МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

ТЕМАТИЧЕСКАЯ КЛАССИФИКАЦИЯ ТЕКСТОВ С ПОМОЩЬЮ ЛОГИСТИЧЕСКОЙ РЕГРЕССИИ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Савчук Александра Михайловича

Научный руководитель
к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ А. С. Иванов

Саратов 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Теоретическая часть 4
1.1 Машинное обучение 4
1.2 Постановка задачи 4
1.3 Метрики для выбора наилучшей модели 4
1.4 Методы векторизации текста 5
1.5 Логистическая регрессия 7
2 Практическая часть 9
2.1 Модель логистической регрессии 9
2.2 Web-приложение для демонстрации работы классификатора 12
ЗАКЛЮЧЕНИЕ 14
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 15

ВВЕДЕНИЕ

Тематическая классификация текстов - задача компьютерной лингвистики, которая заключается в применении алгоритма машинного обучения (или иного алгоритма) к тексту (сообщению электронной почты, социальной сети) для определения его принадлежности к одному из классов определенной тематики [1]. Классы документов могут быть обозначены различным способом: по эмоциональной составляющей текста (позитивный и негативный комментарий), по «полезности» сообщения (спам, реклама, важное сообщение), по теме сообщения (продажа автомобиля, погода, фильм). Классификация текстов применяется для автоматического определения тематики сайта, определение языка текста, борьбы со спамом, показа более релевантной рекламы.

Цель данной работы - построение модели для тематической классификации сообщений электронной почты на несколько классов и разработка web-приложения для демонстрации работы классификатора.

Тематическая классификация сайтов - задача, становящаяся все более актуальной в связи с постоянно растущим объемом информации в интернете и потребностью в ней ориентироваться. При использовании электронной почты возникает проблема разделения сообщений на важные сообщения, рекламу, спам и др. Интернет-компании активно используют и развивают технологии машинного обучения для классификации сообщений. Например, компания Google использует классификацию сообщений электронной почты в продукте GMail.

Данная работа состоит теоретической и практической частей. В первой главе дано описание способа векторизации текста и логистической регрессии. Также даны определения основных терминов, необходимых для разработки классификатора. Во второй части дано описание построения классификатора и web-приложения с помощью библиотек языка программирования Python. При построении модели машинного обучения для векторизации текста используется метод TF-IDF, который позволяет более точно определять вес токена (слога, слова, словосочетания) в документе. В качестве классификатора выбрана модель логистической регрессии. Выбор обусловлен простотой реализации и высокой скоростью работы алгоритма.

1 Теоретическая часть

1.1 Машинное обучение

Идея обучающихся машин (learning machines) принадлежит А. М. Тьюрингу [2]. Машинное обучение (Machine Learning) - обширный подраздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться. Идея в том, чтобы не программировать алгоритм решения задачи вручную, а «выучить» его из данных. Различают два типа обучения. Обучение по прецедентам, или индуктивное обучение, основано на выявлении общих закономерностей по частным эмпирическим данным. Дедуктивное обучение предполагает формализацию знаний экспертов и их перенос в компьютер в виде базы знаний. Дедуктивное обучение принято относить к области экспертных систем, поэтому термины машинное обучение и обучение по прецедентам можно считать синонимами. Существуют также и другие определения. Машинное обучение - процесс, в результате которого машина (компьютер) способна показывать поведение, которое в нее не было явно заложено (запрограммировано) [3]. Говорят, что компьютерная программа обучается на основе опыта E по отношению к некоторому классу задач T и меры качества P, если качество решения задач из T, измеренное на основе P, улучшается с приобретением опыта E [4].

1.2 Постановка задачи

Пусть имеется множество категорий (классов, меток) $C = \{c_1, \dots c_k\}$ и множество документов $D = \{d_1, \dots d_n\}$. Также существует неизвестная целевая функция $\Phi : C \times D \to \{0, 1\}$. Необходимо построить классификатор Φ' максимально близкий к Φ. Близость определяется функцией потерь на тестовой выборке, в данном случае в качестве функции потерь используется категориальная кросс-энтропия $CE(\hat{y}, y) = -\sum_{i=1}^{k} y_i * log(\hat{y}_i)$. Имеется некоторая начальная коллекция размеченных документов $R \subset C \times D$, для которых известны значения Φ. Данная выборка разделена на «обучающую» и «тестовую» подвыборку. Первая используется для обучения классификатора, вторая - для независимой проверки качества его работы.

1.3 Метрики для выбора наилучшей модели

Для оценки качества примененного алгоритма и сравнения разных алгоритмов на одной задаче существуют метрики (функции), позволяющие опреде-

лить насколько хорошо алгоритм справляется со своей задачей. Как показано в [5] основными метриками для оценки качества решения задачи классификации являются доля правильных ответов (accuracy), точность (precision), полнота (recall) и F-мера.

Самой простой и интуитивно понятной метрикой является доля правильных ответов: $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$. Однако, данная метрика плохо подходит под задачу с классами, сильно отличающимся по мощности. Например, необходимо оценить модель фильтрации спама. Пусть дано 100 писем, которые не являются спамом. Из них алгоритм распознал 90 писем верно (TN) и 10 писем как спам (FP). И дано 10 писем, которые являются спамом. Из них алгоритм распознал верно 5 писем (5 TP и 5 FN). Таким образом, accuracy составляет: $accuracy = \frac{5+90}{5+90+10+5} = 86,4\%$. Однако, если бы алгоритм предсказывал все письма как не являющиеся спамом, то: $accuracy = \frac{0+100}{0+100+0+10} = 90,9\%$.

1.4 Методы векторизации текста

Как показано в [6] основными методами извлечения признаков из текста являются:

- двоичных вектор, описывающий встречаемость слов в документе;
- вектор вещественных чисел, описывающий встречаемость слов в тексте с учетом их частотности;
- N-граммы символов и токенов, словосочетания;
- плотные векторные представления слов, предложения и текстов (word embeddings, doc embeddings);
- ядерные методы (kernel methods) и графовые ядра.

Чем чаще слово встречается в документе, тем оно более характерно для этого документа, тем лучше описывает его тематику. С другой стороны, чем реже это слово встречается в корпусе (выборке документов), тем оно более специфично и информативно. За этот баланс отвечают две величины: TF и IDF. TF (term frequency) - это частота слова в документе (значимость слова в документе):

$$TF(w,d) = \frac{WordCount(w,d)}{Count(d)}, \text{ где} \tag{1}$$

$WordCount(w, d)$ - количество словоупотреблений слова w в документе d,
$Count(d)$ - длина документе d (количество слов в документе).

IDF (inverse document frequency) - обратная частота слова в документах:

$$IDF(w, c) = \frac{Size(c)}{DocCount(w, c)}, \text{ где} \tag{2}$$

$Size(c)$ - размер коллекции c в документах,
$DocCount(w, c)$ - количество документов в коллекции c, в которых встречается слово w.

В данной формуле размер коллекции делится на количество документов, в которых употребляется заданное слово. Таким образом, наибольший вес будет иметь слово, которое встречается в одном документе. Итоговый вес слова можно посчитать как произведение величин TF и IDF:

$$TDIDF(w, d, c) = TF(w, d) * IDF(w, c) \tag{3}$$

На практике, величину TF часто логарифмируют следующим образом:

$$TF(w, d) = log\left(\frac{WordCount(w, d)}{Count(d)} + 1\right) \tag{4}$$

Данное преобразование позволяет сделать распределение весов слов менее разбросанным, т.е. уменьшить его дисперсию.

Иногда выгоднее считать веса не слов, а N-граммы, то есть последовательности подряд идущих символов или токенов. N-граммы бывают символьными, а бывают пословными. Символьные N-граммы содержат три или более подряд идущих символа. А в пословных N-граммах, соответственно, несколько подряд идущих слов. Использование N-грамм не сильно усложняет модель. Кроме того, модель получается более устойчивой к опечаткам, а также к словоизменению, т. е. можно до определенной степени обойтись без исправления опечаток, а также без сложных алгоритмов нормализации текста - например, лемматизации. Пословные N-граммы более специфичны по сравнению с отдельными словами, т. е. они встречаются реже, но при этом (если они

встречаются) являются более сильным фактором. И поэтому они могут лучше описывать особенности тематики текстов. Недостатки N-грамм следуют из их преимуществ. Размерность пространства растет очень быстро, а вектора получаются очень разреженными. Чем больше N, тем реже соответствующая N-грамма встречается. Другим недостатком N-грамм является то, что близкие по смыслу слова («чашка» и «кружка») кодируются не зависящими друг от друга элементами вектора, поэтому обобщающая способность может быть невысока. Также размерность вектора достаточно большая, следовательно на небольшой выборке высока вероятность переобучения.

К прикладным задачам обработки естественных языков относятся те задачи, которые можно положить в основу продукта. Одна из распространенных задач - классификация. Существуют следующие виды задачи классификации: тематическая классификация длинных текстов и классификация коротких текстов (по тональности, интенции). Для обучения модели в случае тематической классификации используется размеченные тексты, т. е. тексты, над каждым из которых проставлена метка, относящая текст к одной из категории. Длина текста должна позволять набрать определенную статистику (каждый текст должен состоять хотя бы из нескольких предложений). Метки определяются составом текста в целом, а не отдельными фразами и их структурой. Наиболее подходящий алгоритм для этой задачи - линейный классификатор с разреженными признаками, взвешенными по частоте (например, TF-IDF). Нейронные сети также подходят для данной задачи, однако они не дают большого прироста в качестве модели.

1.5 Логистическая регрессия

Метод логистической регрессии можно использовать для нескольких классов документов, когда нужно предсказывать одну метку из нескольких классов:

$$Classifier\colon \mathbb{R}^d \to \{1, \ldots, c\}, \text{ где} \tag{5}$$

c - количество различных классов, $c > 2$

Метод многоклассовой логистической регресии определяется следующей формулой:

$$\hat{\vec{y}} = softmax(W * \vec{x}), \text{ где} \tag{6}$$

$\hat{\vec{y}} \in \mathbb{R}^k, \sum_{i=1}^{k} \hat{y}_i = 1$ - вектор вероятностей отнесения объекта к определенному классу,

$W \in \mathbb{R}^{k \times d}$ - матрица весов,

$\vec{x} \in \mathbb{R}^d$ - вектор признаков,

$softmax(\vec{z}) = \left\{ \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \right\}_i$ - функция активации $softmax$

В данном случае, W является не вектором весов, а прямоугольной матрицей. Количество строк в этой матрице соответствует количеству классов, а количество столбцов - количеству входных признаков. Результатом применения модели является не одно число (как в случае с бинарной логистической регрессией), а вектор. Этот вектор описывает распределение вероятностей принадлежности объекта x к одному из c классов. Чтобы получить в результате распределение вероятностей необходима функция активации $softmax$. Это вектор-функция, преобразующая вектор вещественных чисел из произвольного отрезка в вектор чисел в отрезке от 0 до 1, сумма которых всегда равна единице. Таким образом, значение этой функции всегда удовлетворяет определению распределения вероятностей. Функция потерь в многоклассовой логистической регресии определяется следующей формулой:

$$CE(\hat{y}, y) = -\sum_{i=1}^{c} y_i * log(\hat{y}_i) \tag{7}$$

В качестве градиентного метода для минимизации функции потерь был выбран современный метод SAGA. Данный метод создан исследователями Aaron Defazio (Австралия), Francis Bach (Франция) и Simon Lacoste-Julien (Франция) в 2014 году. Метод основан на методах SAG, SDCA, MISO и SVRG - наборе недавно открытых инкрементальных градиентных алгоритмах с быстрой линейной скоростью сходимости. Подробное описание метода изложено в [7].

2 Практическая часть

2.1 Модель логистической регрессии

В практической части показана работа многоклассовой логистической регрессии на примере датасета для тематической классификации, состоящего из 18 тысяч сообщений электронной почты, распределенным по 20 категориям. Данный датасет был взят из библиотеки scikit-learn [8]. Векторизация текста и метод логистической регрессии реализован с использованием библиотеки scikit-learn.

Реализация метода логистической регрессии состоит из векторизации и, собственно, алгоритма логистической регрессии. В качестве среды разработки используется инструмент Jupyter Notebook из установленного набора пакетов Anaconda. Данный набор пакетов содержит множество библиотек для машинного обучения и анализа данных.

После загрузки датасета выводится информация о его размер:

```
from sklearn.datasets import fetch_20newsgroups

dataset_bunch = fetch_20newsgroups(subset='all')
print('Размер датасета равен', len(dataset_bunch.data))
print()
print('Список всех классов в датасете', dataset_bunch.target_names)
print()
```

Данный код выводит размер датасета:

```
Размер датасета равен 18846
```

Также данный код выводит список всех классов с их именами:

```
Список всех классов в датасете ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.
    misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.
    forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', '
    sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', '
    talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.
    misc']
```

С помощью функции train_test_split из библиотеки scikit-learn определяется обучающая и тестовая выборка, а также их размер:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(dataset_bunch.data, dataset_bunch.
    target, test_size =TEST_SIZE)
```

```
print('Длина train датасета ', len(x_train))
print('Длина test датасета ', len(x_test))
```

Данные выборки согласно приведенному коду имеют следующий размер:

```
Длина train датасета 13192
Длина test датасета 5654
```

Следующим шагом является преобразование сырых текстов в признаки. В результате подготовки признаков получится две прямоугольные матрицы (для обучающей и тестовой выборок), строки которых соответствуют текстам, а столбцы - признакам. Сначала был выполнен препроцессинг текста. В результате препроцессинга была осуществлена очистка текста с помощью регулярных выражений, т. е. были удалены ссылки, адреса электронной почты, хэш-теги и последовательности, которые не являются словами. Также удалены слова, длины которых выходят за диапазон, заданный константами min_word_length и max_word_length. Кроме этого в ходе препроцессинга произведена нормализация текста - в зависимости от значения константы normalize_method использовалась либо лемматизация, либо стемминг.

Непосредственно для векторизации используется класс TfidfVectorizer из библиотеки scikit-learn. Для препроцессинга текста используется определенная выше функция. Один из основных шагов почти для всех задач обработки текстов - это токенизация. Токенизация - это разбиение исходного текста на базовые лексические элементы (токены). В данном случае в качестве токенизации используется разбиение на слова (используется параметр analyzer='word'). Из результата токенизации удаляются слова, которые присутствуют в списке стоп-слов (используется список стоп-слов из библиотеки NLTK) или используются в слишком большом или малом количестве документов (за это отвечают константы min_doc_freq и max_doc_freq). Словарь корпуса был построен для N-грамм длиной от 1 до 3 токенов:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords

stop_words=stopwords.words('english')
print(stop_words[:10])
vectorizer = TfidfVectorizer(analyzer='word', stop_words=stop_words, \
                             ngram_range=(1, 3), min_df=min_doc_freq, max_df=
                                 max_doc_freq)
```

Для хранения разреженных матриц внутри алгоритмов пакета scikit-learn используется библиотека scipy. В данной библиотеке разреженные матрицы хранят только ненулевые элементы. В данных матрицах заполнено приблизительно 0.17% элементов. Таким образом, используя разреженные матрицы, сэкономлено большое количество памяти.

Логистическая регрессия - это линейная регрессия, выход которой сжимается в диапазон от 0 до 1 с помощью логистической функции (сигмоиды). В качестве функции потерь используется категориальная кросс-энтропия. Обучение классификатора происходит с помощью класса LogisticRegression библиотеки scikit-learn [9].

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

clf=LogisticRegression(penalty='l2', tol=1e-4, max_iter=100)
parameters = {'C':[1, 2, 5, 10]}
cv=GridSearchCV(clf, parameters)
cv.fit( vectorized_train , y_train )
cv_df=pd.DataFrame.from_dict(cv.cv_results_ )
cv_df
```

Далее рассматривается оценка качества классификатора на тестовой выборке. Сначала необходимо векторизовать каждый документ тестовой выборки с помощью векторизатора, который использовался при работе с обучающей выборкой. После векторизации документов матрица объекты-признаки передается для классификации в метод predict и predict_proba в классификатор. Таким образом, на выходе функции получается матрица, в котором количество строк соответствует количеству элементов в датасете, а количество столбцов соответствует количеству классов. Различие этих двух методов состоит в том, что метод predict возвращает 1 в случае соответствии классу и 0 в случае, если объект из другого класса, а метод predict_proba возвращает вероятности отнесения документа к определенному классу.

```
y_predict = best_clf.predict( vectorized_test )
y_predict_proba = best_clf.predict_proba( vectorized_test )
```

Для целей анализа процесса обучения вычисляется значение функции потерь (категориальная кросс-энтропия) на обучающейся выборке, а также находится accuracy (т. е. долю верных ответов). Данная метрика может ис-

пользоваться только тогда, когда датасет сбалансирован. В противном случае данная метрика приводит к завышенной оценки качества модели.

Вывод значений функции потерь с помощью программного кода представлен ниже:

```
Среднее значение функции потерь на обучающей выборке = 0.117542
Сумма значений функции потерь на обучающей выборке = 1550.611616
Доля правильных ответов на обучающей выборке = 0.999545

Среднее значение функции потерь на тестовой выборке = 0.441406
Сумма значений функции потерь на тестовой выборке = 2495.707405
Доля правильных ответов на тестовой выборке = 0.914751
```

Таким образом, обучающую выборку модель практически запомнила - классификатор идеально работает на обучающей выборке. На тестовой выборке модель дает правильные результаты только в 91.5% случаев. Значение функции потерь на обучении - порядка одной десятой, и на тестовой выборке значение функции потерь имеет тот же порядок. Таким образом можно сделать вывод о том, что модель не переобучается.

2.2 Web-приложение для демонстрации работы классификатора

Для демонстрации работы классификатора (т. е. для классификации любого текст, заданный пользователем) было создано простое web-приложение с возможностью ввода текста. Приложение состоит из текстового поля и кнопки, после нажатия которой классификатору подается на вход текст из текстового поля; результат работы классификатора отображается в отдельном модальном окне.

Для разработки web-приложения на серверной части использовался язык программирования Python, так как необходимо было использовать сериализованные в Python модели для предсказания значения целевой переменной. В качестве web-библиотеки был выбран микрофреймворк Flask. Интерфейс был построен с помощью технологий HTML, CSS, JavaScript, а также библиотеки SemanticUI, содержащей визуальные компоненты.

Для взаимодействия с клиентской частью приложения (интерфейсом) были созданы маршруты для статических ресурсов: страницы index.html, css свойствам, javascript файлам. Для применения классификатора (запуска классификатора на тексте) был создан helper. При запуске данный helper вычи-

тывает сериализованные векторизатор TF-IDF и собственно классификатор. Для взаимодействия helper и модуля, который отвечает за маршруты, был создан модуль classifier_controller, который принимает запрос, посланный в web-приложение, и отдает его в helper. Данный модуль преобразует класс текста в читаемый вид. Также в случае, если классификатор не распознал класс документ, модуль делает преобразование из None в текст «Unknown» для удобной обработки на клиентской части.

На клиентской части наибольший интерес представляется логика взаимодействия с серверной частью. Взаимодействие осуществляется при нажатии кнопки «Классифицировать текст» и реализовано с помощью AJAX запроса к серверу (файл app.js).

Начальный интерфейс приложения представлен на рисунке 1:

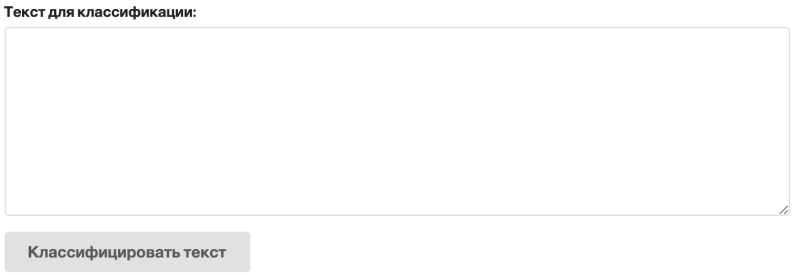

Рисунок 1 – Интерфейс пользователя для приложения классификации текстов

ЗАКЛЮЧЕНИЕ

Машинное обучение используется во многих областях нашей жизни — от оптимизации поиска в Google до прогнозирования распространения раковых клеток. Оно повлияло на то, как мы живем и как работаем, больше, чем любая другая технология со времени появления интернета.

В данной работе была описана одна из задач обработки естественного языка, а именно тематическая классификация текста. Модель обучалась на размеченном датасете, содержащем около 18 тысяч сообщений электронной почты. Данный датасет является встроенным в библиотеку машинного обучения scikit-learn. Была использована простая модель классификатора из класса линейных классификаторов - логистическая регрессия. Доля правильных ответов на тестовой выборке составила приблизительно 91%, что является хорошим результатом для линейных моделей. Также было создано web-приложение для демонстрации работы классификатора. Данная работа может найти свое применение при построении систем автоматической классификации текстов (почтовое приложение, социальная сеть и другие). Дальнейшее улучшение данного классификатора возможно с помощью более сложных алгоритмов, например, методов глубокого обучения нейронных сетей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Ильвовский, Д.* Системы автоматической обработки текстов [Электронный ресурс] / Д. Ильвовский, Е. Черняк. — 2014. — URL: https://www.osp.ru/os/2014/01/13039687/ (Дата обращения 01.05.2020). Загл. с экр. Яз. рус.

2 *Turing, A. M.* Computing machinery and intelligence [Электронный ресурс] / A. M. Turing. — 1950. — URL: https://www.csee.umbc.edu/courses/471/papers/turing.pdf (Дата обращения 01.05.2020). Загл. с экр. Яз. англ.

3 *Samuel, A.* Some studies in machine learning using the game of checkers [Электронный ресурс] / A. Samuel. — 1959. — URL: https://www.cs.virginia.edu/~evans/greatworks/samuel.pdf (Дата обращения 02.05.2020). Загл. с экр. Яз. англ.

4 *Mitchell, T. M.* Machine Learning / T. M. Mitchell. — USA, New York: McGraw-Hill, 1997.

5 Accuracy, precision, recall or f1? [Электронный ресурс]. — URL: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 (Дата обращения 03.05.2020). Загл. с экр. Яз. англ.

6 *Bengfort, B.* Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning / B. Bengfort, T. Ojeda, R. Bilbro. — USA, Sebastopol: O'Reilly Media, 2018.

7 *Defazio, A.* Saga: A fast incremental gradient method with support for non-strongly convex composite objectives / A. Defazio, F. Bach, S. Lacoste-Julien. — 2014. — URL: https://arxiv.org/abs/1407.0202 (Дата обращения 04.05.2020). Загл. с экр. Яз. англ.

8 The 20 newsgroups text dataset [Электронный ресурс]. — URL: https://scikit-learn.org/stable/datasets/index.html#newsgroups-dataset (Дата обращения 04.05.2020). Загл. с экр. Яз. англ.

9 Logistic regression [Электронный ресурс]. — URL: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (Дата обращения 06.05.2020). Загл. с экр. Яз. англ.