СТАТЬИ НА РУССКОМ ЯЗЫКЕ

ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ

DOI - 10.32743/UniTech.2023.107.2.15064

АНАЛИЗ АЛГОРИТМОВ КЛАССИФИКАЦИИ ТЕКСТОВ

Логунова Татьяна Викторовна
канд. техн. наук, доц. кафедры О7,
Балтийский Государственный Технический Университет «ВОЕНМЕХ»
им. Д.Ф. Устинова,
РФ, г. Санкт-Петербург

Щербакова Лидия Викторовна
канд. техн. наук, доц. кафедры О7,
Балтийский Государственный Технический Университет «ВОЕНМЕХ»
им. Д.Ф. Устинова,
РФ, г. Санкт-Петербург

Васюков Василий Михайлович
ст. преподаватель,
Балтийский Государственный Технический Университет «ВОЕНМЕХ»
им. Д.Ф. Устинова,
РФ, г. Санкт-Петербург

Шимкун Вячеслав Владиславович
преподаватель,
Балтийский Государственный Технический Университет «ВОЕНМЕХ»
им. Д.Ф. Устинова,
РФ, г. Санкт-Петербург
E-mail: d0ct0r@list.ru

ANALYSIS OF ALGORITHMS FOR TEXT CLASSIFICATION

Tatiana Logunova
Candidate of Technical Sciences,
Associate Professor of the O7 Department,
Baltic State Technical University "VOENMEH" them. D.F. Ustinov,
Russia, St. Petersburg

Lidiia Shcherbakova
Candidate of Technical Sciences,
Associate Professor of the O7 Department,
Baltic State Technical University "VOENMEH" them. D.F. Ustinov,
Russia, St. Petersburg

Vasilii Vasiukov
Senior Lecturer,
Baltic State Technical University
"VOENMEH" them. D.F. Ustinov,
Russia, St. Petersburg

Viacheslav Shimkun
Lecturer,
Baltic State Technical University
"VOENMEH" them. D.F. Ustinov,
Russia, St. Petersburg

Библиографическое описание: АНАЛИЗ АЛГОРИТМОВ КЛАССИФИКАЦИИ ТЕКСТОВ // Universum: технические науки: электрон. научн. журн. Логунова Т.В. [и др.]. 2023. 2(107). URL: *https://7universum.com/ru/tech/archive/item/15064*

АННОТАЦИЯ

Во многих программных продуктах существует потребность в модуле классификации текстов, например, при решении следующих задач возникает необходимость решать задачу классификации текстов:

- распознавание эмоциональной окраски текстов,
- фильтрация спама,
- анализ тональности текста,
- разделение документов по тематическим каталогам,
- создание голосовых пользовательских интерфейсов.

В целом, любые задачи классификации, где часть данных являются текстовыми, можно рассматривать как задачи классификации текстов.

При классификации текста, над текстом, как правило, осуществляются следующие последовательные операции: предобработка текста, извлечение признаков, классификация текста.

Разработчики программных модулей классификации текстов сталкиваются с проблемой выбора какого-то варианта для каждой из вышеперечисленных операций.

Так как на каждом шаге существует достаточно большое количество возможных вариантов, то на поиск лучших вариантов, зачастую, необходимо значительное время и вычислительные ресурсы. Неправильный выбор на любом шаге построения модуля классификации текстов ведет к снижению качества классификации. При этом трудозатраты на проведение подобных исследований при разработке программного обеспечения могут быть сопоставимы с трудозатратами на разработку самого программного обеспечения. В данной статье рассматриваются все вышеперечисленные операции и предлагается анализ алгоритмов классификации текстов.

ABSTRACT

In many software products, there is a need for a text classification module, for example, when solving the following tasks, it becomes necessary to solve the problem of text classification:

- recognition of the emotional coloring of texts,
- spam filtering,
- analysis of the tone of the text,
- division of documents into thematic directories,
- creation of voice user interfaces.

In general, any classification problem, where part of the data is textual, can be considered as a text classification problem.

When classifying a text, the following sequential operations are usually performed on the text: text preprocessing, feature extraction, text classification.

Developers of software modules for text classification are faced with the problem of choosing some option for each of the above operations.

Since at each step there are a fairly large number of possible options, the search for the best options often requires significant time and computational resources. Wrong choice at any step of building a text classification module leads to a decrease in the quality of classification. At the same time, the labor costs for conducting such research in software development can be comparable to the labor costs for developing the software itself. This article discusses all of the above operations and proposes an analysis of text classification algorithms.

Ключевые слова: алгоритмы классификации текстов; мешок слов; методы извлечения признаков; предобработка текста; векторное представление; градиентный бустинг; многослойный персептрон; нейронные сети.

Keywords: text classification algorithms; bag of words; feature extraction methods; text preprocessing; vector representation; gradient boosting; multilayer perceptron; neural networks.

1. Параметры влияющие на качество классификации текста

Процесс классификации текста, обычно, состоит из следующих трёх шагов:

- предобработка текста;
- извлечение признаков из текста;
- классификация текста с помощью некоторого алгоритма.

Любой из этих шагов влияет на качество классификации, поэтому для проведения сравнительного эксперимента возникает необходимость рассмотрения различных вариантов процесса классификации применительно к различным задачам классификации. В данном разделе описываются данные три шага процесса, а именно:

- из чего, обычно, состоит предобработка текста;
- какие наиболее распространённые методы извлечения признаков существуют;
- какие алгоритмы классификации можно использовать для решения задачи классификации текста.

2. Предобработка текста

Наиболее часто встречающаяся предобработка текста содержит следующие шаги [1]:

- удаление всех нерелевантных символов (например, любые символы, не относящиеся к цифробуквенным);
- разбиение текста на токены;
- удаление нерелевантных слов (например, упоминания в Twitter или URL-ы);
- перевод всех слов в один регистр (например, нижний) для уменьшения количества одинаковых слов;

- удаление стоп-слов;
- проведение стемминга или лемматизации.

Токенизация текста – это метод предварительной обработки текстов, при котором текст разбивается на слова, фразы, символы или другие значимые элементы, называемые токенами.

Стоп-слова – это слова, не несущие никакой смысловой нагрузки. Например, слова "и", "в", "только" не несут никакой ценности и только добавляют шум в данные. При этом надо понимать, что невозможно создать универсальный список стоп-слов, для каждого конкретного случая список будет отличаться.

Обычно тексты содержат разные грамматические формы одного и того же слова, а также могут встречаться однокоренные слова. Лемматизация и стемминг преследуют цель привести все встречающиеся словоформы к одной, нормальной словарной форме.

Стемминг – это грубый эвристический процесс, который отрезает «лишнее» от корня слов, часто это приводит к потере словообразовательных суффиксов.

Лемматизация – это более тонкий процесс, который использует словарь и морфологический анализ, чтобы в итоге привести слово к его канонической форме – лемме.

Отличие в том, что стеммер (конкретная реализация алгоритма стемминга) действует без учёта контекста и, соответственно, не делает разницы между словами, которые имеют разный смысл в зависимости от части речи. Однако у стеммеров есть своё преимущество – они работают быстрее.

Большинство современных алгоритмов машинного обучения ориентированы на признаковое описание объектов, поэтому все документы обычно переводят в вещественное пространство признаков. Для этого используют идею о том, что за принадлежность документа к некоторому классу отвечают слова, а тексты из одного класса будут использовать много схожих слов.

3. Методы извлечения признаков

3.1 Bag-of-words

При использовании данного подхода каждому тексту сопоставляется вектор размерности словаря набора текстов, что описывается формулой (1). Под словарём подразумевается множество всех слов входящих в набор текстов.

$$d \to v \in \mathbb{R}^m: v_i = n_{w_i}, w_i \in W, i=1 \ldots m \qquad (1)$$

где: d – документ,

$v \in \mathbb{R}$ – последовательность векторов размерности m,

m – размер словаря,

i -

w – слово из словаря W,

n_w – количество вхождений слова w в документ d.

При всей простоте реализации данный подход имеет ряд недостатков:

- для больших наборов текстов размерность словаря, а, следовательно, и размерность вектора, представляющего текст, может исчисляться сотнями тысяч, а иногда и миллионами;
- не учитывается контекст слова в документе.

3.2 Bag-of-words и TF-IDF

Как и в методе Bag-of-words, каждый документ представляется вектором, но координаты этого вектора рассчитываются в соответствии со статистической мерой TF-IDF [1].

TF-IDF (Term Frequency – Inverse Document Frequency) – статистическая мера, используемая для оценки важности слова в контексте документа. Вычисляется по формуле (2). Большой вес в TF-IDF получают слова с высокой частотой в пределах конкретного документа и с низкой частотой употребления в других документах.

$$TF\text{-}IDF\ (w, d, D) = TF(w, d) \cdot IDF(w, D), \qquad (2)$$

где w – слово из словаря,

d – документ из множества документов D,

TF – частота слова, оценивает важность слова w в пределах отдельного документа согласно формуле (3),

IDF – обратная частота документа.

$$TF(w, d) = \frac{n_w}{\sum_{i=1}^{m} n_{w_i}}, \qquad (3)$$

где n_w – количество вхождений слова w в документ d,

m – размер словаря.

Учёт IDF уменьшает вес широко употребляемых слов, что показывает формула (4).

$$IDF(w, D) = \log \frac{|D|}{|\{d \in D \mid w \in d\}|}, \qquad (4)$$

где $|D|$ – количество документов в коллекции,

$|\{d \in D \mid w \in d\}|$ – число документов из коллекции D, в которых встречается слово w.

3.3 Bag-of-ngrams и TF-IDF

Часто информацию в тексте несут не только отдельные слова, но и некоторая последовательность слов, например, фразеологизмы – устойчивые сочетание слов, значение которых не определяется значением входящих в них слов, взятых по отдельности. Речевой оборот «как рыба в воде» означает чувствовать себя уверенно, очень хорошо в чем-либо разбираться. Смысл данного выражения будет передан неверно, если учитывать его слова по отдельности.

Для того чтобы учесть такие особенности языка предлагается при переводе текстов в векторное представление учитывать помимо слов, N-граммы.

N-граммы – это последовательности из N слов. К примеру, для текста «мама мыла раму» получаем биграммы «мама мыла» и «мыла раму». В задаче классификации текстов N-граммы являются индикаторами того, что данные N слов встретились рядом. На практике в словарь добавляют не все N-граммы, а только те, которые встречаются достаточно часто в текстах, а также редко применяют N-граммы длиннее, чем триграммы, так как словосочетания длинны более трёх редко встречаются в текстах.

Метод Bag-of-ngrams и TF-IDF аналогичен методу Bag-of-words и TF-IDF, только вектор признаков для каждого документа содержит TF-IDF k-грамм, где k = 1…N.

3.4 Word2vec

В алгоритме word2vec [2], [3] каждому слову сопоставляется вектор из вещественных чисел евклидова пространства $\mathbb{R}^d$ для некоторого d (обычно несколько сотен). Базовое положение состоит в том, что геометрические соотношения в пространстве $\mathbb{R}^d$ будут соответствовать семантическим соотношениям между словами, например, ближайшие соседние слова в этом пространстве окажутся его синонимами или другими тесно связанными словами и т. д. Данное предположение основано на дистрибутивной гипотезе – слова с похожим смыслом будут встречаться в похожим контекстах.

Основные преимущества word2vec перед статистическими подходами:

- размерность вектора, представляющего слово, не зависит от размера словаря, а задаётся при обучении модели;
- вектора в word2vec моделируют семантические отношения между словами.

Модель word2vec была представлена сразу в двух вариантах: в виде непрерывного мешка слов (continuous bag of words, CBOW) и в виде архитектуры skip-gram.

Введём следующие обозначения:

i – слово,

c_k – слово из контекста слова i (слово, находящееся перед словом i), $k = 1…n$,

n – размер окна – количество слов, находящихся перед словом i, которые модель будет учитывать,

w_k – вектор из контекста слова i (вектор, представляющий собой слово c_k), $k = 1…n$,

L – функция потерь модели (её модель и будет оптимизировать).

Архитектура сети CBOW представлена на рисунке 1 [2].

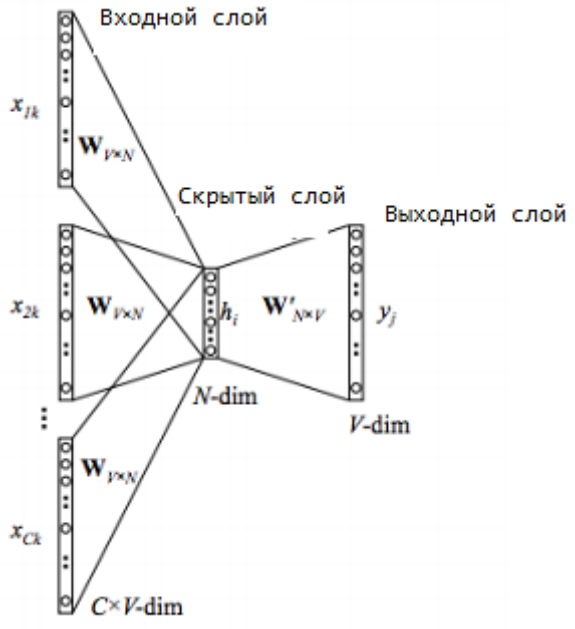

Рисунок 1. Архитектура сети CBOW

Принцип работы сети CBOW заключается в следующем:

- каждый вход сети – это вектор в one-hot представлении (вектор, состоящий из нулей и единиц, где всего одна единица в позиции, обозначающей номер слова в словаре) размерности V, где V – размер словаря;
- скрытый слой сети – это фактически и есть матрица W векторных представлений слов; n-я строка W содержит представление n-го слова из словаря;
- при вычислении выхода скрытого слоя берётся среднее всех входных векторов;
- на выходе получается некая оценка u_j для каждого слова в словаре; затем апостериорное распределение модели вычисляется с помощью функции softmax, описанной по формуле (5):

$$\hat{p}(i|c_1, c_2, \dots, c_n) = \frac{e^{u_j}}{\sum_{j'=1}^{V} e^{u_{j'}}} \quad (5)$$

- таким образом, функция потери на одном окне состоит в том, чтобы сделать апостериорное распределение как можно более похожим на распределение данных, как показано в формуле (6):

$$L = -\log \hat{p}(i|c_1, c_2, \dots, c_n) = -u_j + \log \sum_{j'=1}^{V} e^{u_{j'}} \quad (6)$$

где u_j – это некая оценка для каждого слова в словаре.

А модель skip-gram работает прямо противоположным образом – модель пытается предсказать каждое слово контекста по данному слову. Архитектура соответствующей сети показан на рисунке 2 [2].

На выходном слое сети получается n мультиномиальных распределений, по одному на каждое слово контекста (7).

$$\hat{p}(\mathrm{c}_k|i) = \frac{e^{u_{c_k}}}{\sum_{j'=1}^{V} e^{u_{j'}}} \quad (7)$$

Функция потери на одном окне может быть выражена формулой (8).

$$L = -\log \hat{p}(c_1, c_2, \dots, c_n|i) = -\sum_{k=1}^{n} u_k + n \log \sum_{j'=1}^{V} e^{u_{j'}} \quad (8)$$

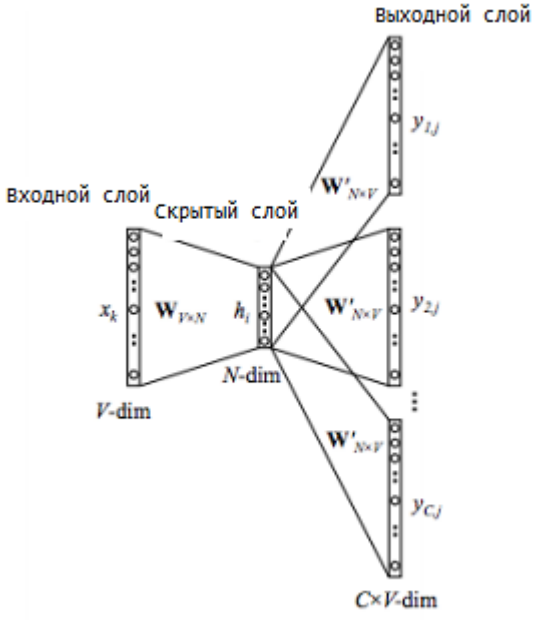

Рисунок 2. Архитектура сети skip-gram

В обоих подходах нейронная сеть используется для обучения матрицы векторных представлений слов, которые являются весами скрытого слоя нейронной сети. При этом размерность векторного представления слова (которая является размерностью выходного слоя нейронной сети) задается при обучении модели, так же, как и длина контекста (какое количество слов будет оказывать влияние на данное слово).

3.5 Doc2vec

Целью алгоритма doc2vec является создание векторного представления документа, вне зависимости от его длины. Для реализации этой цели используется модель word2vec с добавлением ещё одного вектора, который называется Paragraph ID [4]. Теперь для того чтобы предсказать следующее слово используются не только слова, но и уникальный вектор документа, в котором это слово предсказывается.

Таким образом обучается не только матрица *W*, соответствующая векторному представлению слов, но и матрица *D*, строки которой соответствуют векторному представлению документов. Такая модель называется Distributed Memory version of Paragraph Vector (PV-DM) и является изменённой версией модели word2vec CBOW. Архитектура модели PV-DM показана на рисунке 3 [4].

Другой вариант архитектуры модели doc2vec основан на модели word2vec skip-gram называется Distributed Bag of Words version of Paragraph Vector (PV-DBOW). Архитектура модели PV-DBOW показана на рисунке 4 [4].

В данном случае модель обучается путём предсказывания для текстового документа слов, которые могут в него входить.

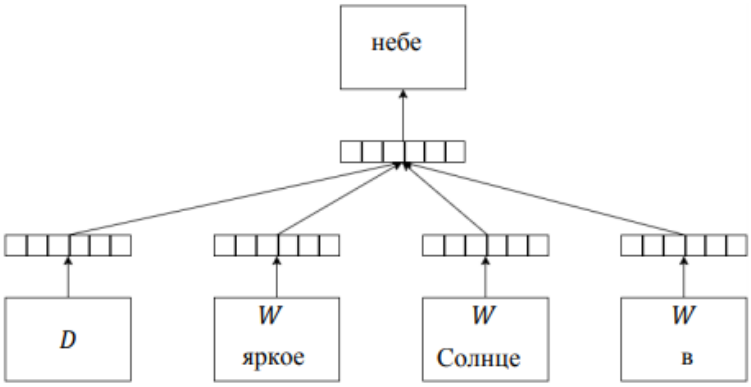

Рисунок 3. Архитектура модели doc2vec PV-DM

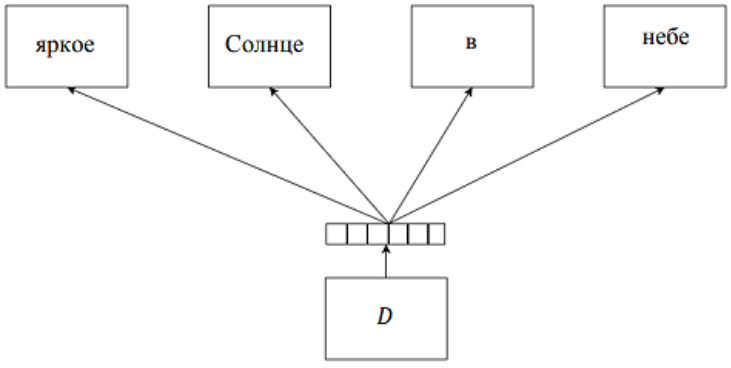

Рисунок 4. Архитектура модели doc2vec PV-DBOW

3.6 GloVe

В отличие от алгоритма word2vec, алгоритм GloVe опирается на глобальные статистики. В качестве такой глобальной статистики берётся вероятность совместного появления слов в документах [5].

Введём некоторые обозначения:

X – матрица совместной встречаемости слов,

X_{ij} (элементы матрицы X) – количество раз, когда слово j появилось в контексте слова i,

$X_i - \sum_k X_{ik}$ – сумма элементов строки i матрицы X,

$P_{ij} - \frac{X_{ij}}{X_i}$ – вероятность появления слова j в контексте слова i.

Рассмотрим отношение вероятностей встречаемости слов i и j к пробному слову k, описанное в формуле (9). Отношения между словами i и j можно понять, изучая отношения вероятностей их появления для различных пробных слов k (в силу дистрибутивной гипотезы).

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}, \quad (9)$$

где w_i, w_j – вектора слов i и j,

$\tilde{w}_k$ – вектор пробного слова.

После некоторых предположений о функции F (зависит не от двух аргументов w_i, w_j, а от их разности и имеет единственный аргумент – число) ее можно представить в виде формулы (10).

$$F\left(\left(w_i - w_j\right)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \quad (10)$$

В силу симметрии (замена X на X^T) можно потребовать от соотношения, описанного в формуле (10) быть представимым в виде формулы (11).

$$F\left(\left(w_i - w_j\right)^T \tilde{w}_k\right) = \frac{F({w_i}^T \tilde{w}_k)}{F({w_j}^T \tilde{w}_k)} \quad (11)$$

Из соотношений, описанных в формулах (10) и (11) можно получить формулу (12).

$$F({w_i}^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i} \quad (12)$$

Это соотношение можно разрешить, приняв за функцию F экспоненту, описанную в формуле (13).

$$F{w_i}^T \tilde{w}_k = \log X_{ik} - \log X_i \quad (13)$$

Так как в соотношении, описанном в формуле (13) X_i не зависит от k, поэтому можно заменить X_i на два смещения (их два для поддержания симметрии соотношения относительно i и k), получая формулу (14).

$${w_i}^T \tilde{w}_k + b_i + \tilde{b}_k = \log X_{ik} \quad (14)$$

где b – это смещение.

В статье [5] предлагается в качестве функции потерь взять взвешенную среднеквадратичную ошибку, описанную в формуле (15) с весовой функцией, описанной в формуле (16). Оптимизируя функционал J, описанный в формуле (15), можно получить векторы w_i.

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left({w_i}^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 \quad (15)$$

$$f(x) = \begin{cases} (x/x_{max})^a, \text{если } x < x_{max} \\ 1, \text{в другом случае} \end{cases} \quad (16)$$

4. Алгоритмы классификации

4.1 Логистическая регрессия

Логистическая регрессия [1] является линейной моделью, описанной в формуле (17), позволяющей предсказывать вероятности принадлежности объекта к данному классу. К преимуществам логистической регрессии относятся: интерпретируемость человеком и высокая скорость работы алгоритма, а к недостаткам – низкая точность, по сравнению с более сложными моделями.

$$z = \sum_{i=1}^{n} \theta_i * x_i + \theta_0, \quad (17)$$

где $\theta = (\theta_i), i = 0..n$ – вектор параметров модели;

$x = (x_i), i = 1..n$ – вектор, описывающий объект.

В логистической регрессии делается предположение о том, что вероятность принадлежности объекта x к классу 1 будет выражаться формулой (18).

$$P\{y = 1|x\} = f(z) = \frac{1}{1+e^{-z}}, \quad (18)$$

Обучающая выборка состоит из пар $\left(x^{(i)}, y^{(i)}\right), i = 1..m$.

При подборе параметров модели используется метод максимального правдоподобия, согласно которому выбираются параметры θ, максимизирующие значение функции правдоподобия, что описано в формуле (19) на обучающей выборке.

$$\begin{aligned} \hat{\theta} &= \arg\max_{\theta} L(\theta) \\ &= \arg\max_{\theta} \prod_{i=1}^{m} P\left\{y = y^{(i)} \middle| x = x^{(i)}\right\} \end{aligned} \quad (19)$$

Максимизация функции правдоподобия эквивалентна максимизации её логарифма, согласно формуле (20).

$$\begin{aligned} \ln L(\theta) = \sum_{i=1}^{m} & y^{(i)} \ln f(\theta^T x^{(i)}) \\ & + \left(1 - y^{(i)}\right) \ln\left(1 - f(\theta^T x^{(i)})\right) \end{aligned} \quad (20)$$

Выражение из формулы (20), обычно, максимизируется с использованием градиентного спуска.

4.2 Наивный байесовский классификатор

Наивный байесовский классификатор – это набор алгоритмов, основанных на применении теоремы Байеса с «наивным» предположением об условной независимости между каждой парой признаков при заданном значении переменной класса [6].

Функционал среднего риска из формулы (21) – ожидаемая величина потери при классификации объектов алгоритмом a.

Существует теорема об оптимальности байесовского классификатора [6], согласно которой, если известны априорные вероятности $P(y)$, функции правдоподобия $p(x|y)$ и величина потери λ_y при ошибке на объекте класса $y \in Y$, тогда минимум среднего риска $R(a)$ достигается алгоритмом по формуле (22).

$$R(a) = \sum_{y \in Y} \lambda_y \int [a(x) \neq y]\, p(x,y)dx \qquad (21)$$

где a – алгоритм классификации объектов,

x – элемент матрицы X

$$a(x) = \arg\max_{y \in Y} \lambda_y P(y) p(x|y) \qquad (22)$$

В наивном байесовском классификаторе делается предположение, что все признаки, описывающие объект x, являются независимыми случайными величинами в формуле (23).

$$p(x|y) = \prod_{i=1}^{n} p_i(x_i|y), \qquad (23)$$

где $p_i(x_i|y)$ – плотность распределения значений i-ого признака для класса y;

x – объект из множества X, $x = (x_1, \dots, x_n)$.

С учетом формулы (23) классификатор из формулы (22) можно записать в виде формулы (24).

$$a(x) = \arg\max_{y \in Y} \left(\ln \lambda_y \hat{P}(y) + \sum_{i=1}^{n} \ln \hat{p}_i(x_i|y) \right) \qquad (24)$$

Наивный байесовский классификатор может быть как параметрическим, так и непараметрическим, в зависимости от того, каким методом восстанавливаются одномерные плотности.

Например, если были выбраны гауссовские распределения для плотностей $p_i(x_i|y)$, то алгоритм, описанный в формуле (24), можно записать в виде формулы (25). В таком случае алгоритм из формулы (25) будет являться линейным классификатором.

$$a(x) = \arg\max_{y \in Y} \left(\ln \lambda_y \hat{P}(y) - \frac{1}{2} \hat{\mu}_y^{\mathrm{T}} \hat{\Sigma}^{-1} \hat{\mu}_y + x^{\mathrm{T}} \hat{\Sigma}^{-1} \hat{\mu}_y \right), \qquad (25)$$

где $\hat{\mu}_y$ – оценка математического ожидания,

$\hat{\Sigma}^{-1}$ – оценка ковариационной матрицы.

4.3 Метод опорных векторов (*SVM*)

Метод опорных векторов основан на понятии оптимальной разделяющей гиперплоскости [7]. Рассмотрим случай бинарной классификации.

Введем обозначения:

- обучающая выборка $X^l = (x_i, y_i)_{i=1}^l$,
- x_i – объекты, векторы из множества $X = \mathbb{R}^n$,
- y_i – метки классов, элементы множества $Y = \{-1, +1\}$.

Необходимо найти параметры $w \in \mathbb{R}^n, w_0 \in \mathbb{R}$ линейной модели классификации по формуле (26). Данные параметры и будут являться разделяющей гиперплоскостью.

$$a(x; w, w_0) = \mathrm{sign}(\langle x, w \rangle - w_0) \qquad (26)$$

Будем минимизировать эмпирический риск – количество объектов в тестовой выборке, на которых классификатор ошибся согласно формуле (27).

$$\sum_{i=1}^{l} [a(x_i; w, w_0) \neq y_i] = \sum_{i=1}^{l} [M_i(w, w_0) < 0] \to \min_{w, w_0}, \qquad (27)$$

где $M_i(w, w_0) = (\langle x, w \rangle - w_0) y_i$ – отступ объекта x_i.

Так как эмпирический риск – кусочно-постоянная функция, то заменяя ее непрерывной по параметрам оценкой сверху, получим формулу (28).

$$Q(w, w_0) = \sum_{i=1}^{l} [M_i(w, w_0) < 0] \leq \leq \sum_{i=1}^{l} \big(1 - M_i(w, w_0)\big)_+ + \frac{1}{2C} \|w\|^2 \to \min_{w, w_0}, \qquad (28)$$

где C – гиперпараметр регуляризации,

$(x)_+$ – операция положительной срезки,

$\|w\|$ – норма вектора w.

Для случая линейно неразделимой выборки формулу (28) можно переписать в виде задачи математического программирования по формуле (29).

$$\begin{cases} \frac{1}{2C} \|w\|^2 + C \sum_{i=1}^{l} \xi_i \to \min_{w, w_0, \varepsilon} \\ \xi_i \geq 1 - M_i(w, w_0), i = 1, \dots, l \\ \xi_i \geq 0, i = 1, \dots, l \end{cases} \qquad (29)$$

где ξ_i – набор дополнительных переменных, характеризующих величину ошибки. Это позволит алгоритму допускать ошибки на обучающей выборке и работать в случае, если классы линейно неразделимы.

Задаче, описанной формулой (29), эквивалентна задача безусловной минимизации, которую можно описать формулой (30).

$$\frac{1}{2C}\|w\|^2 + C\sum_{i=1}^{l}\left(1 - M_i(w, w_0)\right)_+ \to \min_{w,w_0} \qquad (30)$$

Для существования минимума из формулы (30) необходимо несколько условий, описанных в формуле (31).

$$\begin{gathered} w = \sum_{i=1}^{l} \lambda_i y_i x_i \\ \sum_{i=1}^{l} \lambda_i y_i = 0 \\ \eta_i + \lambda_i = C, i = 1, \ldots, l \end{gathered} \qquad (31)$$

Решая задачу, описанную в формулой (31) получаем решение, описанное формулой (32), где числа λ_i определяются численными методами оптимизации из формулы (31).

$$w = \sum_{i=1}^{l} \lambda_i y_i x_i \qquad (32)$$

$w_0 = (\langle x, w\rangle - y_i)$, для любого i: $\lambda_i > 0$, $M_i = 1$

Тогда классификатор можно будет записать в виде формулы (33).

$$a(x) = \text{sign}\left(\sum_{i=1}^{l} \lambda_i y_i \langle x, x_i\rangle - w_0\right) \qquad (33)$$

У *SVM* существует нелинейное обобщение, в котором скалярное произведение $\langle x, x'\rangle$ заменяется нелинейной функцией $K(x, x')$, которая называется ядром. При такой замене происходит переход из исходного признакового пространства, в котором классы могут быть линейно неразделимы, в другое признаковое пространство, в котором классы уже могут быть линейно разделены (в зависимости от выбранного ядра).

Примеры использования различных видов ядер в *SVM* показаны на рисунке 5.

Рисунок 5. Примеры использования различных ядер в SVM

4.4 *NBSVM*

Алгоритм использует мультиномиальный наивный байесовский классификатор (*MNB*) для получения признакового описания обучающей выборки и обучает на полученных признаках *SVM*, после чего использует интерполяцию, полученного при обучении *SVM* вектора весов, для получения модели *NBSVM* [8]. За основу алгоритма берётся линейный классификатор, описанный в формуле (34).

$$a(x) = \text{sign}(\langle w^T, x\rangle + b) \qquad (34)$$

Пусть решается задача бинарной классификации, где: $f^{(i)}$ – bag-of-words вектор для объекта i,

$y^{(i)} \in \{-1,1\}$ – класс объекта i,

$|V|$ – объём словаря.

Определим вектор r в соответствии с формулой (35).

$$r = \ln \frac{\frac{p}{\|p\|}}{\frac{q}{\|q\|}}, \qquad (35)$$

где $p = \alpha + \sum_{i:y^{(i)}=1} f^{(i)}$ – вектор, где каждый элемент равен количеству раз, когда слово i встретилось в документах класса 1, плюс сглаживающий параметр, $q = \alpha + \sum_{i:y^{(i)}=-1} f^{(i)}$– вектор, где каждый элемент равен количеству раз, когда слово i встретилось в документах класса -1, плюс сглаживающий параметр,

α – сглаживающий параметр (необходим для избегания случаев равенства числителя или знаменателя нулю).

В (*MNB*) вектор весов модели будет равен вектору r. Однако, вместо прямого использования $f^{(i)}$, лучший результат даёт бинаризация признаков $f^{(i)}$, что достигается заменой $f^{(i)}$ на $\hat{f}^{(i)}$ согласно формуле (36).

$$\hat{f}^{(i)} = \begin{cases} 1, f^{(i)} \geq 1 \\ 0, f^{(i)} = 0 \end{cases} \qquad (36)$$

Обучая *SVM* (30), с использованием признаков $r^{(i)}\hat{f}^{(i)}$ можно получить вектор весов w. Сама модель *NBSVM* использует интерполяцию полученного при обучении *SVM* вектора весов с помощью формулы (37).

$$w' = (1 - \beta)\bar{w} + \beta w, \qquad (37)$$

где $\bar{w} = \frac{\|w\|}{|V|}$ – средняя значимость вектора w,

$\beta \in [0,1]$ – параметр интерполяции.

4.5 Бинарное решающее дерево

Бинарное решающее дерево – это алгоритм классификации $a(x)$, задающийся бинарным деревом, в котором каждой внутренней вершине $v \in V$ приписан предикат $\beta_v: X \to \{0,1\}$, а каждой терминальной вершине $v \in V$ приписано имя класса $c_v \in Y$ [7].

При классификации объекта $x \in X$ он проходит по дереву путь от корня до некоторого листа. В листе решающего дерева, как правило, содержатся объекты тренировочной выборки (обычно не сами объекты, а их классы), попавшие туда при обучении, и класс классифицируемого объекта определяется в зависимости от преобладающего класса объектов в данном листе.

Пример части решающего дерева показан на рисунке 6.

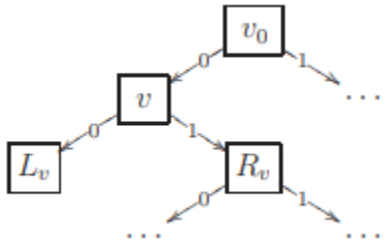

Рисунок 6. Пример части решающего дерева

В основе каждого алгоритма построения решающего дерева лежит жадное добавление следующей вершины. При добавлении новой вершины предикат выбирается таким образом, чтобы максимизировать информативность.

4.6 Градиентный бустинг

Градиентный бустинг – это алгоритм машинного обучения, являющийся линейной (выпуклой) комбинацией базовых алгоритмов [7], что может быть выражено формулой (38).

$$a(x) = \sum_{t=1}^{T} a_t b_t, x \in X, a_t \in \mathbb{R}_+, \quad (38)$$

где T – количество базовых алгоритмов,
a_t – вес базового алгоритма с номером t,
b_t – базовый алгоритм с номером t,
X – множество объектов.

Общая идея бустинга состоит в том, что на каждом шаге построения алгоритма в композицию добавляется такой новый базовый алгоритм b_t с весом a_t так, чтобы минимизировать функционал качества, описанный в формуле (39).

$$Q(a,b;\ X^l) = \\ = \sum_{i=1}^{l} \mathcal{L}\left(\sum_{t=1}^{T-1} a_t b_t(x_i) + ab(x_i), y_i\right) \to \min_{a,b}, \quad (39)$$

где l – количество объектов в обучающей выборке,
$\mathcal{L}$ – функция потерь на одном объекте.

Если взять выражение $f_{k-1,i} = \sum_{t=1}^{k-1} a_t b_t(x_i)$ в качестве текущего приближения на k-ом шаге обучения алгоритма, а $f_{k,i} = \sum_{t=1}^{k} a_t b_t(x_i)$ – в качестве следующего приближения, то можно применить градиентный метод оптимизации к функционалу из формулы (38), тогда компоненты вектора градиента будут иметь вид, описанный в формуле (40).

$$g_i = \mathcal{L}'\left(f_{k-1,i}, y_i\right), \quad (40)$$

Главная идея алгоритма заключается в поиске такого базового алгоритма b_T, чтобы вектор $(b_T(x_i))_{i=1}^{l}$ приближал вектор антиградиента $(-g_i)_{i=1}^{l}$, что может быть выражено в виде формулы (41):

$$b_T := arg\max_b \sum_{i=1}^{l} (b(x_i) + g_i)^2 \quad (41)$$

В качестве базовых алгоритмов в градиентном бустинге, чаще всего, используют решающие деревья небольшой глубины, так как при использовании более «сильных» алгоритмов происходит быстрое переобучение.

4.7 Многослойный персептрон (*MLP*)

Многослойный персептрон представляет собой искусственную нейронную сеть прямого распространения, содержащую входной слой, один или несколько скрытых слоёв и выходной слой [7]. Основным элементом такой сети является нейрон, показанный на рисунке 7 [7].

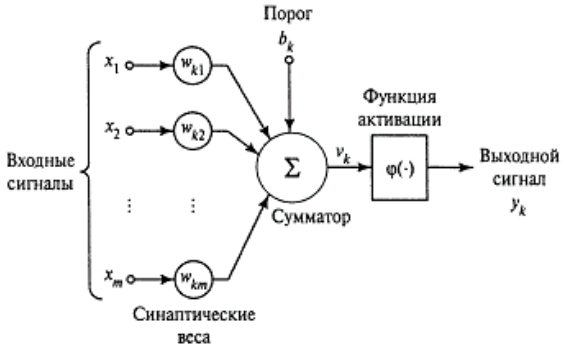

Рисунок 7. Модель нейрона

Описание обозначений на рисунке 7:

$x = (x_i)_{i=1}^m$ – векторное представление объекта,

x_{kj} – веса нейрона,

φ – функция активации.

Многослойный персептрон показан на рисунке 8 [7].

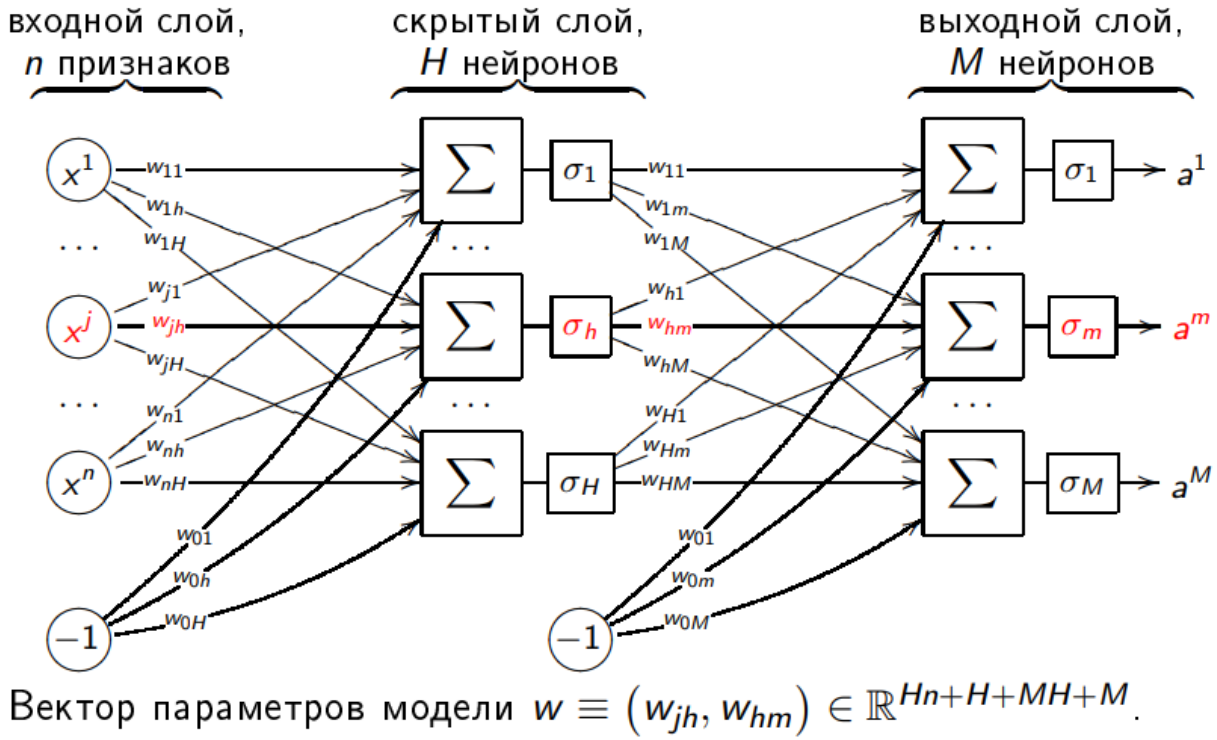

Рисунок 8. Многослойный персептрон

Обучение сети происходит посредством минимизации функционала качества Q (42).

$$Q(w) := \sum_{i=1}^{l} \mathcal{L}(w, x_i, y_i) \to \min_w, \qquad (42)$$

где w – вектор весов нейронной сети,

$\mathcal{L}$ – функция потерь на одном объекте,

x_i – признаковое описание i-ого объекта,

y_i – ответ на i-ом объекте.

Основным методом обучения нейронных сетей является градиентный спуск и его адаптивные варианты, такие как: *Adagrad*, *Adadelta*, *RMSprop*, *Adam*.

4.8 Рекуррентная нейронная сеть (*RNN*)

Рекуррентные нейронные сети предназначены для решения задач, связанных с обработкой последовательностей, а так как текст является последовательностью символов, то данные сети часто применяются при решении задач, связанных с обработкой текстов [9].

В отличие от многослойных персептронов в рекуррентных нейронных сетях связи между нейронами могут идти не только от нижнего слоя к верхнему, но и от нейрона к «самому себе», точнее, к предыдущему значений самого этого нейрона или других нейронов того же слоя.

Архитектура «простой» RNN представлена на рисунке 9 [9].

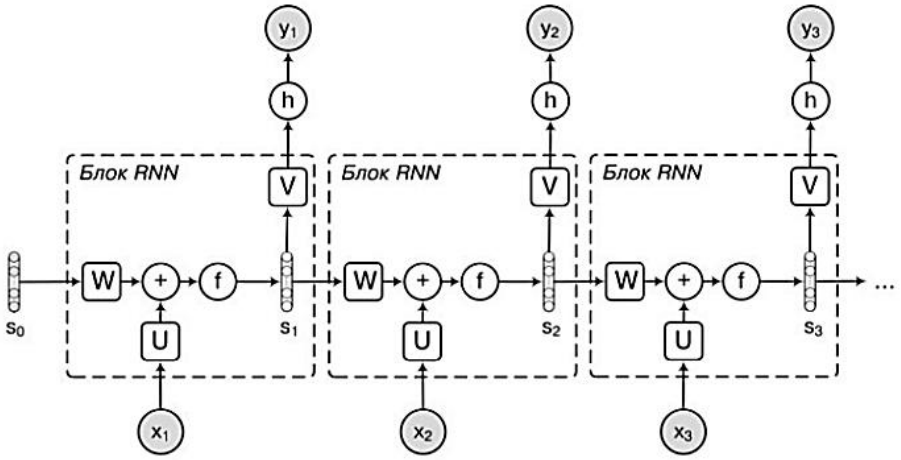

Рисунок 9. Архитектура «простой» RNN

Описание обозначений элементов рисунка 9:

- W – матрица весов для перехода между скрытыми состояниями,
- U – матрица весов для входов,
- V – матрица весов для выходов,
- s_i – скрытое состояние на шаге i,
- x_i – вход сети на шаге i,
- y_i – выход сети на шаге i,
- f и h – функции активации.

Из рисунка 9 можно получить формальное описание сети в момент времени t (43).

$$a_t = b + Ws_{t-1} + Ux_t, s_t = f(a_t)$$
$$o_t = c + Vs_t, y_t = h(o_t) \quad (43)$$

Часто вместо «обычной» *RNN* используют двунаправленные *RNN* – две *RNN* одна из которых читает слова текста слева направо, а другая – справа налево. При обучении такой сети используются скрытые состояния обеих сетей (обычно происходит конкатенация векторов, описывающих скрытые состояния). Мотивация использования двунаправленных *RNN* состоит в том, чтобы получить состояние, отражающее контекст и слева, и справа для каждого элемента последовательности.

4.9 *LSTM*

«Обычные» *RNN* очень плохо справляются с ситуациями, когда нужно что-то запомнить «надолго»: влияние скрытого состояния или входа с шага t на последующие состояния *RNN* экспоненциально затухает. Для решения данной проблемы конструкцию нейрона сети усложняют, моделируя в том или ином виде «долгую память». Одной из таких конструкций является *LSTM* (*Long Short-Term Memory*) [10]. Стандартная архитектура *LSTM*-ячейки показана на рисунке 10 [10]. В *LSTM* есть три основных вида узлов, которые называются гейтами: входной (*input gate*), забывающий (*forget gate*) и выходной (*output gate*), а также собственно рекуррентная ячейка со скрытым состоянием.

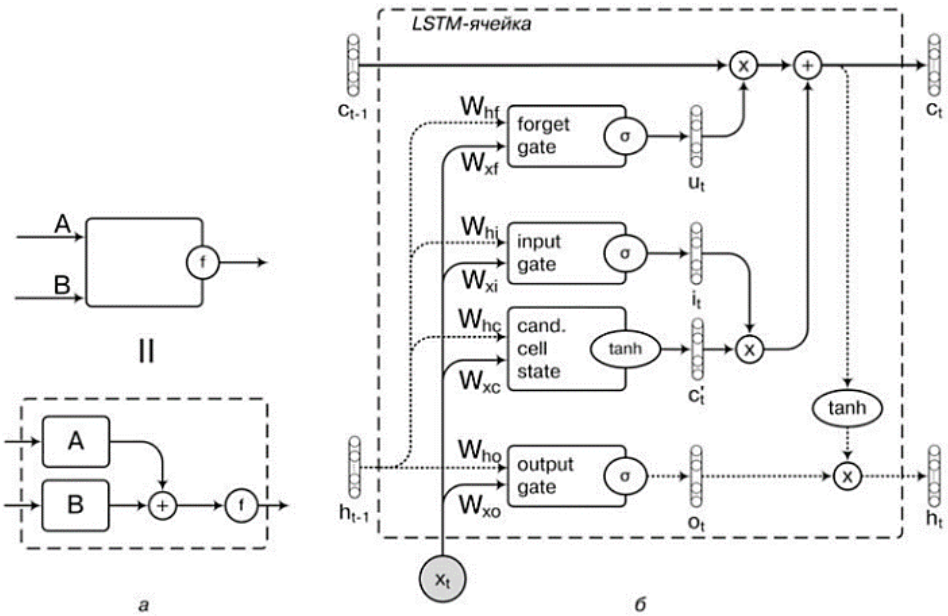

Рисунок 10. LSTM: а – обозначение гейта с двумя входами; б – структура LSTM-ячейки

Описание обозначений элементов рисунка 10:

- x_t – входной вектор во время *t*,
- h_t – вектор скрытого состояния во время *t*,
- c_t – вектор состояния ячейки во время *t*,
- W_i (с различными вторыми индексами) – матрицы весов, применяемые ко входу,
- W_h (с различными вторыми индексами) – матрицы весов, в различных рекуррентных соединениях,
- b (с различными индексами) – векторы свободных членов.

Формальное описание сети в момент времени *t* представлено формулами (44) – (49).

Кандидат в состояние ячейки c_t' выражается формулой (44).

$$c_t' = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_{c'}) \quad (44)$$

Входной гейт i_t выражается формулой (45).

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (45)$$

Забывающий гейт f_t выражается формулой (46).

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right) \quad (46)$$

Выходной гейт o_t выражается формулой (47).

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (47)$$

Состояние ячейки c_t выражается формулой (48).

$$c_t = f_t \odot c_{t-1} + i_t \odot c_t' \quad (48)$$

Скрытое состояние h_t выражается формулой (49).

$$h_t = o_t \odot tanh(c_t) \quad (49)$$

Если сравнивать с ячейкой «обычной» *RNN*, *LSTM*-ячейка имеет более сложную структуру – у *LSTM*-ячейки по 4 матрицы *W* и *U*, а также имеется вектор состояния ячейки c_t, который выполняет функцию памяти. Скрытые состояния в *LSTM* могут, если сама ячейка не решит их перезаписать, сохранять свои значения неограниченно долго, что решает проблему «исчезающих градиентов»: независимо от матрицы рекуррентных весов ошибка сама собой затухать не будет.

4.10 GRU

LSTM требует довольно значительных ресурсов (в *LSTM*-ячейке гораздо больше матриц, чем в ячейке «обычной» *RNN*), поэтому после исследований *LSTM*, направленных на уменьшение количества обучаемых параметров при сохранении эффекта долгосрочной памяти была создана модель *GRU* (*gated recurrent unit*) [11]. Было выяснено, что критически важными компонентами для успешной работы *LSTM* являются два гейта: выходной и забывающий. В архитектуре *GRU* используется идея совмещения выходного и забывающего гейта, а скрытое состояние h_t совмещено со значением памяти c_t. Архитектура *GRU*-ячейки показана на рисунке 11 [11].

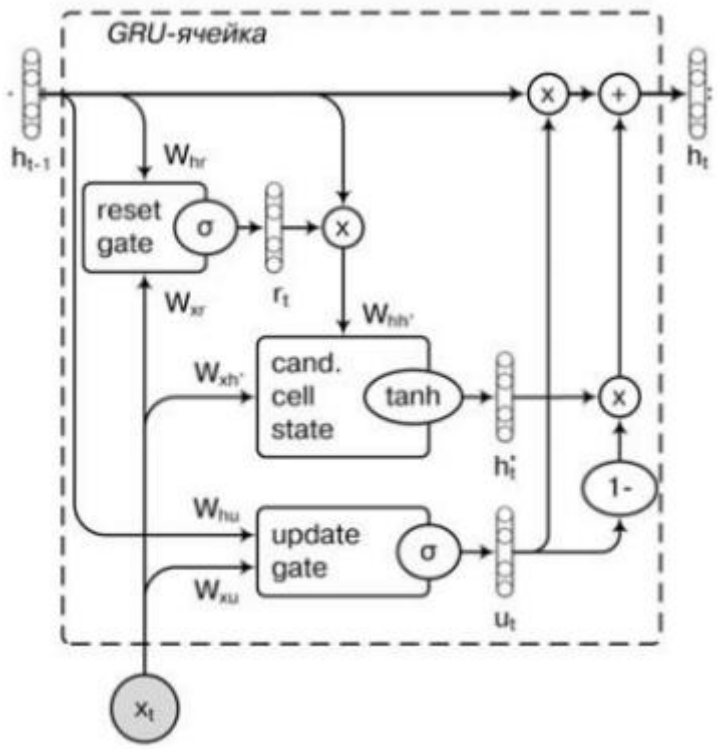

Рисунок 11. Архитектура GRU-ячейки

Описание обозначений элементов рисунка 11:

- u_t – гейт обновления (update gate), который и является комбинацией выходного и забывающего гейта,
- r_t – гейт перезагрузки,
- h_t – скрытое состояние в момент времени *t*.

Принцип работы *GRU* описан формулами (50) – (53).

Гейт обновления u_t выражается формулой (50).

$$u_t = \sigma(W_{xu}x_t + W_{hu}h_{t-1} + b_u) \quad (50)$$

Гейт перезагрузки r_t выражается формулой (51).

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \quad (51)$$

Кандидат в скрытое состояние h'_t выражается формулой (52).

$$h'_t = tanh\left(W_{xh'}x_t + W_{hh'}(r_t \odot h_{t-1})\right) \quad (52)$$

Скрытое состояние h_t выражается формулой (53).

$$h_t = (1 - u_t) \odot h'_t + u_t \odot h_{t-1} \quad (53)$$

Основная разница между *GRU* и *LSTM* заключается в том, что *GRU* пытается сделать двумя гейтами то же самое, что *LSTM* делает тремя. Обязанности забывающего гейта f_t в *LSTM* здесь разделены между двумя гейтам: r_t и u_t. Кроме того, не возникает второй нелинейности на пути от входа к выходу, как в случае с *LSTM*.

4.11 Сверточная нейронная сеть (*CNN*)

Сверточные нейронные сети (*convolutional neural networks, CNN*) – это весьма широкий класс архитектур, основная идея которых состоит в том, чтобы переиспользовать одни и те же части нейронной сети для работы с разными маленькими, локальными участками входов [9]. Изначально, сверточные сети были придуманы для решения задач, связанных с обработкой изображений, для чего в них применяются двумерные свертки. Для решения задач, связанных с обработкой последовательностей, вместо двумерных сверток используются одномерные. Основная идея сверточных сетей в том, чтобы дать сети обучиться выявлению признаков во входных данных с помощью сверточных слоев, а затем, подать вычисленные признаки на вход полносвязной сети.

Сверточная сеть для обработки текстов показана на рисунке 12.

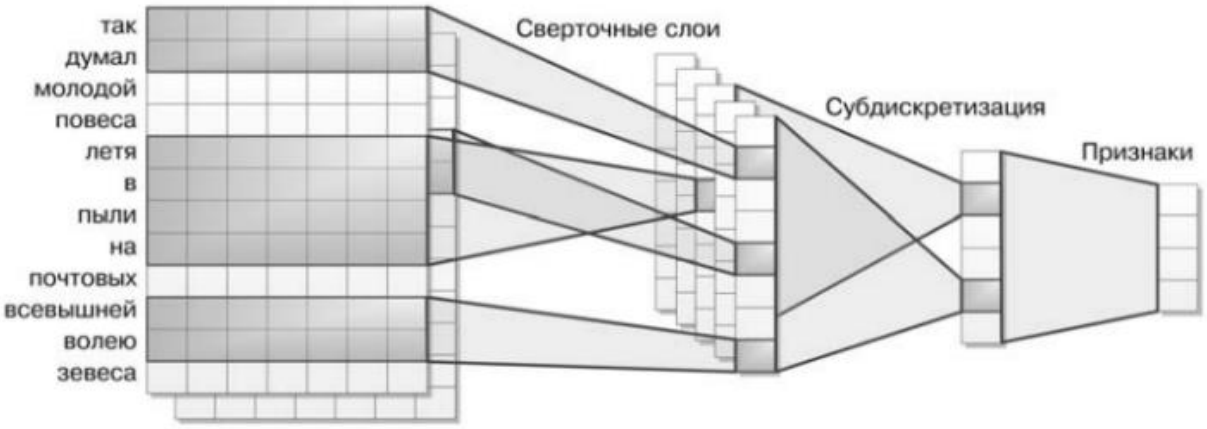

Рисунок 12. Одномерная сверточная сеть для обработки текстов

4.12 Трансформер

Многие архитектуры, использующие модель внимания, применяют ее вместе с рекуррентными слоями. В отличие от таких моделей архитектура трансформер полагается только на механизм внимания [12].

Self-attention – это механизм внимания, связывающий различные позиции одной последовательности для вычисления представления последовательности.

Трансформер использует архитектуру кодер-декодер [9]. Кодер отображают входную последовательность символов $(x_1, \dots, x_n)$ в последовательность вещественных значений $z = (z_1, \dots, z_n)$, а декодер берёт последовательность z и создаёт выходную последовательность символов $(y_1, \dots, y_m)$. На каждом шаге модель является авторегрессионной, потребляя ранее сгенерированные символы в качестве дополнительных входных данных при генерации следующего.

Трансформатор следует этой общей архитектуре, используя сложенные слои self-attention и точечные, полносвязные слои как для кодера, так и для декодера, показанные в левой и правой частях рисунка 13 соответственно [12].

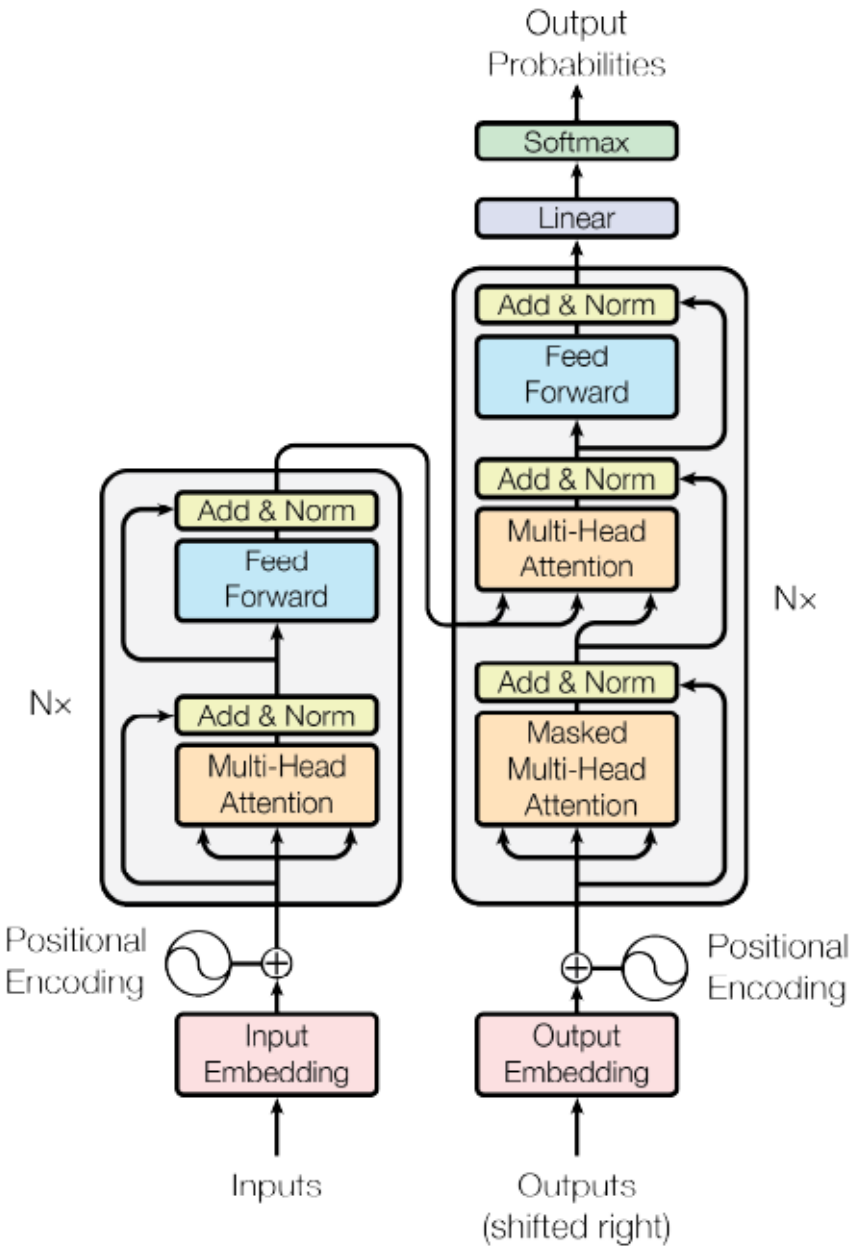

Рисунок 13. Архитектура трансформер

Кодер состоит из стека *N*=6 одинаковых слоёв. Каждый слой состоит из двух подслоёв. Первый подслой – *Multi-Head self-attention*, а второй – полносвязная нейронная сеть (*Feed Forward*). Вокруг каждого из двух подслоёв используется остаточное соединение с последующей нормализацией слоя. Выход каждого подслоя выражается формулой (54). Чтобы упростить эти остаточные соединения, все подслои в модели, а также слои *embedding* (*input* и *output*), производят выходы размерности $d_{model} = 512$.

$$LayerNorm(x + Sublayer(x)), \qquad (54)$$

где $Sublayer(x)$ – функция, реализуемая самим подслоем.

Декодер также состоит из стека *N*=6 одинаковых слоёв. В дополнение к двум подслоям в каждом слое кодера, декодер добавляет третий подслой, который производит *Multi-Head attention* над выходом стека кодера. Также, как и в кодере, каждый подслой использует остаточное соединение вместе с последующей нормализацией слоя. Для того, чтобы предотвратить обращение внимания позиций к последующим позициям, подслой *self-attention* несколько модифицируется путем добавления маски. Эта маска, в сочетании с тем фактом, что выходные *embedding* смещены на одну позицию, гарантирует, что предсказания для позиции i могут зависеть только от известных выходов в позициях меньше i.

Функция внимания может быть описана как отображение запроса и набора пар ключ-значение на выход, где запрос, ключи, значения и выход являются векторами. Выходные данные вычисляются как взвешенная сумма значений, где вес, присвоенный каждому значению, вычисляется функцией совместимости запроса с соответствующим ключом.

В архитектуре трансформер используется собственный слой внимания, названный *Scaled Dot-Product Attention*, который показан на рисунке 14. На вход подаются запросы и ключи размерности d_k, а также значения размерности d_v. Сам слой делает следующие последовательные операции: вычисляет скалярное произведение между запросом и ключами, делит получившиеся значения на $\sqrt{d_k}$ и применяет функцию *softmax* к результату.

На практике, составляется матрица Q из всех запросов. Ключи и значения, также упаковываются в матрицы K и V соответственно, а затем вычисляется матрица выходов по формуле (55).

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (55)$$

Слой *Multi-Head Attention* показан на рисунке 14 [12]. *Multi-Head Attention* позволяет модели совместно воспринимать информацию из разных подпространств представления в разных положениях, описанных формулой (56).

$$MultiHead(Q,K,V) = Concat(head_1,\dots,head_h)W^O, \qquad (56)$$

где $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$,

$W_i^Q \in \mathbb{R}^{d_{model}\times d_k}, W_i^K \in \mathbb{R}^{d_{model}\times d_k}, W_i^V \in \mathbb{R}^{d_{model}\times d_v}, W^O \in \mathbb{R}^{hd_v\times d_{model}}$ – матрицы линейных преобразований,

h – количество параллельных уровней внимания.

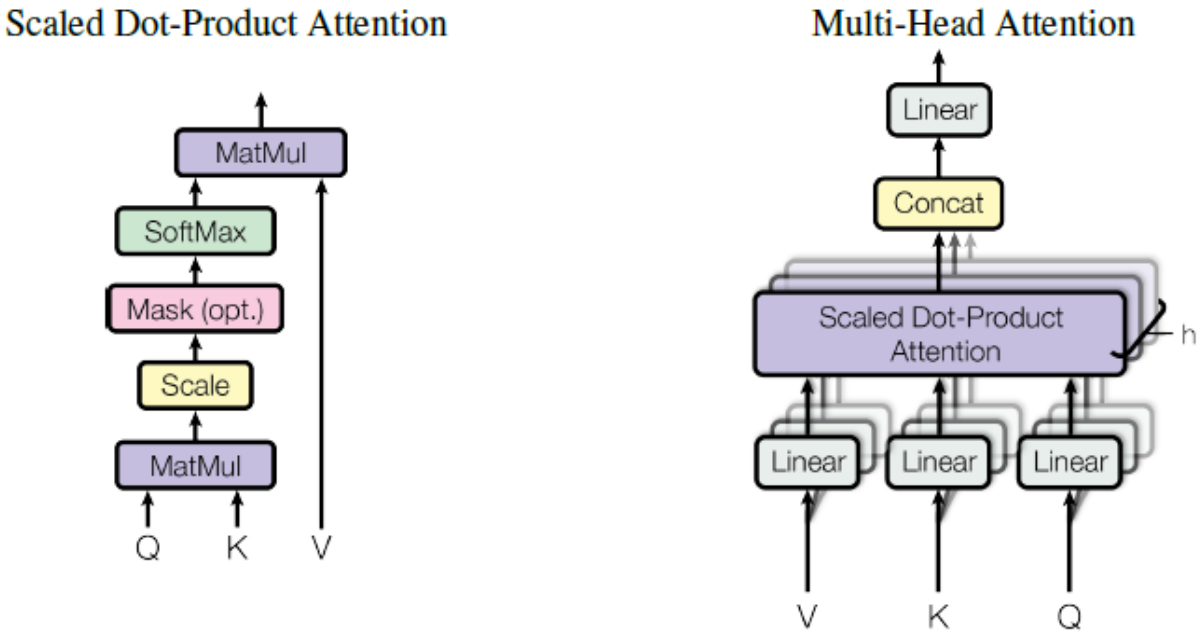

Рисунок 14. Scaled Dot-Product Attention (слева) и Multi-Head Attention (справа)

В дополнение к подслоям внимания, каждый из слоев в кодере и декодере содержит полносвязанную сеть прямого распространения (*FFN*), которая применяется к каждой позиции отдельно и идентично. *FFN* состоит из двух линейных преобразований с активационной функцией *ReLU* между ними, что описано в формуле (57) с размерностями входного и выходного векторов $d_{model} = 512$. Размерность скрытого слоя *FFN* равна $d_{ff} = 2048$.

$$FFN(x) = \max(0, xW_1 + b_1)\,W_2 + b_2 \qquad (57)$$

Подобно другим моделям в трансформере используются обученные слои *embedding* для преобразования входных и выходных токенов в векторы размерности d_{model}. Также используется обычное обученное линейное преобразование и функция *softmax* для преобразования выходных данных декодера в предсказанные вероятности следующего токена. Оба слоя *embedding* используют одну и ту же весовую матрицу.

Поскольку модель не содержит ни рекуррентных, ни сверточных слоев, то, для того чтобы модель использовала информацию о порядке символов, в последовательности используются «позиционные кодировщики» (*positional encoding*).

4.13 *BERT*

Существует два метода использования предобученных моделей: извлечение признаков (*feature-based*) и дообучение (*fine-tuning*). Алгоритм *BERT* (*Bidirectional Encoder Representations from Transformers*) предлагает улучшенный подход на основе дообучения [13]. *BERT* представляет собой многоуровневый двунаправленный кодер на архитектуре трансформер.

BERT использует *MLM* (*masked language model*) – в процессе обучения случайные входные токены маскируются, таким образом маскированное слово предсказывается только исходя из его контекста. В отличие от других способов *MLM* позволяет обучить глубокую двунаправленную сеть на основе архитектуры трансформер (рисунок 15).

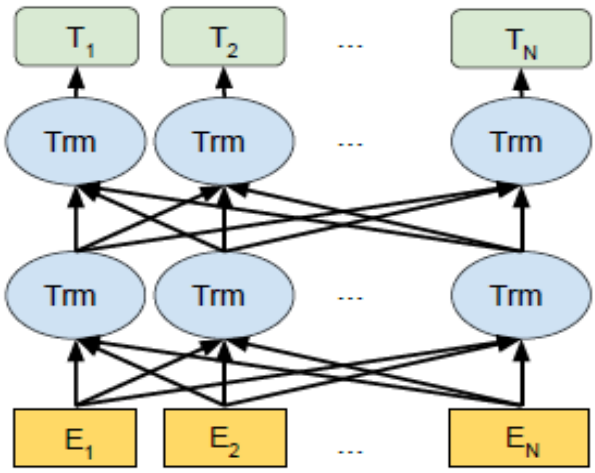

Рисунок 15. Архитектура BERT

Процесс замены токена на маскировочный токен [*MASK*] выглядит следующим образом: случайно выбираются 15% позиций из тренировочных данных, затем, если *i*-ый токен был выбран, то он заменяется токеном [*MASK*] в 80% случаев, в 10% случаев он заменяется случайным токеном и в оставшихся 10% случаев он не меняется.

BERT работает за два шага: предобучение (*pre-training*) и дообучение (*fine-tuning*). В процессе предобучения *BERT* обучается без учителя (на неразмеченных данных) на различных задачах. Каждая последующая задача имеет свою модель для дообучения (входные и выходные слои). *BERT* предобучается на двух корпусах: *BooksCorpus* (800M слов) и английской Википедии (2,500M слов). Процесс дообучения *BERT* для решения конкретной задачи выглядит следующим образом: к предобученной модели *BERT* подсоединяются зависящие от задачи входные и выходные слои, после чего модель дообучается.

Пример модели *BERT* для задачи классификации показан на рисунке 16.

BERT может использоваться для решения различных задач *NLP*. Для этого входная последовательность токенов преобразуется следующим образом: в начало последовательности добавляется токен [*CLS*], обозначающий класс текста, а между предложениями текста добавляется токен [*SEP*].

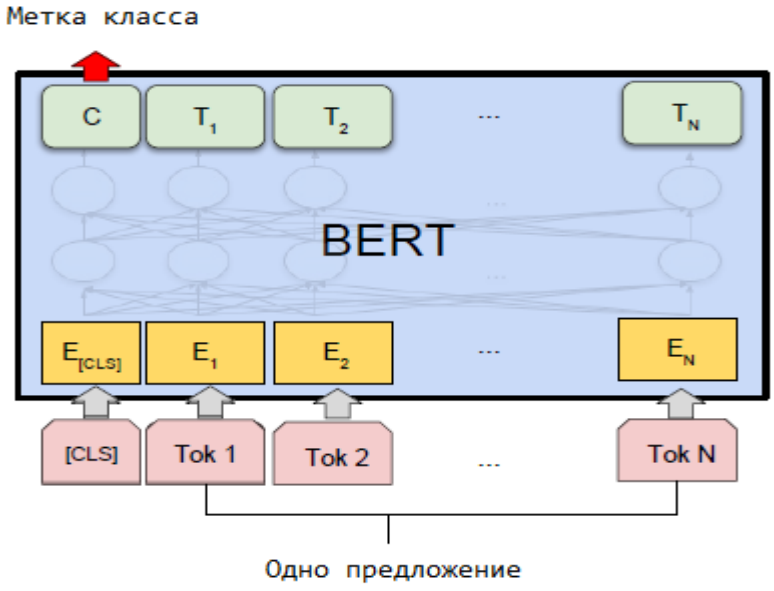

Рисунок 16. BERT для решения задачи классификации

Заключение

В ходе анализа процесса классификации текста рассмотрены основные шаги по классификации текста: предобработка текста, извлечение признаков и классификация текста.

Можно разделить рассмотренные в статье алгоритмы классификации на два класса:

- «универсальные» алгоритмы: наивный байесовский классификатор, логистическая регрессия, метод опорных векторов, градиентный бустинг, многослойный персептрон;
- «специальные» алгоритмы: свёрточная нейронная сеть, рекуррентная нейронная сеть и сеть, основанная на трансформерах.

«Специальные» алгоритмы отличаются от «универсальных» тем, что могут решать задачи с входными данными определённого вида (в данном случае с данными, представляющими собой последовательности).

Список литературы:

1. Kowsari K., Meimandi K. Jafari., Heidarysafa M. Text Classification Algorithms: A Survey. – URL: https://arxiv.org/abs/1904.08067v5.
2. Mikolov T., Chen K., Corrado G. Efficient estimation of word representations in vector space. – URL: https://arxiv.org/abs/1301.3781.
3. Mikolov T., Sutskever I., Chen K. Distributed representations of words and phrases and their compositionality. – URL: https://arxiv.org/abs/1310.4546.
4. Le V. Quoc, Mikolov T. Distributed Representations of Sentences and Documents. – URL: https://arxiv.org/abs/1405.4053.
5. Pennington J., Socher R., Manning D. Christopher. GloVe: Global Vectors for Word Representation. – URL: https://nlp.stanford.edu/pubs/glove.pdf.
6. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning, 2nd edition. — Springer, 2009. — P. 533.
7. Машинное обучение (курс лекций, К.В.Воронцов). – URL: http://www.machinelearning.ru/wiki/index.php?title=Mo.
8. Sida Wang, Christopher D. Manning. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. – URL: https://nlp.stanford.edu/pubs/sidaw12_simple_sentiment.pdf.
9. Николенко С. Глубокое обучение. Погружение в мир нейронных сетей. – СПб: Питер, 2018. – С. 480.
10. Levy O., Lee K., FitzGerald N. Long Short-Term Memory as a Dynamically Computed Element-wise Weighted Sum. – URL: https://arxiv.org/abs/1805.03716.
11. Cho K., Merrienboer B., Gulcehre C. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. – URL: https://arxiv.org/abs/1406.1078.
12. Vaswani A., Shazeer N., Parmar N. Attention Is All You Need. – URL: https://arxiv.org/abs/1706.03762v5.
13. Delvin J., Chang, Lee K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. – URL: https://arxiv.org/abs/1810.04805.