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

Оглавление

[Введение 4](#_Toc9687737)

[1 Теория классификации текстов 6](#_Toc9687738)

[1.1 Основные понятия и определения 6](#_Toc9687739)

[1.1.1 Постановка задачи классификации текстов 6](#_Toc9687740)

[1.1.2 Определение анализа тональности 6](#_Toc9687741)

[1.2 Методы решения 8](#_Toc9687742)

[1.2.1 Подход на правилах (Rule based classification) 8](#_Toc9687743)

[1.2.2 Подход на словарях (Weight based classification) 8](#_Toc9687744)

[1.2.3 Машинное обучение (Machine Learning) 9](#_Toc9687745)

[1.3 Этапы классификации текстов 10](#_Toc9687746)

[1.4 Индексация 11](#_Toc9687747)

[1.4.1 N-граммы 11](#_Toc9687748)

[1.4.2 Bag of words 12](#_Toc9687749)

[1.4.3 Word2vec 12](#_Toc9687750)

[1.5 Выбор признаков 14](#_Toc9687751)

[1.5.1 TF-IDF 14](#_Toc9687752)

[1.5.2 Дельта TF-IDF 14](#_Toc9687753)

[1.6 Классификаторы 16](#_Toc9687754)

[Метод Байеса (Naive Bayes, NB) 16](#_Toc9687755)

[Метод k ближайших соседей (k Nearest Neighbors, KNN) 17](#_Toc9687756)

[Метод деревьев решений (Decision Trees, DT) 18](#_Toc9687757)

[Метод опорных векторов (Support Vector Machine, SVM) 20](#_Toc9687758)

[1.6.1 Логистическая регрессия (logit model, logistic regression) 23](#_Toc9687759)

[1.6.2 Методы на основе искусственных нейронных сетей. 24](#_Toc9687760)

[2 Сравнение алгоритмов 28](#_Toc9687761)

[2.1 Методики оценки 28](#_Toc9687762)

[2.1.1 Матрица ошибок 28](#_Toc9687763)

[2.1.2 Доля правильных ответов (accuracy) 28](#_Toc9687764)

[2.1.3 Precision, recall и F-мера 37](#_Toc9687765)

[3 Архитектура нейронной сети 43](#_Toc9687766)

[3.1 Основные определения 43](#_Toc9687767)

[3.1.1 Нейрон 43](#_Toc9687768)

[3.1.2 Функции активации 46](#_Toc9687769)

[3.2 Программа для сравнения архитектур и параметров НС 59](#_Toc9687770)

[3.2.1 Параметры и их значения 59](#_Toc9687771)

[3.2.2 Принцип работы 60](#_Toc9687772)

[3.3 Результаты сравнения 61](#_Toc9687773)

[4 Реализация 62](#_Toc9687774)

[Заключение 63](#_Toc9687775)

[5 Список использованной литературы 64](#_Toc9687776)

# 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.

# Теория классификации текстов

## Основные понятия и определения

### 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 = {d1, ..., dn}, множество категорий (классов, меток) C = {c1, ..., cn|} и неизвестная целевая функция Φ: C × D → {0, 1}. Необходимо построить классификатор Φ’, максимально близкий к Φ.

Имеется некоторая начальная коллекция размеченных документов , для которых известны значения Φ. 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 или степень подобия .

### 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 ХХХ 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

## Методы решения

### 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.

### 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.

### 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.

## 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.

## 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.

### 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.

### 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).

### 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.

## Выбор признаков

### 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 = nt,d / nd, где nt,d – количество употреблений слова t в документе d; nd – 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| / Dt), где |D| – общее количество документов в коллекции; Dt – количество всех документов, в которых встречается слово t. Итоговый вес термина в документе относительно всей коллекции документов вычисляется по формуле Vt, d = TF × 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.

### 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:

Vt,d=Ct,dlog(|N|Pt|P|Nt)

где:

Vt,d — вес слова t в документе d

Сt,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

Nt — 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 (Pt) отзывов и почти не встречается в отрицательных (Nt), в итоге вес будет большим положительным числом, т.к. отношение Pt/Nt будет числом гораздо больше 1.

Слово «нудный» наоборот встречается в основном в отрицательных отзывах, поэтому отношение Pt/Nt 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 Pt/Nt 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.

## 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 вычисляется расстояние . 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:

http://www.swsys.ru/uploaded/image/2017-1/image148.gifгде Lk (d) – ближайшие k документов из L к d; F(dz , ci ) – 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.

Дерево решений как алгоритм машинного обучения – по сути объединение логических правил вида "Значение признака a меньше x И Значение признака b меньше y … => Класс 1" в структуру данных "Дерево". Значительное преимущество деревьев решений в том, что они легко интерпретируемы, понятны человеку.

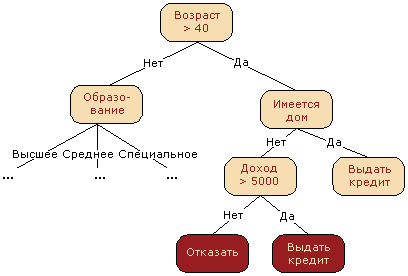
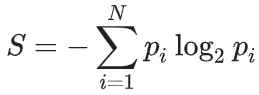


Рисунок 1.1 Пример дерева решений

Например, по схеме на рисунке выше можно объяснить заемщику, почему ему было отказано в кредите. Например, потому, что у него нет дома и доход меньше 5000. Как мы увидим далее, многие другие, хоть и более точные, модели не обладают этим свойством и могут рассматриваться скорее как "черный ящик", в который загрузили данные и получили ответ. В связи с этой "понятностью" деревьев решений и их сходством с моделью принятия решений человеком, деревья решений получили огромную популярность, а один из представителей этой группы методов классификации, С4.5, рассматривается первым в списке 10 лучших алгоритмов интеллектуального анализа данных ["Top 10 algorithms in data mining", Knowledge and Information Systems, 2008. PDF)].

#### Энтропия

Энтропия Шеннона определяется для системы с N возможными состояниями следующим образом:



Где pi – вероятности нахождения системы в i-ом состоянии. Это очень важное понятие, используемое в физике, теории информации и других областях. Опуская предпосылки введения (комбинаторные и теоретико-информационные) этого понятия, отметим, что, интуитивно, энтропия соответствует степени хаоса в системе. Чем выше энтропия, тем менее упорядочена система и наоборот. Это поможет нам формализовать "эффективное разделение выборки".

#### Другие критерии качества разбиения

Также существуют другие критерии качества разбиения.

**Неопределенность Джини** (Gini impurity):

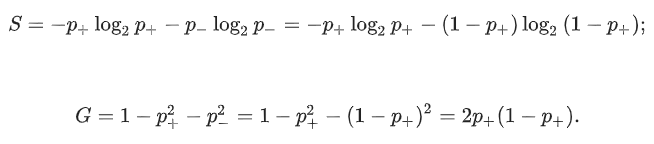
Максимизацию этого критерия можно интерпретировать как максимизацию числа пар объектов одного класса, оказавшихся в одном поддереве.

**Ошибка классификации** (misclassification error):



На практике ошибка классификации почти не используется, а неопределенность Джини и прирост информации работают почти одинаково.

В случае задачи бинарной классификации (p+ – вероятность объекта иметь метку +) энтропия и неопределенность Джини примут следующий вид:



Если построить графики этих двух функций от аргумента p+, то увидим, что график энтропии очень близок к графику удвоенной неопределенности Джини, и поэтому на практике эти два критерия "работают" почти одинаково.

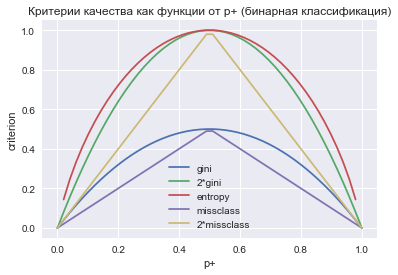


Рисунок 1.2 Критерии качества как функции от p+

#### Алгоритм построения дерева

В основе популярных алгоритмов построения дерева решений, таких как ID3 и C4.5, лежит принцип жадной максимизации прироста информации – на каждом шаге выбирается тот признак, при разделении по которому прирост информации оказывается наибольшим. Дальше процедура повторяется рекурсивно, пока энтропия не окажется равной нулю или какой-то малой величине (если дерево не подгоняется идеально под обучающую выборку во избежание переобучения).

В разных алгоритмах применяются разные эвристики для "ранней остановки" или "отсечения", чтобы избежать построения переобученного дерева.

Преимущества метода:

* быстрый процесс обучения;
* генерация правил в областях, где эксперту трудно формализовать свои знания;
* извлечение правил на естественном языке;
* интуитивно понятная классификационная модель;
* высокая точность прогноза, сопоставимая с другими методами (статистика, нейронные сети);
* построение непараметрических моделей

Недостатки метода:

* неустойчивость алгоритма по отношению к выбросам в исходных данных
* большой объем данных для получения точных результатов.

Метод опорных векторов (Support Vector Machine, SVM)

Метод опорных векторов (support vector), называемый ранее алгоритмом “обобщенного портрета”, был разработан советскими математиками В. Н. Вапником и А. Я. Червоненкисом (1974) и с тех пор приобрел широкую популярность.

Основная идея классификатора на опорных векторах заключается в том, чтобы строить разделяющую поверхность с использованием только небольшого подмножества точек, лежащих в зоне, критической для разделения, тогда как остальные верно классифицируемые наблюдения обучающей выборки вне этой зоны игнорируются (точнее, являются “резервуаром” для оптимизационного алгоритма).

Если имеется два класса наблюдений и предполагается линейная форма границы между классами, то возможны два случая. Первый из них связан с возможностью идеального разделения данных при помощи некоторой гиперплоскости  (двумерный вариант представлен слева на рис. 1.2). Поскольку таких гиперплоскостей может быть множество, то оптимальной является разделяющая поверхность, которая максимально удалена от обучающих точек, т.е. имеющая максимальный зазор M (margin).

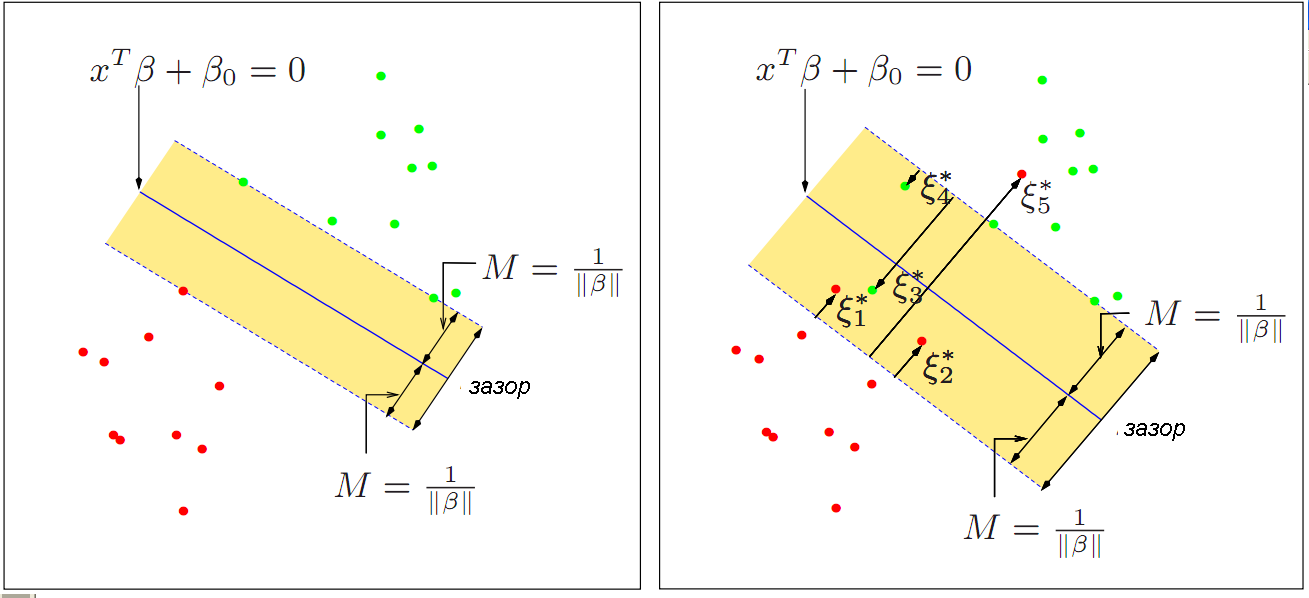


Рисунок 1.3 Классификаторы с минимальным зазором (слева) и на опорных векторах (справа)

На рис. 1.2 справа показан другой случай, когда облако точек перекрывается и оба класса линейно неразделимы. Собственно опорными векторами называются наблюдения, лежащие непосредственно на границе разделяющей полосы, либо на неправильной для своего класса стороне относительно границ зазора. Для граничных и всех остальных точек принимается ξ∗J=0 .

Оптимальную разделяющую гиперплоскость такого классификатора  также находят из условия максимизации ширины зазора M , но при этом разрешается неверно классифицировать некоторую небольшую группу наблюдений, относящихся к опорным векторам. Для этого задаются дополнительным условием оптимизации  , где C - допустимое число нарушений границы зазора и их выраженность, которое обычно выбирается с использованием перекрестной проверки. Математически поиск решения сводится к удобной задаче квадратичной оптимизации с линейными ограничениями, которая гарантировано сходится к одному глобальному минимуму (Vapnik, 1995; Джеймс и др., 2016).

Поскольку на расположение гиперплоскости оказывают влияние только те наблюдения, которые лежат на границах зазора или нарушают его, то решающее правило такого классификатора довольно устойчиво к выбросам большинства точек, расположенных вне “критической зоны” разделения. Это свойство отличает его от свойств других классификаторов. Например, разделяющая плоскость LDA на рис. 6.2 проводится перпендикулярно дискриминационной функции z и зависит от средних и ковариационной матрицы, вычисляемых по всем имеющимся наблюдениям.

### 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 x1, …, xn, на основе значений которых требуется вычислить вероятность принятия того или иного значения зависимой переменной. В случае классификации документов роль зависимой переменной выполняет категория ci, а роль независимых переменных – набор документов d1, …, dn.

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: http://www.swsys.ru/uploaded/image/2017-1/image150.gif

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.

### 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 di 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.

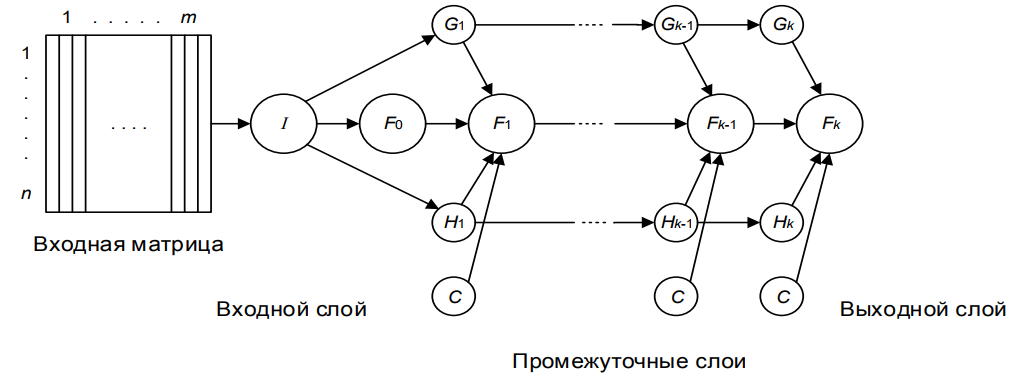


Рисунок 1.2 Структурная схема нейронной сети

Обобщенная схема DAN2 приведена на рисунке …. Каждый элемент Fk представляет собой функцию, которая содержит текущий элемент накопленных знаний (Current Accu­mulated Knowledge Element), полученный на предыдущем шаге обучения сети. C обозначают константы. Вершины Gk и Hk 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.

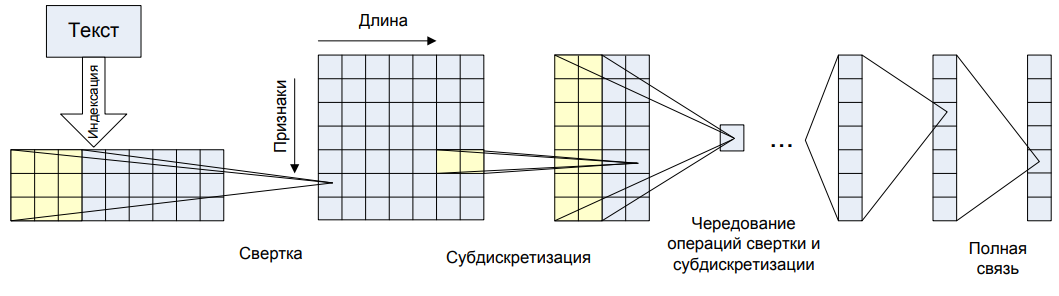


Рисунок 1.3. Сверточная нейронная сеть

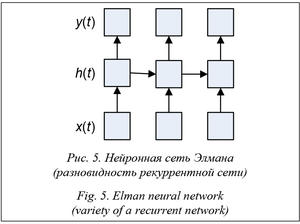
[](http://www.swsys.ru/uploaded/image/2017_1/42.jpg)Рекуррентная нейронная сеть получается из многослойного перцептрона введением обратных связей. Одна из широко распространенных разновидностей рекуррентных нейронных сетей – сеть Элмана – изображена на рисунке.

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.

# Сравнение алгоритмов

## 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.

### Матрица ошибок

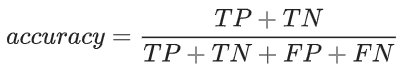
Матрица ошибок для задачи классификации на два класса представляет собой таблицу 2х2:

|  |  |  |
| --- | --- | --- |
|  | 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:

### Доля правильных ответов (accuracy)

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



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-граммы (отдельные слова)
   2. 1- и 2- граммы (отдельные слова и словосочетания длиной 2)
   3. 2-граммы (словосочетания длиной 2)
2. Трансформация
   1. Линейный TF
   2. Логарифмический TF
   3. Линейный TF-IDF
   4. Логарифмический TF-IDF
3. Классификация
   1. Метод K ближайших соседей
   2. Метод опорных векторов с линейным ядром
   3. Метод опорных векторов с ядром RBF (радиальная базисная функция)
   4. Дерево решений
   5. Метод «случайного леса»
   6. AdaBoost
   7. Многомерный наивный Байесовский классификатор
   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:

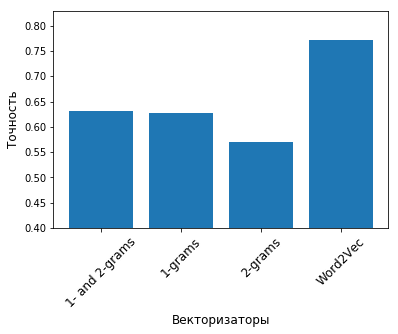


Рисунок 2.1 Средняя доля правильных ответов для векторизаторов

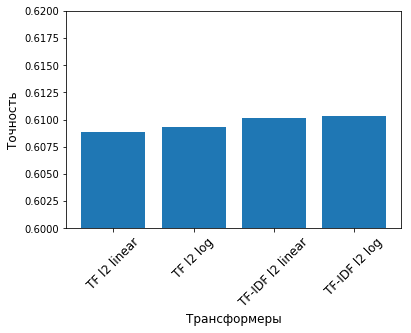


Рисунок 2.2 Средняя доля правильных ответов для трансформеров

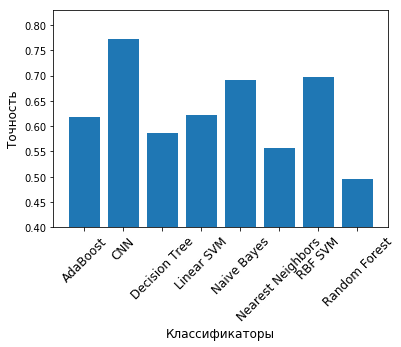


Рисунок 2.3 Средняя доля правильных ответов для классификаторов

Лучшие:

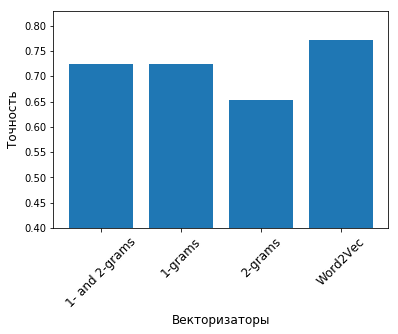


Рисунок 2.4 Наилучшая доля правильных ответов для векторизаторов

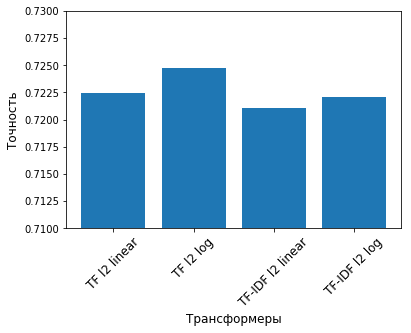


Рисунок 2.5Наилучшая доля правильных ответов для трансформеров

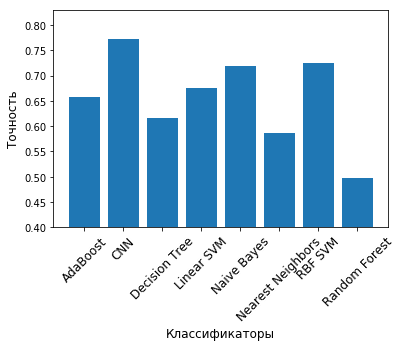
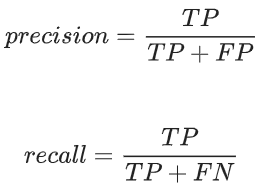


Рисунок 2.6 Наилучшая доля правильных ответов для классификаторов

### Precision, recall и F-мера

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



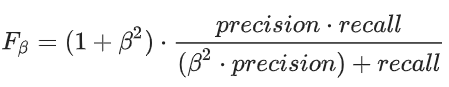
Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

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 ) — среднее гармоническое precision и recall :



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

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.

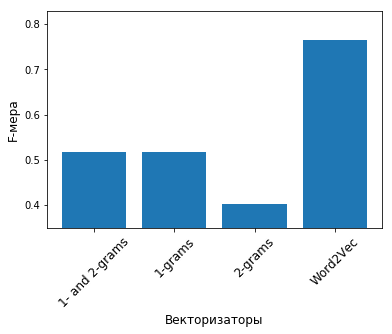


Рисунок 2.7 Средняя F-мера для векторизаторов

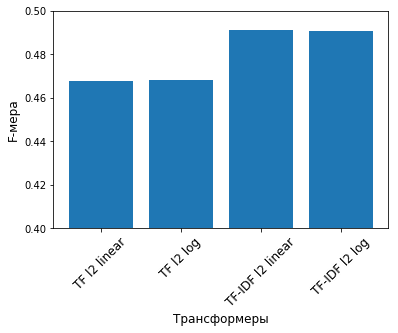


Рисунок 2.8 Средняя F-мера для трансформеров

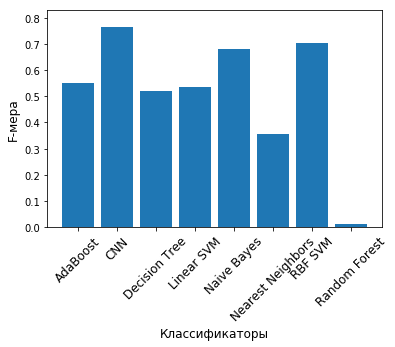


Рисунок 2.9 Средняя F-мера для классификаторов

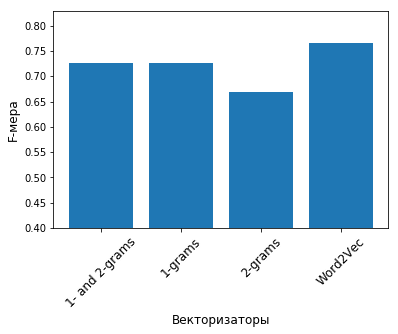


Рисунок 2.10 Наилучшая F-мера для векторизаторов

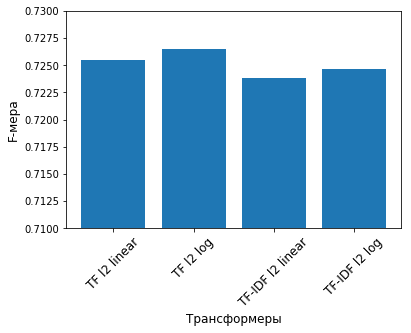


Рисунок 2.11 Наилучшая F-мера для трансформеров

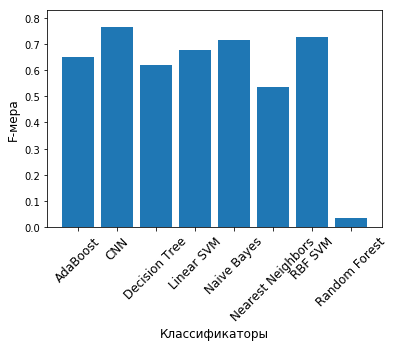


Рисунок 2.12 Наилучшая F-мера для векторизаторов

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

# Архитектура нейронной сети

## Основные определения

### Нейронная сеть

Как известно, понятие нейронной сети (НС) пришло из биологии и представляет собой несколько упрощенную модель строения человеческого мозга. Проще всего представить нейрон (в том числе, искусственный) как некий черный ящик с множеством входных отверстий и одним выходным.

Математически, искусственный нейрон осуществляет преобразование вектора входных сигналов (воздействий) X в вектор выходных сигналов Y при помощи функции, называемой функцией активации. В рамках соединения (искусственной нейронной сети — ИНС) функционируют три вида нейронов: входные (принимающие информацию из внешнего мира – значения интересующих нас переменных), выходные (возвращающие искомые переменные – к примеру, прогнозы, или управляющие сигналы), а также промежуточные – нейроны, выполняющие некие внутренние («скрытые») функции. Классическая ИНС, таким образом, состоит из трех или более слоев нейронов, причем на втором и последующих слоях («скрытых» и выходном) каждый из элементов соединен со всеми элементами предыдущего слоя.

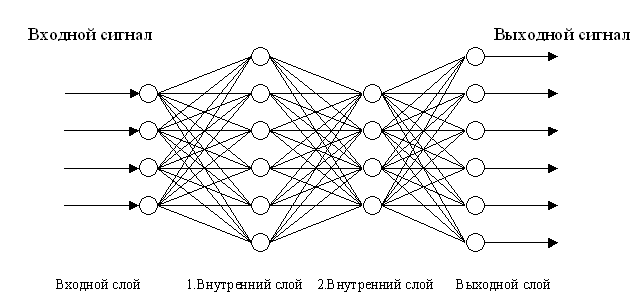


Рисунок 3.1 Пример нейронной сети

Важно помнить о понятии обратной связи, которое определяет вид структуры ИНС: прямой передачи сигнала (сигналы идут последовательно от входного слоя через скрытый и поступают в выходной слой) и рекуррентной структуры, когда сеть содержит связи, идущие назад, от более дальних к более ближним нейронам). Все эти понятия составляют необходимый минимум информации для перехода на следующий уровень понимания ИНС – обучения нейронной сети, классификации его методов и понимания принципов работы каждого из них.

### Нейрон

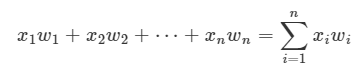


Рисунок 3.2 Схема нейрона

У каждого нейрона, в том числе и у искусственного, должны быть какие-то входы, через которые он принимает сигнал. Мы уже вводили понятие весов, на которые умножаются сигналы, проходящие по связи. На картинке выше веса изображены кружками.

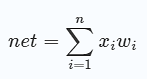
Поступившие на входы сигналы умножаются на свои веса. Сигнал первого входа ​x1​ умножается на соответствующий этому входу вес ​w1​. В итоге получаем ​x1w1​. И так до ​n​-ого входа. В итоге на последнем входе получаем ​xnwn​.

Теперь все произведения передаются в сумматор. Уже исходя из его названия можно понять, что он делает. Он просто суммирует все входные сигналы, умноженные на соответствующие веса:



Результатом работы сумматора является число, называемое взвешенной суммой.

Взвешенная сумма (Weighted sum) — сумма входных сигналов, умноженных на соответствующие им веса:



Роль сумматора очевидна – он агрегирует все входные сигналы (которых может быть много) в какое-то одно число – взвешенную сумму, которая характеризует поступивший на нейрон сигнал в целом. Еще взвешенную сумму можно представить как степень общего возбуждения нейрона.

### Функции активации

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

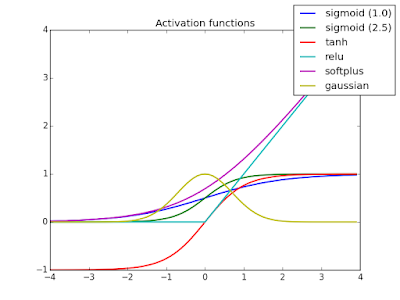


Рисунок 3.3 Графики основных функций активации

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

#### Логистическая (сигмоидальная) функция

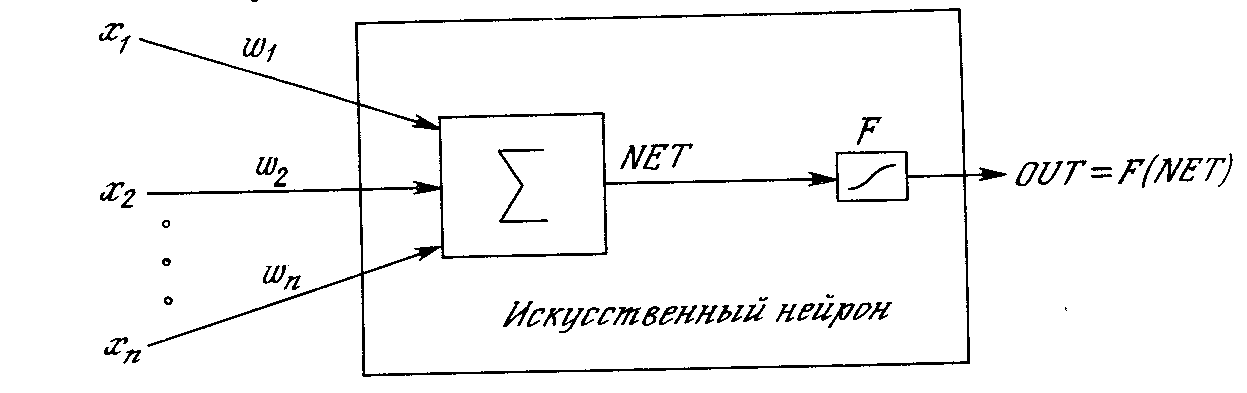


Рисунок 3.4 Искусственный нейрон с активационной функцией

Сигмоида (sigmoid) выражается следующей формулой: σ(x) = 1 / (1 + e-x). Эта функция принимает на входе произвольное вещественное число, а на выходе дает вещественное число в интервале от 0 до 1. В частности, большие (по модулю) отрицательные числа превращаются в ноль, а большие положительные – в единицу. Исторически сигмоида находила широкое применение, поскольку ее выход хорошо интерпретируется, как уровень активации нейрона: от отсутствия активации (0) до полностью насыщенной активации (1).

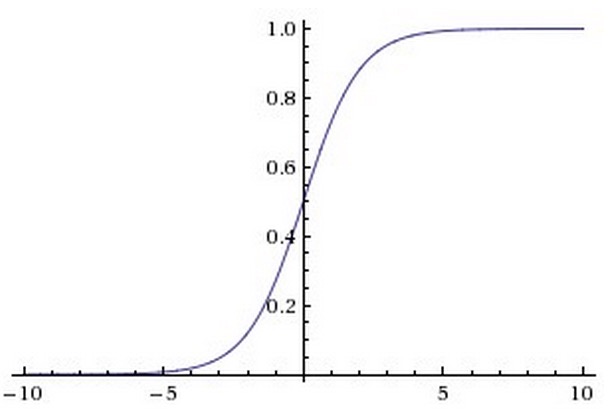


Рисунок 3.5 График сигмоиды

На текущий момент сигмоида утратила свою былую популярность и используется очень редко. Данная функция имеет два серьезных недостатка:

Насыщение сигмоиды приводит к затуханию градиентов. Крайне нежелательное свойство сигмоиды заключается в том, что при насыщении функции с той или иной стороны (0 или 1), градиент на этих участках становится близок к нулю. Напомним, что в процессе обратного распространения ошибки данный (локальный) градиент умножается на общий градиент. Следовательно, если локальный градиент очень мал, он фактически обнуляет общий градиент. В результате, сигнал почти не будет проходить через нейрон к его весам и рекурсивно к его данным. Кроме того, следует быть очень осторожным при инициализации весов сигмоидных нейронов, чтобы предотвратить насыщение. Например, если исходные веса имеют слишком большие значения, большинство нейронов перейдет в состояние насыщения, в результате чего сеть будет плохо обучаться.

Выход сигмоиды не центрирован относительно нуля. Это свойство является нежелательным, поскольку нейроны в последующих слоях будут получать значения, которые не центрированы относительно нуля, что оказывает влияние на динамику градиентного спуска (gradient descent). Если значения, поступающие в нейрон, всегда положительны (например, x > 0 поэлементно в f = ωTx + b), тогда в процессе обратного распространения ошибки все градиенты весов ω будут либо положительны, либо отрицательны (в зависимости от градиента всего выражения f). Это может привести к нежелательной зигзагообразной динамике обновлений весов. Однако следует отметить, что когда эти градиенты суммируются по пакету, итоговое обновление весов может иметь различные знаки, что отчасти нивелирует описанный недостаток. Таким образом, отсутствие центрирования является неудобством, но имеет менее серьезные последствия, по сравнению с проблемой насыщения.

#### 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.

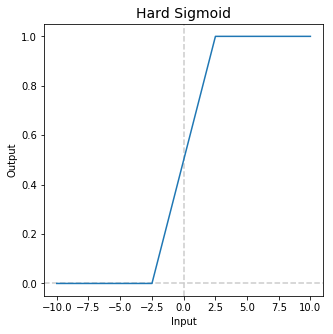


Рисунок 3.6 Приближенно-сигмоидальная фунцкия

#### Гиперболический тангенс

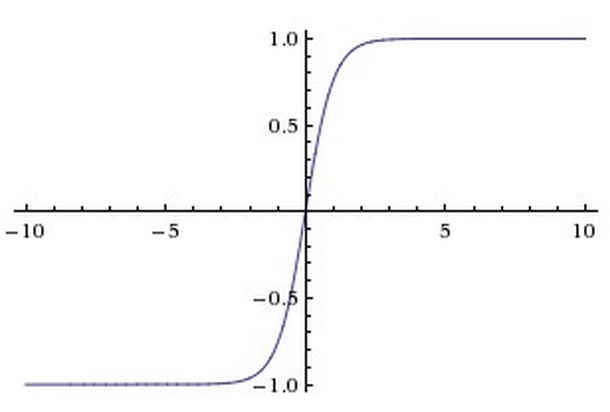
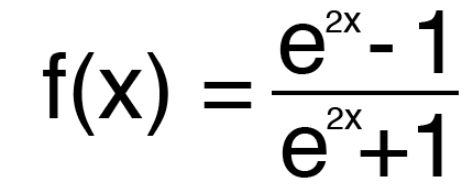


Рисунок 3.7 Гиперболический тангенс

Гиперболический тангенс (hyperbolic tangent, tanh) принимает на входе произвольное вещественное число, а на выходе дает вещественное число в интервале от –1 до 1. Гиперболический тангенс вычисляется следующим образом:



Подобно сигмоиде, гиперболический тангенс может насыщаться. Однако, в отличие от сигмоиды, выход данной функции центрирован относительно нуля.

#### 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:

https://habrastorage.org/webt/tq/by/ht/tqbyhtxpyotkoeqrn6aygpmne2m.png

График функции ReLU в соответствии с рисунком ниже:

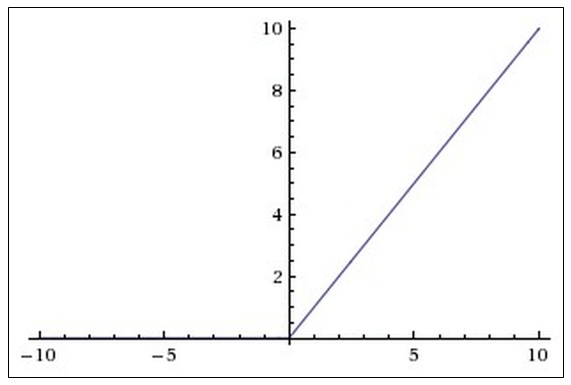


Рисунок 3.8 График функции ReLU

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.

#### 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) = αx при x < 0 и f(x) = x при x ≥ 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.

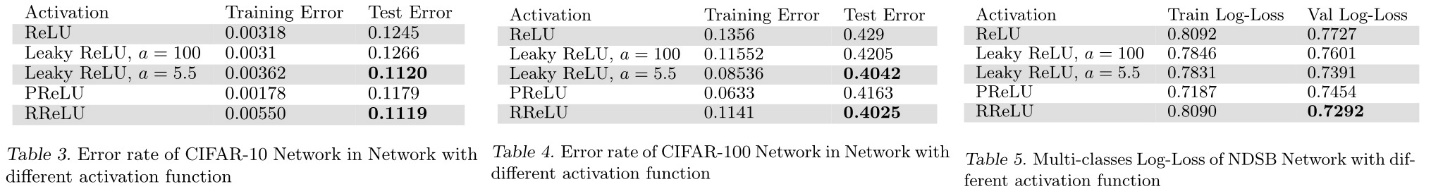


Рисунок 3.9 Сравнение модификаций ReLU

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.

#### Softplus

Еще одной разновидностью функции активации является функция softplus (f(x) = ln(1+ex))

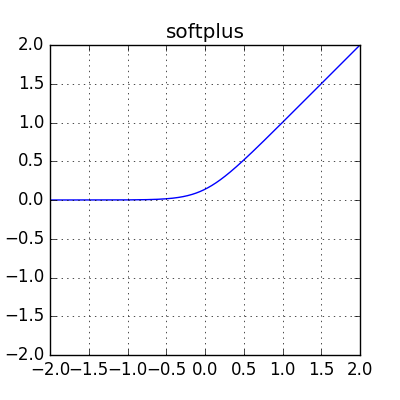


Рисунок 3.10 Функция softplus

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.

### Обучение нейронной сети

Под искусственными нейронными сетями понимают класс методов для решения определенных практических задач, среди которых главными являются задачи распознавания образов, принятия решений, аппроксимации и сжатия данных, а также наиболее интересные для нас задачи кластерного анализа и прогнозирования.

При любых обстоятельствах именно способность нейронной сети к обучению (с учителем или «самостоятельно») и является ключевым моментом использования ее для решения практических задач.

В общем случае, обучение ИНС заключается в следующем:

1. входные нейроны принимают переменные («стимулы») из внешней среды;
2. в соответствии с полученной информацией изменяются свободные параметры НС (работают промежуточные слои нейронов);
3. в результате изменений в структуре НС сеть «реагирует» на информацию уже иным образом.

Очевидно, что универсального алгоритма обучения не существует и, скорее всего, существовать не может; концептуально подходы к обучению делятся на обучение с учителем и обучение без учителя. Первый алгоритм предполагает, что для каждого входного («обучающегося») вектора существует требуемое значение выходного («целевого») вектора – таким образом, два этих значения образуют обучающую пару, а вся совокупность таких пар – обучающее множество. В случае варианта обучения без учителя обучающее множество состоит лишь из входных векторов – и такая ситуация является более правдоподобной с точки зрения реальной жизни.

#### Глубокое обучение

Понятие глубокого обучения (deep learning) относится к другой классификации и обозначает подход к обучению так называемых глубоких структур, к которым можно отнести многоуровневые нейронные сети. Простой пример из области распознавания образов: необходимо научить машину выделять все более абстрактные признаки в терминах других абстрактных признаков, то есть определить зависимость между выражением всего лица, глаз и рта и, в конечном итоге, скопления цветных пикселов математически. Таким образом, в глубокой нейронной сети за каждый уровень признаков отвечает свой слой; понятно, что для обучения такой «махины» необходим соответствующий опыт исследователей и уровень аппаратного обеспечения. Условия сложились в пользу глубокого обучения НС только к 2006 году – и спустя восемь лет можно говорить о революции, которую произвел этот подход в машинном обучении.

Итак, прежде всего стоит заметить следующее: глубокое обучение в большинстве случае не контролируется человеком. То есть этот подход подразумевает обучение нейронной сети без учителя. Это и есть главное преимущество «глубокого» подхода: машинное обучение с учителем, особенно в случае глубоких структур, требует колоссальных временных – и трудовых – затрат. Глубокое же обучение – подход, моделирующий человеческое абстрактное мышление (или, по крайней мере, представляет собой попытку приблизиться к нему), а не использующий его.

### Инициализация нейронных сетей

#### Инициализация нулями

В идеальном случае при правильной нормализации данных логично предположить, что примерно половина весов будет иметь положительные значения, а другая половина – отрицательные. Далее нам может показаться, что рационально будет задать все исходные веса равными нулю. Однако это суждение ошибочно. В этом случае все нейроны вычислят одинаковые выходы, следовательно, далее они также вычислят одинаковые градиенты в процессе обратного распространения ошибки и, соответственно, обновление параметров также будет одинаковым. Другими словами, если веса всех нейронов исходно будут одинаковыми, у нас не будет необходимого источника асимметрии между нейронами.

#### Инициализация малыми случайными значениями

Таким образом, нам нужно, чтобы веса были близки к нулю, но не равны ему. Для этого мы можем инициализировать их малыми случайными значениями очень близкими к нулю, что позволит нарушить симметрию. В результате, все исходные веса будут случайными и уникальными, следовательно, обновляться они будут по-разному, что нам и нужно. Вычислить веса можно следующим образом:

weights ~ 0,001 × N(0, 1)

где N(0, 1) – нормальное распределение с математическим ожиданием, равным 0, и среднеквадратическим отклонением, равным 1. Кроме того, можно использовать малые случайные значения из равномерного распределения, но на практике этот подход не оказывает существенного влияния на результат.

#### Калибровка дисперсии

Проблема описанного выше подхода заключается в том, что дисперсия распределения выхода нейрона, инициализированного случайным образом, возрастает с увеличением количества входов. Мы можем нормализовать дисперсию выхода каждого нейрона, разделив вектор весов на квадратный корень из количества входов:

*>>> w = np.random.randn(n) / sqrt(n)*

где функция randn() генерирует случайные числа из упомянутого выше нормального распределения, а n является количеством входов. Благодаря этому подходу, все нейроны сети исходно имеют приблизительно одинаковое выходное распределение, что позволяет повысить скорость сходимости. Подробный вывод этой формулы можно найти на страницах 18 – 23 слайдов. Обратите внимание, в выводе формулы не учитывается влияние ReLU-нейронов.

#### ReLU - нейроны

Как уже было сказано, предыдущий метод инициализации с применением калибровки дисперсии не учитывает влияние ReLU-нейронов. В одной из недавних работ [<https://arxiv.org/abs/1502.01852>.] была выведена формула инициализации, предназначенная специально для ReLU-нейронов:

*>>> w = np.random.randn(n) \* sqrt(2.0/n) # current recommendation*

### Сверточные нейронные сети

#### Свертка

Свертка — операция над парой матриц A (размера nx×ny) и B (размера mx×my), результатом которой является матрица C=A∗B размера (nx−mx+1)×(ny−my+1). Каждый элемент результата вычисляется как скалярное произведение матрицы B и некоторой подматрицы A такого же размера (подматрица определяется положением элемента в результате). То есть, .

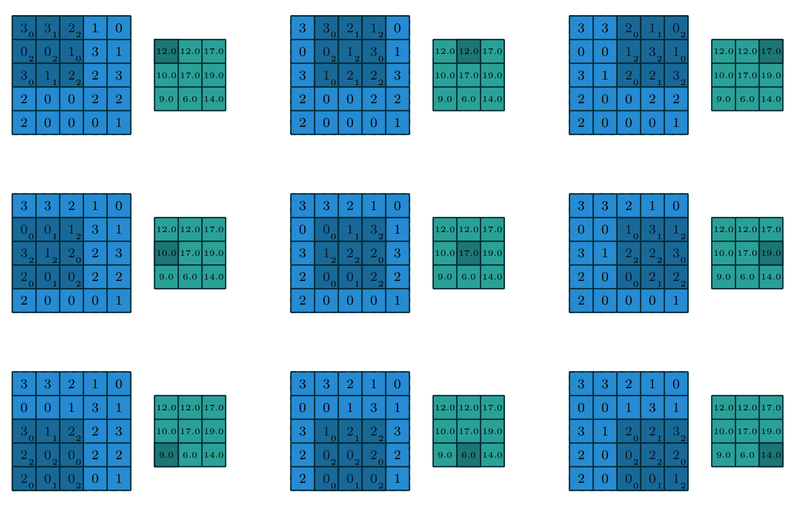


Рисунок 3.11 Пример свертки двух матриц

На рис 3.13 можно видеть, как матрица B «двигается» по матрице A, и в каждом положении считается скалярное произведение матрицы B и той части матрицы A, на которую она сейчас наложена. Получившееся число записывается в соответствующий элемент результата.

Логический смысл свертки такой — чем больше величина элемента свертки, тем больше эта часть матрицы A была похожа на матрицу B (похожа в смысле скалярного произведения). Поэтому матрицу A называют изображением, а матрицу B — фильтром или образцом.

В сверточной нейронной сети выходы промежуточных слоев образуют матрицу (изображение) или набор матриц (несколько слоёв изображения). Так, например, на вход сверточной нейронной сети можно подавать три слоя изображения (R-, G-, B-каналы изображения). Основными видами слоев в сверточной нейронной сети являются сверточные слои (англ. convolutional layer), пулинговые слои (англ. pooling layer) и полносвязные слои (англ. fully-connected layer/ dense layer).

#### Сверточные слои

Сверточный слой нейронной сети представляет из себя применение операции свертки к выходам с предыдущего слоя, где веса ядра свертки являются обучаемыми параметрами. Еще один обучаемый вес используется в качестве константного сдвига. При этом есть несколько важных деталей.

В одном сверточном слое может быть несколько сверток. В этом случае для каждой свертки на выходе получится своё изображение. Например, если вход имел размерность w×h, а в слое было n сверток с ядром размерности kx×ky, то выход будет иметь размерность n×(w−kx+1)×(h−ky+1).

Ядра свертки могут быть трёхмерными. Свертка трехмерного входа с трехмерным ядром происходит аналогично, просто скалярное произведение считается еще и по всем слоям изображения. Например, для усреднения информации о цветах исходного изображения, на первом слое можно использовать свертку размерности 3×w×h. На выходе такого слоя будет уже одно изображение (вместо трёх).

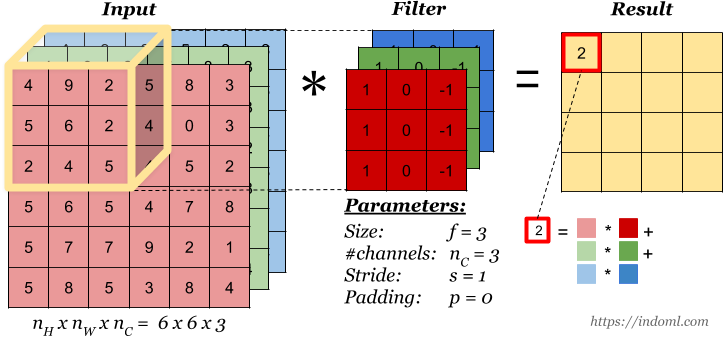


Рисунок 3.12 Пример свертки с трехмерным ядром

Можно заметить, что применение операции свертки уменьшает изображение. Также пиксели, которые находятся на границе изображения участвуют в меньшем количестве сверток, чем внутренние. В связи с этим в сверточных слоях используется дополнение изображени. Выходы с предыдущего слоя дополняются пикселями так, чтобы после свертки сохранился размер изображения. Такие свертки называют одинаковыми, а свертки без дополнения изображения называются правильными. Среди способов, которыми можно заполнить новые пиксели, можно выделить следующие:

zero shift: 00[ABC]00;

border extension: AA[ABC]CC;

mirror shift: BA[ABC]CB;

cyclic shift: BC[ABC]AB.

Еще одним параметром сверточного слоя является сдвиг. Хоть обычно свертка применяется подряд для каждого пикселя, иногда используется сдвиг, отличный от единицы — скалярное произведение считается не со всеми возможными положениями ядра, а только с положениями, кратными некоторому сдвигу s. Тогда, если если вход имел размерность w×h, а ядро свертки имело размерность kx×ky и использовался сдвиг s, то выход будет иметь размерность  .

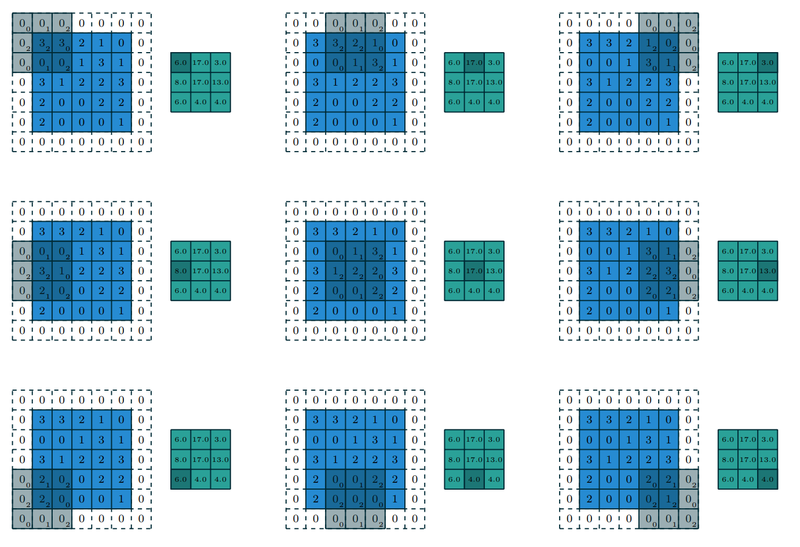


Рисунок 3.13 Пример свертки двух матриц с дополнением нулями и сдвигом 2

#### Пулинговые слои

Пулинговый слой призван снижать размерность изображения. Исходное изображение делится на блоки размером w×h и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума или взвешенного среднего. Обучаемых параметров у этого слоя нет. Основные цели пулингового слоя:

* уменьшение изображения, чтобы последующие свертки оперировали над большей областью исходного изображения;
* увеличение инвариантности выхода сети по отношению к малому переносу входа;
* ускорение вычислений



Рисунок 3.14 Пример операции пулинга с функцией максимума

### Регуляризация

Существует несколько методов контроля емкости нейронной сети, позволяющих предотвратить переобучение.

#### L1-регуляризация

L1-регуляризация является еще одним распространенным методом регуляризации. В рамках этого метода для каждого веса ω мы прибавляем к целевой функции слагаемое λ|ω|.

Применяется также комбинация L1- и L2-регуляризации: λ1|ω| + λ2ω2 Этот метод имеет название эластичная сеть (elastic net).

L1-регуляризация имеет интересное свойство, заключающееся в том, что в ее результате векторы весов становятся разреженными (т.е. очень близкими к нулю). Другими словами, нейроны с L1-регуляризацией в итоге используют только небольшое подмножество наиболее важных входов и, соответственно, почти не подвержены влиянию «шумных» входов.

На практике, если нет необходимости в непосредственном отборе признаков, L2-регуляризация обеспечит лучший результат по сравнению с L1-регуляризацией.

#### L2-регуляризация

L2-регуляризация, вероятно, является наиболее распространенным методом регуляризации. Данный метод штрафует модель с помощью квадратов весов. То есть, для каждого веса ω мы прибавляем к целевой функции слагаемое ½λω2 , где λ – коэффициент регуляризации. Множитель ½ используется для того, чтобы градиент этого слагаемого по параметру ω равнялся λω, а не 2λω. Интуитивная интерпретация L2-регуляризации заключается в том, что она сильно штрафует векторы весов с большими значениями, и слабо затрагивает векторы с умеренными значениями.

#### Ограничение нормы вектора весов

Еще одним методом регуляризации является метод ограничения нормы вектора весов (max norm constraint). В рамках данного метода мы задаем абсолютный верхний предел для нормы вектора весов каждого нейрона. Соблюдение ограничения обеспечивается с помощью проецируемого градиентного спуска (projected gradient descent). На практике это реализуется следующим образом: обновление весов выполняется как обычно, а затем вектор весов ω каждого нейрона ограничивается так, чтобы выполнялось условие ||ω||2 < c. Обычно значение c составляет порядка 3 или 4. Некоторые исследователи сообщают о положительном эффекте при использовании данного метода регуляризации. Одно из полезных свойств этого метода заключается в том, что он позволяет предотвратить «взрывной» рост весов даже при слишком большой скорости обучения, потому что обновления весов всегда ограничены.

#### Дропаут

Дропаут (dropout) – простой и очень эффективный метод регуляризации, дополняющий вышеназванные методы. Суть метода состоит в том, что в процессе обучения из общей сети случайным образом многократно выделяется подсеть, и обновление весов выполняется только в рамках этой подсети. Нейроны попадают в подсеть с вероятностью p, которая называется коэффициентом дропаута. Во время тестирования дропаут не применяется, вместо этого веса умножаются на коэффициент дропаута, в результате чего можно получить усредненную оценку для ансамбля всех подсетей. На практике коэффициент дропаута p обычно выбирают равным 0,5, но его можно подобрать с помощью валидационного набора данных.

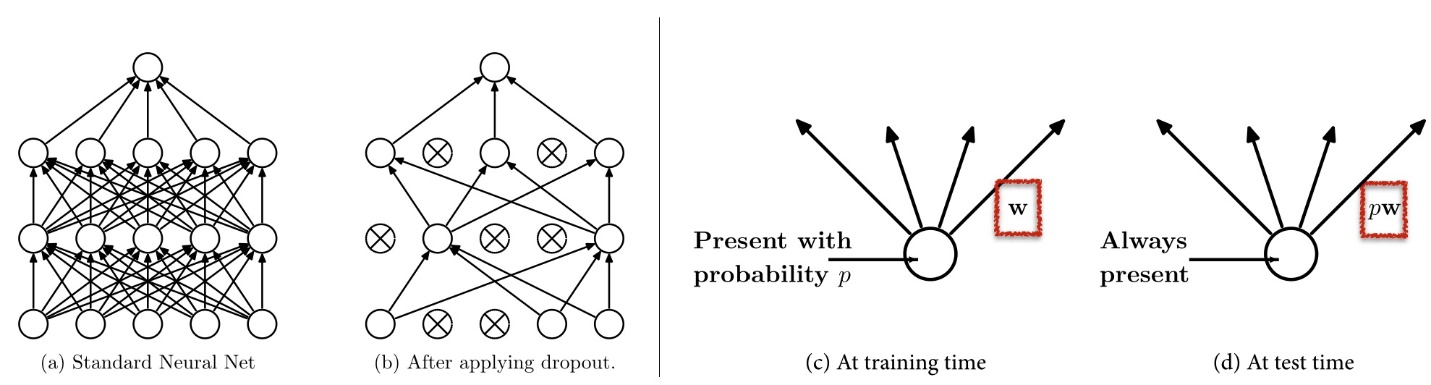


Рисунок 3.15 Dropout

### Ансамбли нейронных сетей

Ансамблевые методы предполагают обучение и совместное использование нескольких моделей. Как правило, ансамбль намного точнее отдельных моделей. Этот передовой подход позволил добиться высоких результатов при решении многих реальных задач. Почти все призеры различных конкурсов и соревнований по машинному обучению использовали ансамблевые методы.

Рассмотрим несколько вариантов создания моделей, которые можно объединить в ансамбль.

#### Модели с одинаковыми гиперпараметрами и различной инициализацией

С помощью кросс-валидации подбираем лучший набор гиперпараметров, а затем обучаем несколько моделей с одинаковым лучшим набором гиперпараметров, но с различной случайной инициализацией. Недостаток такого подхода заключается в том, что источником разнообразия служит только инициализация.

#### Модели с различными гиперпараметрами

С помощью кросс-валидации подбираем несколько лучших наборов гиперпараметров и объединяем в ансамбль модели с этими наборами гиперпараметров. Этот подход обеспечивает разнообразие, но существует опасность включения в ансамбль неоптимальных моделей. На практике данный подход легче реализовать по сравнению с предыдущим, поскольку после кросс-валидации не требуется дополнительное обучение моделей.

#### Различные стадии обучения одной модели

В том случае, когда процесс обучения является очень требовательным к ресурсам, можно объединить в ансамбль различные стадии обучения одной модели (checkpoint), например, после каждой эпохи. Этот подход не обеспечивает большого разнообразия, но на практике может дать достаточно хорошие результаты.

## Результаты сравнения

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.

### Параметры и их значения

1. Максимальная высота фильтров
   1. 3
   2. 4
   3. 5
2. Количество сверточных слоев
   1. 3
   2. 6
   3. 10
3. Высота плотного слоя
   1. 15
   2. 30
   3. 60
4. Функция активации
   1. Tanh (гиперболический тангенс)
   2. Sigmoid
   3. Hard\_sigmoid (упрощенная сигмоида)
   4. ReLU (rectified linear unit)
5. Dropout-регуляризация
   1. 0 (без регуляризации)
   2. 0.1
   3. 0.2

### 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).

### Результаты работы программы

Таблица с результатами работы программы доступна в приложении 4.

#### Точность

**Средняя:**



Рисунок 3.16 Средняя точность для функций активации

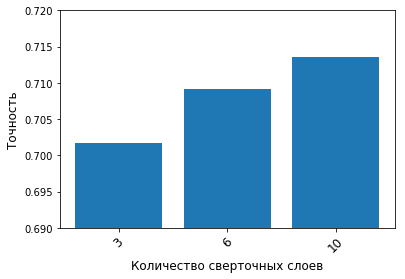


Рисунок 3.17 Средняя точность для числа сверточных слоев

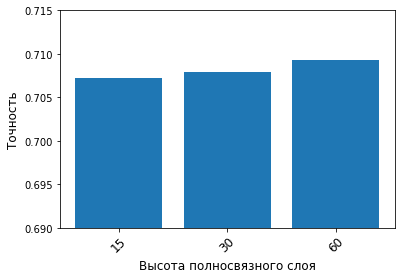


Рисунок 3.18 Средняя точность для высоты полносвязного слоя

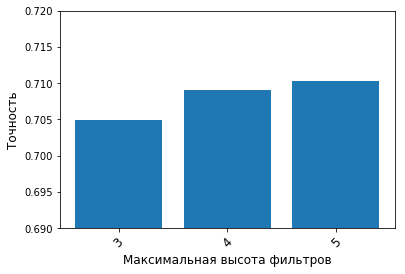


Рисунок 3.19 Средняя точность для высоты фильтров

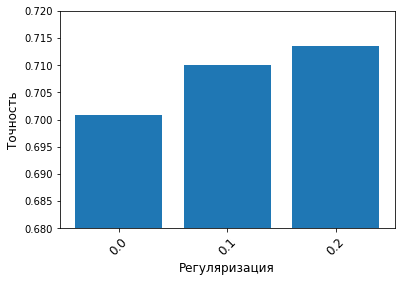


Рисунок 3.20 Средняя точность для регуляризации

**Лучшая:**

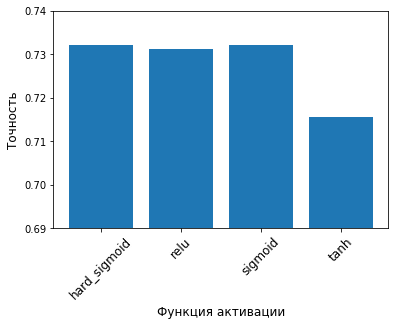
****

Рисунок 3.21 Лучшая точность для фунцкий активации

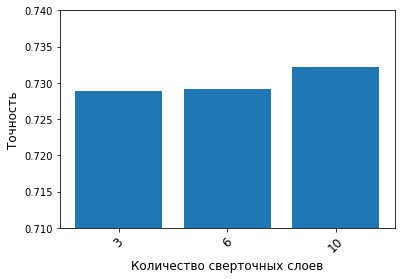
****

Рисунок 3.22 Лучшая точность для количества сверточных слоев

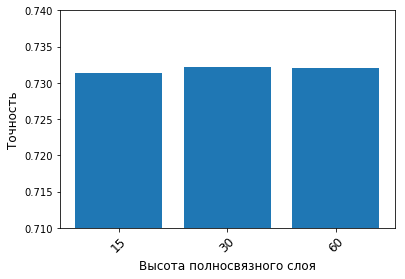
****

Рисунок 3.23 Лучшая точность для высоты полносвязного слоя

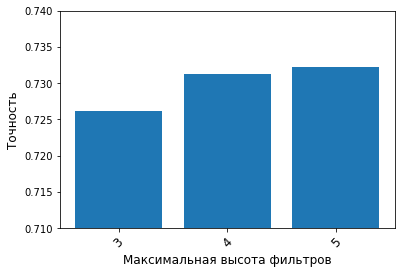
****

Рисунок 3.24 Лучшая точность для максимальной высоты фильтров

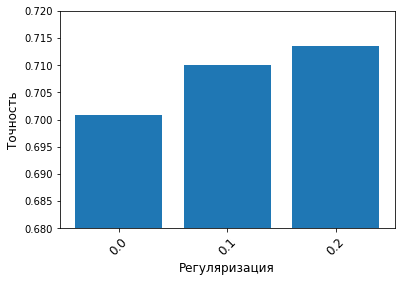
****

Рисунок 3.25 Лучшая точность для регуляризации

#### F-мера

**Средняя:**

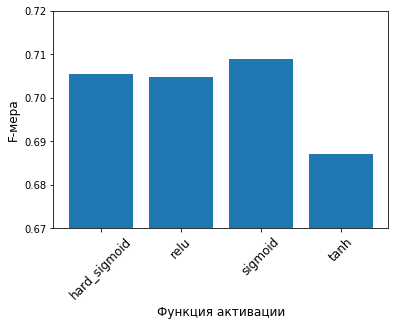
****

Рисунок 3.26 Средняя F-мера для функции активации

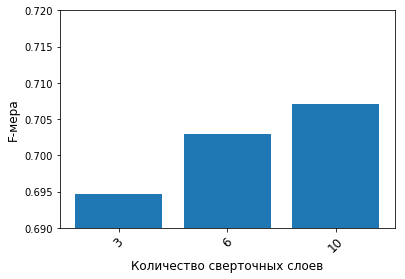
****

Рисунок 3.27 Средняя F-мера для количества сверточных слоев

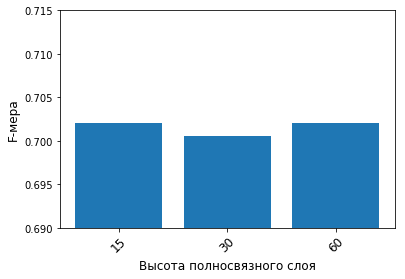
****

Рисунок 3.28 Средняя F-мера для высоты полносвязного слоя

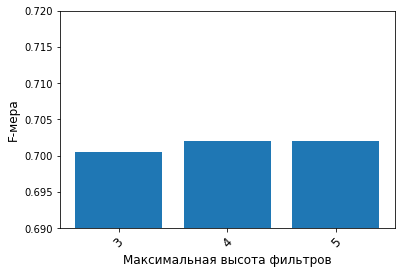
****

Рисунок 3.29 Средняя F-мера для максимальноый высоты фильтров

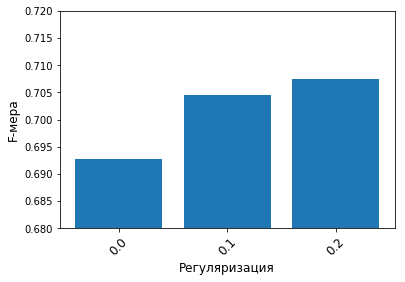
****

Рисунок 3.30 Средняя F-мера для регуляризации

**Лучшая:**

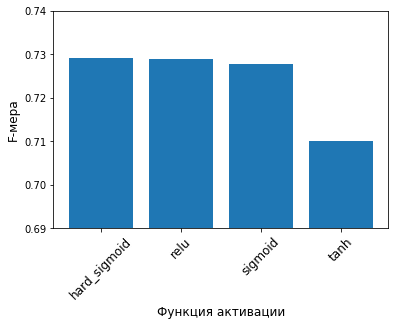
****

Рисунок 3.31 Лучшая F-мера для фунции активации

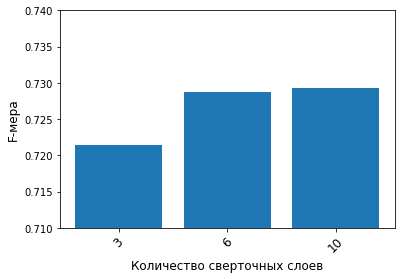
****

Рисунок 3.32 Лучшая F-мера для количества сверточных слоев

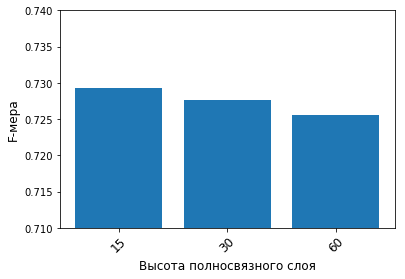
****

Рисунок 3.33 Лучшая F-мера для высоты полносвязного слоя

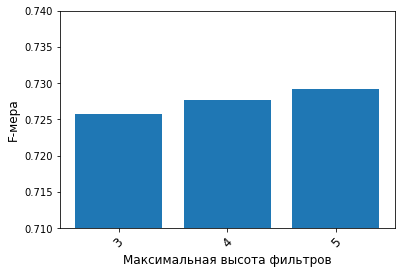
****

Рисунок 3.34 Лучшая F-мера для максимальной высоты фильтров

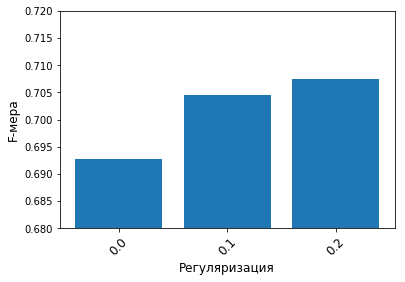
****

Рисунок 3.35 Лучшая F-мера для регуляризации

## Итоговая архитектура нейронной сети

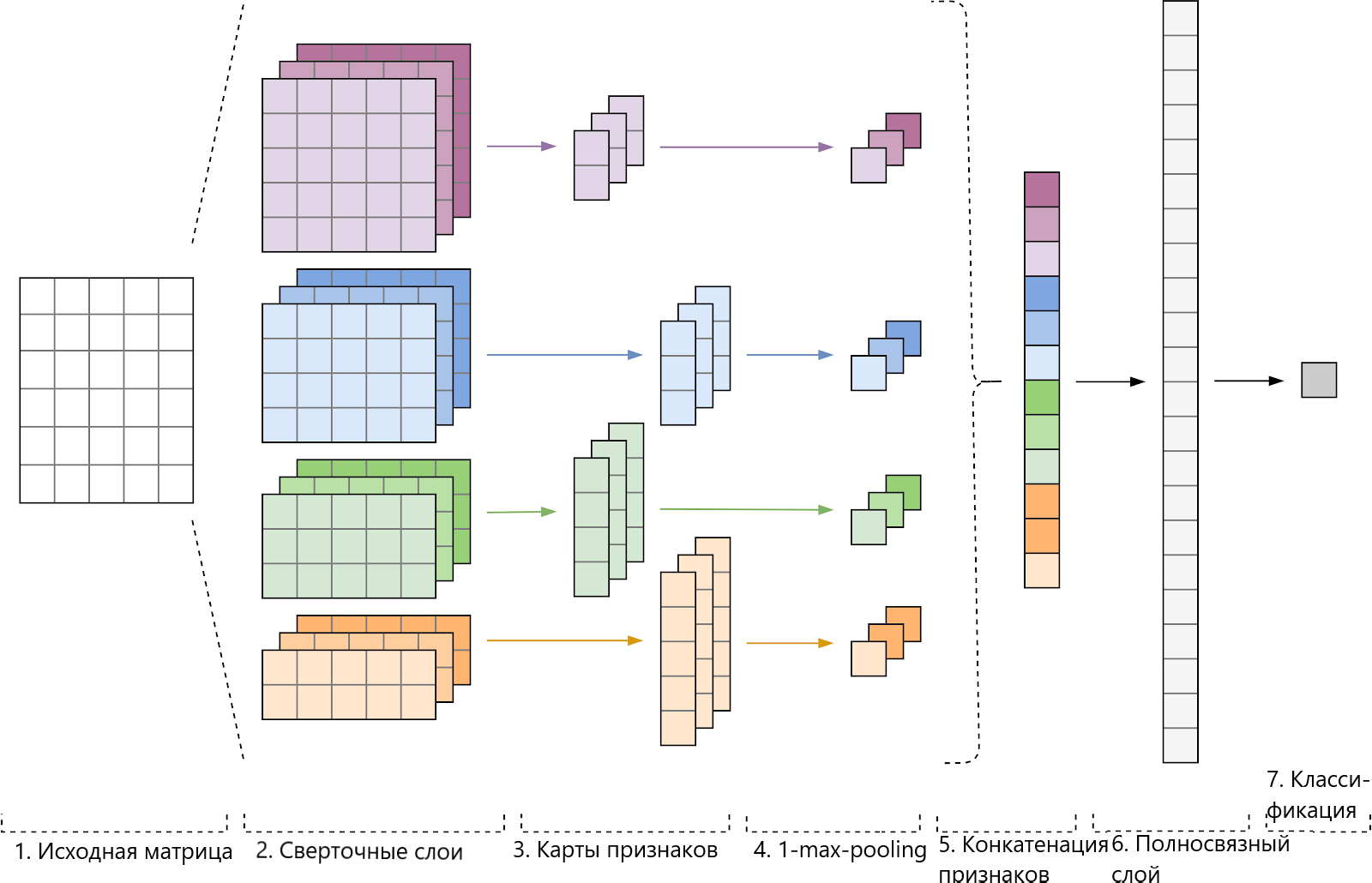


Рисунок 3.36 Итоговая архитектура нейронной сети

Таким образом, итоговая нейронная сеть состоит из следующих слоев:

1. Входная матрица размерности n\*m, где n – размерность пространства Word2vec, а m – количество слов в предложении.
2. Сверточные слои высотой 2, 3, 4 и 5 (по 10 слоев каждого вида) с ReLU –активацией.
3. Слои субдискретизации. Размерность выходной матрицы признаков для каждого фильтра варьируется в зависимости от высоты этого фильтра и высоты исходной матрицы.
4. Карты признаков, уплотненные с помощью 1-max-pooling для уменьшения размерности. Таким образом извлекается наиболее важная информация для каждой свертки независимо от её положения в тексте. Другими словами, для используемого векторного отображения комбинация слоев свёртки и слоев субдискретизации позволяет извлекать из текста наиболее значимые n-граммы.
5. Карты признаков, рассчитанные на выходе каждого слоя субдискретизации, конкатенируются в один общий вектор признаков
6. Полносвязный слой высотой 15 с функцией активации ReLU. К слою применена регуляризация Dropout(0.2).
7. Выходной классификатор с сигмоидальной активацией

# Программная реализация

## Архитектура программы

Итоговая программа представлена в виде тетради jupyter notebook. Код доступен в приложении 1.

Тетрадь разделена на блоки, что позволяет запускать отдельные ее части независимо. Например, существует группа блоков, которая строит, обучает и сохраняет обученную нейронную сеть в виде файла на диск. После этого следующая группа блоков читает этот файл и использует обученную нейронную сеть для классификации текстов и оценки результатов. При следующем запуске этап с обучением нейронной сети можно будет пропустить и сразу запускать блоки классификации (если не появилось новых данных, на которых нужно обучить программу).

Также была написана программа для построения Word2Vec файла из корпуса текстов (приложение 2). Она принимает на вход корпус текстов в виде базы данных sqlite и создает обученную модель Word2Vec, сохраняя ее в файле .w2v. В основной программе используется предобученная модель w2v, так как вычислительной мощности домашнего компьютера недостаточно для обучения на большом корпусе текстов. Тем не менее, эта программа может быть полезна для, например, обучения на специфическом наборе текстов, использующем нестандартный набор лексики.

Вспомогательные программы для анализа лучших методов классификации и оптимизации архитектуры нейронной сети подробно описаны в главах 2 и 3 соответственно и доступны в приложениях 3 и 4.

## Использованные программные средства

### Python

Все программы были разработаны на языке python, являющимся де-факто стандартом в области машинного обучения. Причины выбора данного языка:

* Скриптовый язык (позволяет запускать отдельные части кода)
* Интеграция с инструментами для машинного обучения (перечислены далее)
* Производительность (чувствительные к производительности части кода компилируются в C, что дает скорость вычислений, приближенную к компилируемым языкам).

### IPython + Jupyter notebook

IPython представляет собой мощный инструмент для работы с языком Python. Базовые компоненты IPython – это интерактивная оболочка для с широким набором возможностей и ядро для Jupyter. Jupyter notebook является графической веб-оболочкой для IPython, которая расширяет идею консольного подхода к интерактивным вычислениям.

Основные отличительные особенности данной платформы – это комплексная интроспекция объектов, сохранение истории ввода на протяжении всех сеансов, кэширование выходных результатов, расширяемая система команд, логирование сессии, дополнительный командный синтаксис, подсветка кода, доступ к системной оболочке, стыковка с pdb отладчиком и Python-профайлером.

IPython позволяет подключаться множеству клиентов к одному вычислительному ядру и, благодаря своей архитектуре, может работать в параллельном кластере.

В Jupyter notebook возможно разрабатывать, документировать и выполнять приложения на языке Python, он состоит из двух компонентов: веб-приложение, запускаемое в браузере, и тетради – файлы, в которых можно работать с исходным кодом программы, запускать его, вводить и выводить данные и т.п.

Веб приложение позволяет:

* редактировать Python код в браузере, с подсветкой синтаксиса, автоотступами и автодополнением;
* запускать код в браузере;
* отображать результаты вычислений с медиа представлением (схемы, графики);
* работать с языком разметки Markdown и LaTeX.

Тетради – это файлы, в которых сохраняются исходный код, входные и выходные данные, полученные в рамках сессии. Фактически, он является записью вашей работы, но при этом позволяет заново выполнить код, присутствующий на нем. Тетради можно экспортировать в форматы PDF, HTML.

### Keras

Keras — открытая нейросетевая библиотека, написанная на языке Python. Она представляет собой надстройку над фреймворками Deeplearning4j, TensorFlow и Theano. Нацелена на оперативную работу с сетями глубинного обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. Она была создана как часть исследовательских усилий проекта ONEIROS (англ. Open-ended Neuro-Electronic Intelligent Robot Operating System).

Keras предоставляет высокоуровневый, интуитивный набор абстракций, который делает простым формирование нейронных сетей, независимо от используемой в качестве вычислительного бэкенда библиотеки научных вычислений. Microsoft работает над добавлением к Keras и низкоуровневых библиотек CNTK.

Эта библиотека содержит многочисленные реализации широко применяемых строительных блоков нейронных сетей, таких как слои, целевые и передаточные функции, оптимизаторы, и множество инструментов для упрощения работы с изображениями и текстом.

# 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.

# 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](http://yury.name/modern.html)» (постановка задачи, построение и обучение классификатора, оценка качества).
3. [F. Sebastiani. Machine Learning in Automated Text Categorization](http://nmis.isti.cnr.it/sebastiani/Publications/ACMCS02.pdf)
4. ["Text mining. Классификация текста".](http://statosphere.ru/blog/135-text-mining1.html) 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