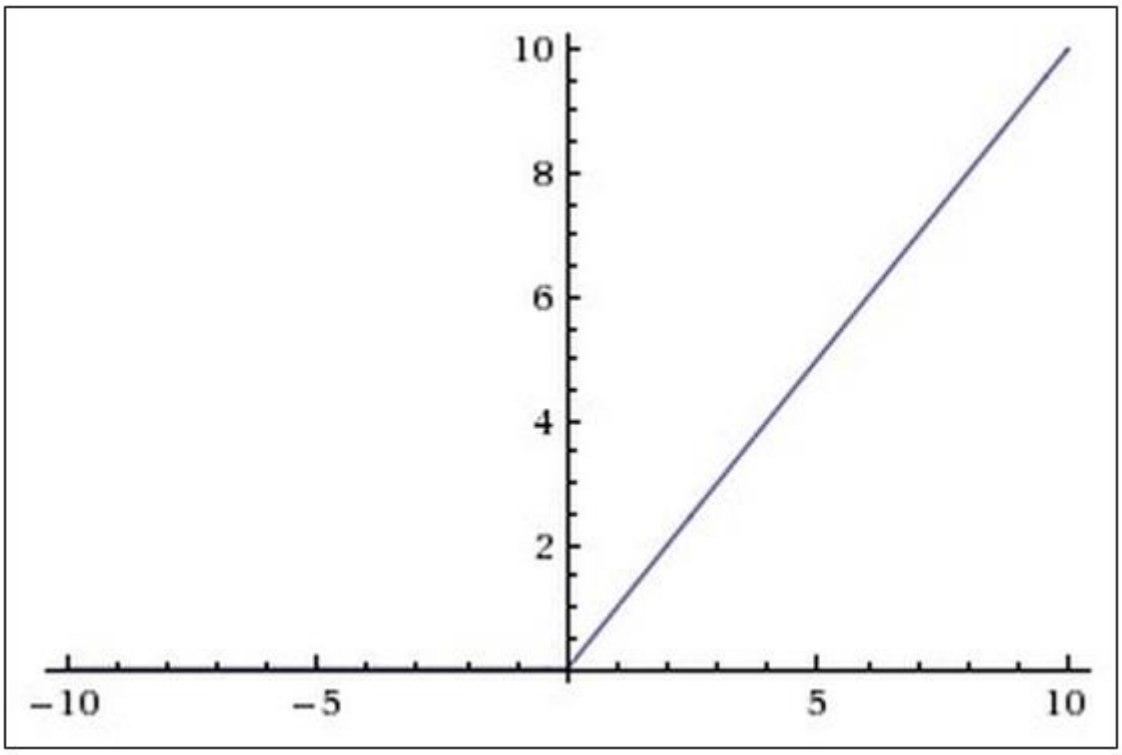

Рисунок 3.8 График функции ReLU

Преимущества использования ReLU:

Во-первых, ее производная равна либо единице, либо нулю, и поэтому не может произойти разрастания или затухания градиентов, т.к. умножив единицу на дельту ошибки мы получим дельту ошибки, если же мы бы использовали другую функцию, например, гиперболический тангенс, то дельта ошибки могла, либо уменьшиться, либо возрасти, либо остаться такой же, то есть, производная гиперболического тангенса возвращает число с разным знаком и величиной, что можно сильно повлиять на затухание или разрастание градиента. Более того, использование данной функции приводит к прореживанию весов;

Во-вторых, вычисление сигмоиды и гиперболического тангенса требует выполнения ресурсоемких операций, таких как возведение в степень, в то время как ReLU может быть реализован с помощью простого порогового преобразования матрицы активаций в нуле;

В-третьих, отсекает ненужные детали в канале при отрицательном выходе.

Из недостатков можно отметить, что ReLU не всегда достаточно надежна и в процессе обучения может выходить из строя («умирать»). Например, большой градиент, проходящий через ReLU, может привести к такому обновлению весов, что данный нейрон никогда больше не активируется. Если это произойдет, то, начиная с данного момента, градиент, проходящий через этот нейрон, всегда будет равен нулю. Соответственно, данный нейрон будет необратимо выведен из строя. Например, при слишком большой скорости обучения (learning rate), может оказаться, что до 40% ReLU «мертвы» (то есть, никогда не активируются). Эта проблема решается посредством выбора надлежащей скорости обучения.

3.1.2.5 Модификации ReLU

Leaky ReLU:

ReLU с «утечкой» (leaky ReLU, LReLU) представляет собой одну из попыток решить описанную выше проблему выхода из строя обычных ReLU. Обычный ReLU на интервале $x < 0$ дает на выходе ноль, в то время как LReLU имеет на этом интервале небольшое отрицательное значение (угловой коэффициент около 0,01). То есть функция для LReLU имеет вид $f(x) = \alpha x$ при $x < 0$ и $f(x) = x$ при $x \geq 0$, где α – малая константа. Некоторые исследователи сообщают об успешном применении данной функции активации, но результаты не всегда стабильны.

Parametric ReLU:

Для параметрического ReLU (parametric ReLU, PReLU] угловой коэффициент на отрицательном интервале не задается предварительно, а определяется на основе данных. Авторы публикации утверждают, что применение данной функции активации является ключевым фактором, позволившим превзойти уровень человека в задаче распознавания изображений ImageNet. Процесс обратного распространения ошибки и обновления для PReLU достаточно прост и подобен соответствующему процессу для традиционных ReLU.

Randomized ReLU:

Для рандомизированного ReLU (randomized ReLU, RReLU) угловой коэффициент на отрицательном интервале во время обучения генерируется случайным образом из заданного интервала, а во время тестирования остается постоянным.

В работе [https://arxiv.org/abs/1505.00853] авторы сравнили точность классификации двух сверточных сетей с различными функциями активации на наборах данных CIFAR-10, CIFAR-100 и NDSB. Результаты приведены в рисунках ниже.

Activation	Training Error	Test Error
ReLU	0.00318	0.1245
Leaky ReLU, $a = 100$	0.0031	0.1266
Leaky ReLU, $a = 5.5$	0.00362	**0.1120**
PReLU	0.00178	0.1179
RReLU	0.00550	**0.1119**

Table 3. Error rate of CIFAR-10 Network in Network with different activation function

Activation	Training Error	Test Error
ReLU	0.1356	0.429
Leaky ReLU, $a = 100$	0.11552	0.4205
Leaky ReLU, $a = 5.5$	0.08536	**0.4042**
PReLU	0.0633	0.4163
RReLU	0.1141	**0.4025**

Table 4. Error rate of CIFAR-100 Network in Network with different activation function

Activation	Train Log-Loss	Val Log-Loss
ReLU	0.8092	0.7727
Leaky ReLU, $a = 100$	0.7846	0.7601
Leaky ReLU, $a = 5.5$	0.7831	0.7391
PReLU	0.7187	0.7454
RReLU	0.8090	**0.7292**

Table 5. Multi-classes Log-Loss of NDSB Network with different activation function

Рисунок 3.9 Сравнение модификаций ReLU

Результаты говорят о том, что для всех трех наборов данных модифицированные ReLU превзошли традиционные. В случае LReLU большее значение углового коэффициента α обеспечивает более высокую точность. PReLU склонны к переобучению на малых наборах данных (ошибка на обучающем наборе наименьшая из всех, в то время как ошибка на тестовом наборе больше, чем у конкурирующих модификаций ReLU). При этом PReLU все же превосходит традиционный ReLU. Следует отметить, что RReLU существенно превосходит другие функции активации на наборе данных NDSB. Это говорит о том, что RReLU позволяет избежать переобучения, поскольку этот набор содержит меньше обучающих данных, чем еньше обучающих данных, чем набор CIFAR-10 и CIFAR-100.

3.1.2.6 Softplus

Еще одной разновидностью функции активации является функция softplus ($f(x) = \ln(1+e^x)$)

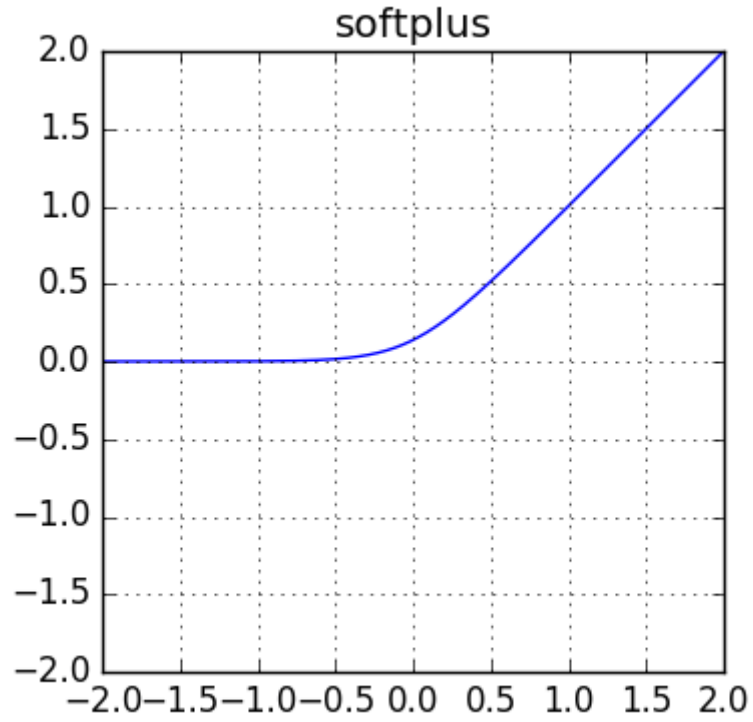

Рисунок 3.10 Функция softplus

В отличие от сигмоидальной функции и гиперболического тангенса, функция не ограничена сверху: область ее значений составляет (0, +∞). Данная функция формой напоминает ReLU, но, в отличие от него, является дифференцируемой в точке 0, что упрощает построение математических моделей нейронных сетей.

3.1.3 Нейронные сети

3.1.3.1 Однослойные нейронные сети

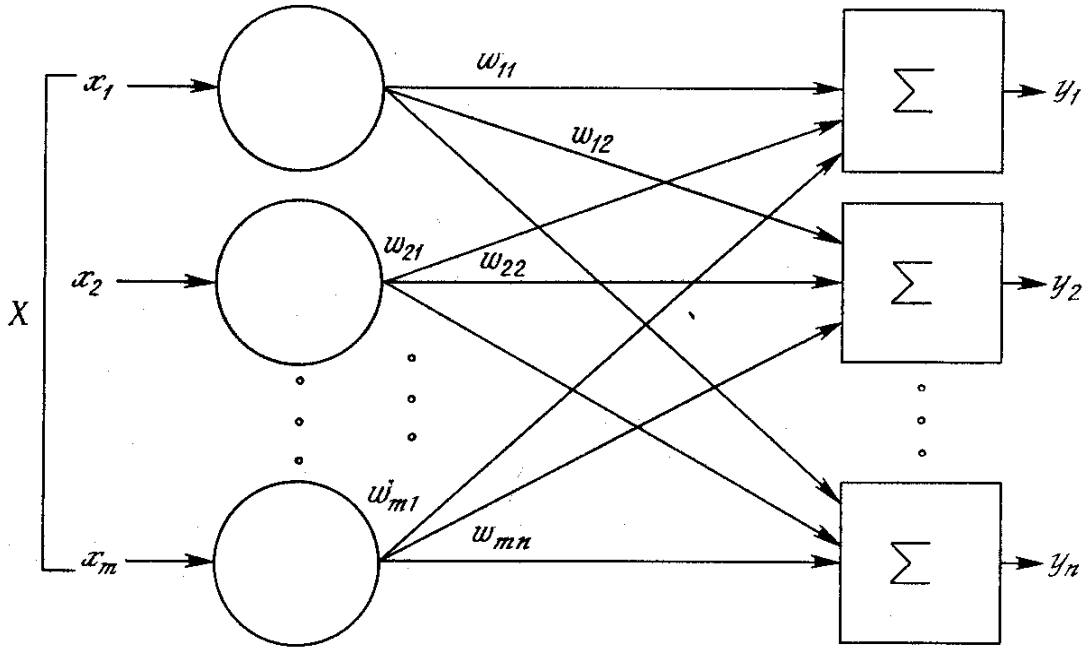

Рисунок 3.11 Однослойная нейронная сеть

Хотя один нейрон и способен выполнять простейшие процедуры распознавания, сила нейронных вычислений проистекает от соединений нейронов в сетях. Простейшая сеть состоит из группы нейронов, образующих слой, как показано в правой части рис. 3.11. Вершины-круги слева служат лишь для распределения входных сигналов. Они не выполняют каких- либо вычислений, и поэтому не будут считаться слоем. По этой причине они обозначены кругами, чтобы отличать их от вычисляющих нейронов, обозначенных квадратами. Каждый элемент из множества входов **X** отдельным весом соединен с каждым искусственным нейроном. А каждый нейрон выдает взвешенную сумму входов в сеть. В искусственных и биологических сетях многие соединения могут отсутствовать, все соединения показаны в целях общности. Могут иметь место также соединения между выходами и входами элементов в слое.

Удобно считать веса элементами матрицы **W**. Матрица имеет *m* строк и *n* столбцов, где m – число входов, а n – число нейронов. Например, $w_{2,3}$ – это вес, связывающий третий вход со вторым нейроном. Таким образом, вычисление выходного вектора **N**, компонентами которого являются выходы OUT

нейронов, сводится к матричному умножению **N = XW,** где **N** и **X** – векторы-строки.

3.1.3.2 Многослойные нейронные сети

Более крупные и сложные нейронные сети обладают, как правило, и большими вычислительными возможностями. Хотя созданы сети всех конфигураций, какие только можно себе представить, послойная организация нейронов копирует слоистые структуры определенных отделов мозга. Оказалось, что такие многослойные сети обладают большими возможностями, чем однослойные, и в последние годы были разработаны алгоритмы для их обучения.

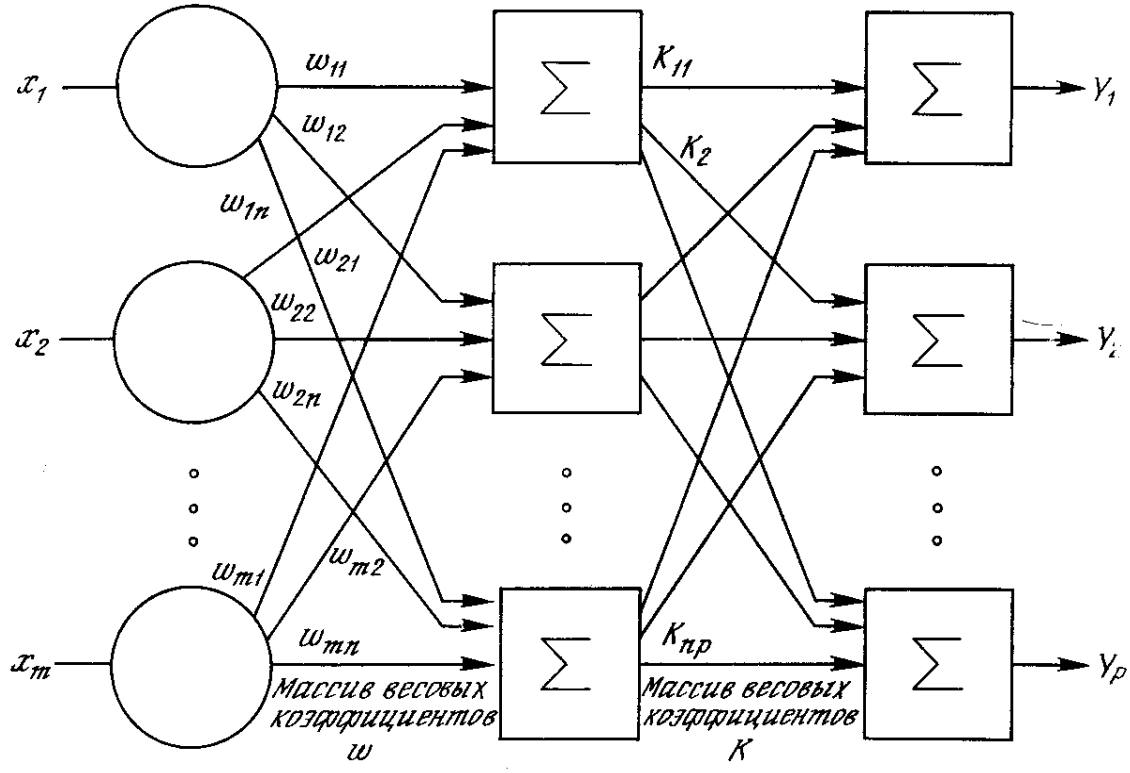

Рисунок 3.12 Двухслойная нейронная сеть

Многослойные сети могут образовываться каскадами слоев. Выход одного слоя является входом для последующего слоя. Подобная сеть показана на рис. 3.12 и снова изображена со всеми соединениями.

3.1.4 Обучение нейронных сетей

3.1.4.1 Цель обучения

Сеть обучается, чтобы для некоторого множества входов давать желаемое (или, по крайней мере, сообразное с ним) множество выходов. Каждое такое входное (или выходное) множество рассматривается как вектор. Обучение осуществляется путем последовательного предъявления входных векторов с одновременной подстройкой весов в соответствии с определенной процедурой. В процессе обучения веса сети постепенно становятся такими, чтобы каждый входной вектор вырабатывал выходной вектор.

3.1.4.2 Обучение с учителем

Различают алгоритмы обучения с учителем и без учителя. Обучение с учителем предполагает, что для каждого входного вектора существует целевой вектор, представляющий собой требуемый выход. Вместе они называются обучающей парой. Обычно сеть обучается на некотором числе таких обучающих пар. Предъявляется выходной вектор, вычисляется выход сети и сравнивается с соответствующим целевым вектором, разность (ошибка) с помощью обратной связи подается в сеть и веса изменяются в соответствии с алгоритмом, стремящимся минимизировать ошибку. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня.

3.1.4.3 Обучение без учителя

Несмотря на многочисленные прикладные достижения, обучение с учителем критиковалось за свою биологическую неправдоподобность. Трудно вообразить обучающий механизм в мозге, который бы сравнивал желаемые и действительные значения выходов, выполняя коррекцию с помощью обратной связи. Если допустить подобный механизм в мозге, то откуда тогда возникают желаемые выходы? Обучение без учителя является намного более

правдоподобной моделью обучения в биологической системе. Развитая Кохоненом и многими другими, она не нуждается в целевом векторе для выходов и, следовательно, не требует сравнения с предопределенными идеальными ответами. Обучающее множество состоит лишь из входных векторов. Обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные векторы, т. е. чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Процесс обучения, следовательно, выделяет статистические свойства обучающего множества и группирует сходные векторы в классы. Предъявление на вход вектора из данного класса даст определенный выходной вектор, но до обучения невозможно предсказать, какой выход будет производиться данным классом входных векторов. Следовательно, выходы подобной сети должны трансформироваться в некоторую понятную форму, обусловленную процессом обучения. Это не является серьезной проблемой. Обычно не сложно идентифицировать связь между входом и выходом, установленную сетью.

3.1.4.4 Алгоритмы обучения

Большинство современных алгоритмов обучения выросло из концепций Хэбба [2]. Им предложена модель обучения без учителя, в которой синаптическая сила (вес) возрастает, если активированны оба нейрона, источник и приемник. Таким образом, часто используемые пути в сети усиливаются и феномен привычки и обучения через повторение получает объяснение.

В искусственной нейронной сети, использующей обучение по Хэббу, наращивание весов определяется произведением уровней возбуждения передающего и принимающего нейронов. Это можно записать как

$$w_{ij}(n+1) = w(n) + \alpha OUT_i \, OUT_j,$$

где $w_{ij}(n)$ – значение веса от нейрона *i* к нейрону *j* до подстройки, $w_{ij}(n+1)$ – значение веса от нейрона *i* к нейрону *j* после подстройки, α –

коэффициент скорости обучения, OUT_i – выход нейрона *i* и вход нейрона *j*, OUT_j – выход нейрона *j*.

Сети, использующие обучение по Хэббу, конструктивно развивались, однако за последние 20 лет были развиты более эффективные алгоритмы обучения. В частности, были развиты алгоритмы обучения с учителем, приводящие к сетям с более широким диапазоном характеристик обучающих входных образов и большими скоростями обучения, чем использующие простое обучение по Хэббу.

3.2 Программа для сравнения архитектур и параметров НС

Все вышеперечисленные алгоритмы и инструменты имеют разную применимость при решение конкретных задач. Помимо этого, они могут использоваться в различных комбинациях друг с другом. Для оптимизации нейронной сети была разработана вспомогательная программа, позволяющая перебирать различные параметры комбинации параметров нейронных сетей.

3.2.1 Параметры и их значения

1) Максимальная высота фильтров
    a) 3
    b) 4
    c) 5
2) Количество сверточных слоев
    a) 3
    b) 6
    c) 10
3) Высота плотного слоя
    a) 15
    b) 30

c) 60

4) Функция активации
   a) Tanh (гиперболический тангенс)
   b) Sigmoid
   c) Hard_sigmoid (упрощенная сигмоида)
   d) ReLU (rectified linear unit)
5) Dropout-регуляризация
   a) 0 (без регуляризации)
   b) 0.1
   c) 0.2

3.2.2 Принцип работы

Программа перебирает все возможные комбинации параметров. Для каждой комбинации строится соответствующая нейронная сеть и обучается. После этого сеть оценивается с помощью метрик, описанных в предыдущей главе (точность и F1-мера).

После этого собирается таблица результатов (приведена ниже) и строятся графики сравнения. Для каждого значения параметра (максимальная высота фильтров, количество сверточных слоев, высота плотного слоя, функция активации и регуляризация) строится 4 графика:

1. Среднее значение точности классификации в комбинациях со всеми остальными значениями параметров

2. Максимальная точность классификации (то есть точность классификации в наилучшей комбинации этого значения параметра с остальными)

3 и 4. Аналогичные графики для F1-меры

Так как полное обучение одной нейронной сети занимает значительное время (несколько часов), в программе каждая нейронная сеть проходила всего 3 эпохи обучения (вместо 10, используемых в основной программе). При этом использовалось допущение, что если нейронные сети А и Б прошли три эпохи

обучения и А показала лучшие результаты, чем Б, то на 10 эпохах обучения А также покажет лучшие результаты.

Допущение было выборочно проверено на полученных результатах. Были выбраны случайным образом 5 пар комбинаций параметров ([(А1, Б1), (А2, Б2), …, (А5, Б5)]). Для каждой из пар был проанализирован результат сравнения точности и F1-меры на 3 эпохах обучения и на 10. Во всех случаях результаты сравнения совпали (то есть нейронная сеть, превосходящая другую на 3 этапах обучения, превосходила ее и на 10 этапах).

3.3 Результаты сравнения

4 Реализация

Было реализовано приложение на языке python с использованием библиотеки Keras для построения нейронной сети. Код программы с комментариями вынесен в приложение 1.

Заключение

В рамках работы было создано приложение, способное классифицировать тексты с точностью, превосходящей традиционные подходы.

Для этого были изучены существующие модели для всех этапов классификации текстов, а также произведено их сравнительное тестирование с целью оценки их качества относительно разрабатываемого приложения.

Таким образом, сверточные нейронные сети являются подходящим инструментом для задач классификации текстов, в частности, анализа тональности коротких текстов.

5 Список использованной литературы

1. Батура Т.В. Методы автоматической классификации текстов // Программные продукты и системы. 2017.
2. Лекция № 6 по классификации текстов курса «Современные задачи теоретической информатики» (постановка задачи, построение и обучение классификатора, оценка качества).
3. F. Sebastiani. Machine Learning in Automated Text Categorization
4. "Text mining. Классификация текста". Пример классификации документов с использованием программных алгоритмов STATISTICA
5. Aggarwal C. Data Classification: Algorithms and Applications. CRC Press, 2014, pp. 245–273
6. Medhat W., Hassan A., Korashy H. Sentiment analysis algorithms and applications: A survey. Ain Shams Engineering Journ. 2014, no. 5, pp. 1093–1113.
7. Воронцов, К. В. Курс лекций по машиному обучению / К. В. Воронцов. — 2015