Федеральное государственное образовательное бюджетное

учреждение высшего образования

**«Финансовый университет при Правительстве**

**Российской Федерации»**

Департамент анализа данных и машинного обучения

Пояснительная записка к курсовой работе по дисциплине

«Технологии анализа данных и машинное обучение»

на тему:

**«Машинное обучение в задачах распознавания объектов на фотографии»**

Выполнил:

студент ПИ19-3

Деменчук Г. М.

Научный руководитель:

доцент, канд. пед. наук

Норм мужик П. В.

**Москва**

**2022**

ОГЛАВЛЕНИЕ

[I. ВВЕДЕНИЕ 4](#_Toc100329498)

[II. ОСНОВНЫЕ ПОНЯТИЯ ПРИ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ 6](#_Toc100329499)

[Различия между глубоким и классическим машинным обучением 6](#_Toc100329500)

[Особенности обработки изображений 6](#_Toc100329501)

[Метрики качества 7](#_Toc100329502)

[III СБОР И ПРЕПРОЦЕССИНГ ДАННЫХ 8](#_Toc100329503)

[Сбор данных 8](#_Toc100329504)

[Предварительный анализ данных 8](#_Toc100329505)

[Преобразование данных 8](#_Toc100329506)

[IV. СОЗДАНИЕ АЛГОРИТМОВ И ОБУЧЕНИЕ МОДЕЛЕЙ 9](#_Toc100329507)

[Метод Байеса 9](#_Toc100329508)

[Метод k-ближайших соседей 10](#_Toc100329509)

[Логистическая регрессия 11](#_Toc100329510)

[Метод опорных векторов 12](#_Toc100329511)

[Метод «случайного леса» 13](#_Toc100329512)

[Линейный классификатор с SGD-обучением 15](#_Toc100329513)

[Метод градиентного бустинга 16](#_Toc100329514)

[V. СРАВНЕНИЕ МОДЕЛЕЙ 18](#_Toc100329515)

[V. ЗАКЛЮЧЕНИЕ 18](#_Toc100329516)

[X. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc100329517)

[XI. ПРИЛОЖЕНИЯ 21](#_Toc100329518)

[Приложение A. Диаграмма классов разработанного приложения-сервера с учётом иерархии модулей 21](#_Toc100329519)

# I. ВВЕДЕНИЕ

На сегодняшний день методы машинного обучения используются во всех областях, где необходимо предсказать, классифицировать или обобщить данные на основе каких-либо критериев: распознавание речи, обработка естественных языков, медицинская диагностика, интеллектуальные игры, техническая диагностика, но большим и наиболее ресурсозатратным направлением является компьютерное зрение, которое в том числе включается себя задачи классификации и распознавания объектов на фотографии.

Данная работа посвящена сравнению классических методов машинного обучения с глубоким машинным обучением при работе с изображениями. Большой акцент сделан на сборе, анализе и предобработке исходных изображений для их последующего использования при обучении, также рассмотрены итеративные методы решения задач машинного зрения и их применение в настоящее время. …Мы также отдельно сравниваем производительность глубокой сети при использовании классического способа обучения (ANN) и при использовании быстрого обучения на основе сверточных нейросетей (CNN).

# II. ОСНОВНЫЕ ПОНЯТИЯ ПРИ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ

## Различия между глубоким и классическим машинным обучением

Для понимания дальнейшего контекста происходящего, необходимо ввести понятия классического и глубокого машинного обучения.

## Особенности обработки изображений

Отличительной чертой при обработке изображений является многофакторность – для изображения 100\*100 пикселей в RGB формате существует 100\*100\*3 входных аргумента. Именно из-за этой особенности все операции для машинного обучения изображений обычно осуществляются не на CPU, где акцент смещен на оперативность обработки данных, а на GPU, где сделан больший акцент на возможность параллельной обработки данных.

\*КРУТОЙ РИСКУНОК С ДРОИДЕРА РАЗНИЦЫ МЕЖДУ CPU и GPU\*

Для работы с GPU разных производителей есть несколько наиболее распространённых API:

* CUDA – лал ла технология от Nvidia для их видеокарт
* OpenCL – лоло технология от AMD
* Metal – лололо технология от Apple

Из особенностей мы получаем сразу несколько следствий:

1. Т.к. библиотека scikit-learn обычно используется либо как препроцессор для библиотек глубокого обучения (pytorch, keras), либо как набор классических алгоритмов и не поддерживает API для работы с GPU, то обучение на столь большом кол-ве факторов может длится от нескольких часов до нескольких дней, причем некоторые алгоритмы даже невозможно распараллелить по нескольким ядрам процессора.
2. Необходимо тщательно фильтровать исходные датасеты для получения хоть сколько-нибудь существенного процента распознавания данных на классических алгоритмах, причем использовать достаточно очевидные и разноплановые классы (например, дерево, птица, автомобиль). Например, модели автомобилей классифицировать практически невозможно т.к. существует достаточно большое кол-во общих элементов и их очертаний (колёса, решетка радиатора, фары и т.д.)

## Метрики качества

Предметной областью автоматизации является чат-мессенджер. Логика его работы достаточно проста: пользователь создает чат-комнату с указанием имени и выбором пользователей-участников, переходит в неё и начинает общение. Участника в комнате может быть два или более, тогда чат-комната будет диалогом или беседой соответственно. В окне диалога у пользователя существует возможность чтения предыдущих сообщений, отправки и получения новых, а также просмотра информации об участниках чата.

Всего со стороны клиента используются 3 сущности для представления данных:

* Пользователь – сущность представления конкретного пользователя решения. В один времени может быть авторизован только один пользователь со стороны клиента;
* Комната – чат конкретных пользователей-участников. У одного пользователя может быть несколько комнат, у комнаты может быть несколько участников, но только один пользователь-администратор;
* Сообщение – текстовое сообщение пользователя в чате. Содержит информацию об идентификаторе пользователя и чат-комнаты, к которой принадлежит.

Со стороны же сервера дополнительно объявляется сущность лонгпула, которая содержит в себе основную информацию о длинных соединениях клиента с сервером. Более подробно о об атрибутах сущностей можно прочитать в пункте «База данных» главы «СОСТАВ ПРИЛОЖЕНИЯ».

# III СБОР И ПРЕПРОЦЕССИНГ ДАННЫХ

## Сбор данных

Точность классификации моделей при использовании классических алгоритмов напрямую зависит от качества исходных датасетов изображений и классов. Для подтверждения данных суждений возникает необходимость создания нескольких датасетов разных классов и качества:

* Собственноручно собранный датасет изображений реальных автомобилей с сайта по продаже автомобилей Drom.ru. Содержит 34747 нефильтрованных изображений, от 910 × 411 до 1200 × 2667 пикселей, 90 классов фирм-производителей.
* Готовый датасет машин Стэнфордского университета. Содержит 16185 изображения 196 моделей машин. По умолчанию предлагается разделение данных 50% на 50% (8144 обучающих и 8041 тестовых изображений)
* Готовый датасет объектов CIFAR-10. Состоит из 60000 цветных изображений 32x32 в 10 классах, по 6000 изображений на класс. По умолчанию разделяется на 50000 обучающих и 10000 тестовых изображений. 10 классов абсолютно разного характера (самолет, автомобиль, птица, кот, олень, собака, лягушка, лошадь, корабль, грузовик)

Наиболее интересен способ сбора собственноручного датасета с Drom.ru. Обычно, если есть необходимость сбора каких-либо данных, то идут двумя самыми распространенными путями:

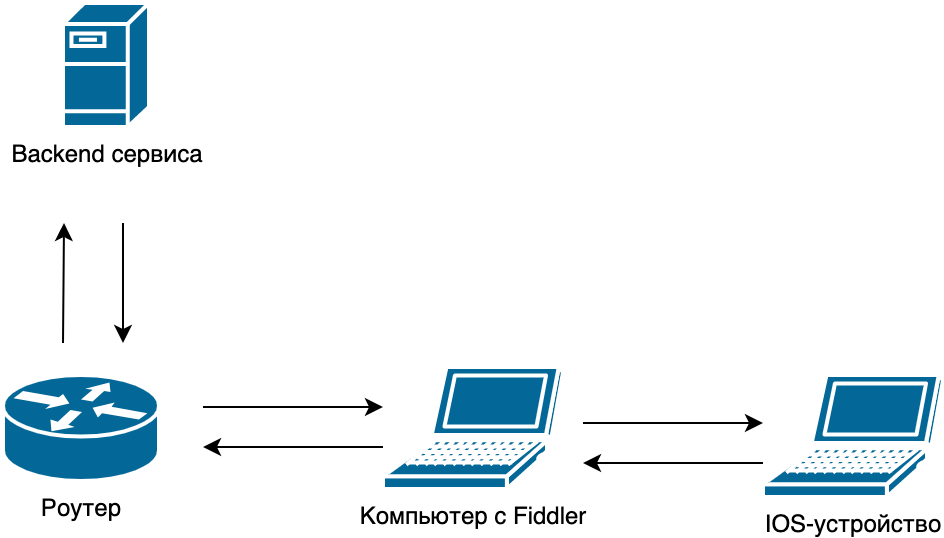
* Парсинг сайта. На стеке Python типично используется связка Chromedriver + Selentium + BeautifulSoup, если сайт генерирует HTML страницу на стороне клиента и Requests + BeautifulSoup, если страничка не использует javascript. Здесь есть пару проблем, из-за которых работа может очень сильно осложниться. Во-первых, это капча. После примерно первых 100 просмотренных объявлений начнет срабатывать механизм защиты от роботов и программ-парсеров. Конечно же, есть возможность использовать сервисы по для её обхода по типу antigate или OpenCV (при условии, что капча не очень сложная), но это существенно замедляет процесс разработки и искусственно расширяет исходную задачу по сбору данных. Во-вторых, при изменении дизайна страницы сайта возникнет необходимость в переписывании парсера, что требует постоянной поддержки и мониторинга со стороны разработчика.
* Парсинг мобильного приложения. Обычно используется MITM атака посредством самоподписанного HTTPS-сертификата приложения. В этом есть ряд существенных преимуществ: во-первых, современное мобильное приложение общается с бекендом сайта посредством REST или GraphQL запросов, в ответ присылается XML или JSON с данными, что очень удобно для последующей обработки через какой-либо язык программирования (в т.ч. Python). Во-вторых, API мобильного приложения меняется очень редко, а если и меняется, то старое, как правило, работает еще некоторое количество времени для того, чтоб пользователи на старых версиях операционных систем имели возможность использовать сервис. В-третьих, мобильное приложение реже вызывает механизмы проверки пользователя на программу-парсер т.к. возможно такой метод сбора данных все ещё не очевиден.

Очевидно, что был выбран способ сбора датасета через мобильное приложение. Возникает вопрос: на какой платформе орагнизовать сбор данных? IOS или Android?

* Android – присутствует возможность установки сторонних программ с помощью .apk файлов, их перекомпиляция и доработка. Возможна организация Root доступа для расширения функционала.
* IOS – по умолчанию закрытая ОС. При осуществлении джейлбрейка есть возможность установки .ipa и сторонних программ (твиков)

С первого взгляда кажется, что выбор очевиден в пользу Android, но все не так очевидно. Android-приложения в больших компаниях, как правило, больше тестируются на безопасность, в т.ч. на подмену сертификатов, а у меня уже был опыт перехвата трафика под IOS, так что исходное устройство будет под управлением IOS.

Для прокси, которое будет использоваться для перехвата трафика использовался Fiddler, а результате схема трафика выглядит так:

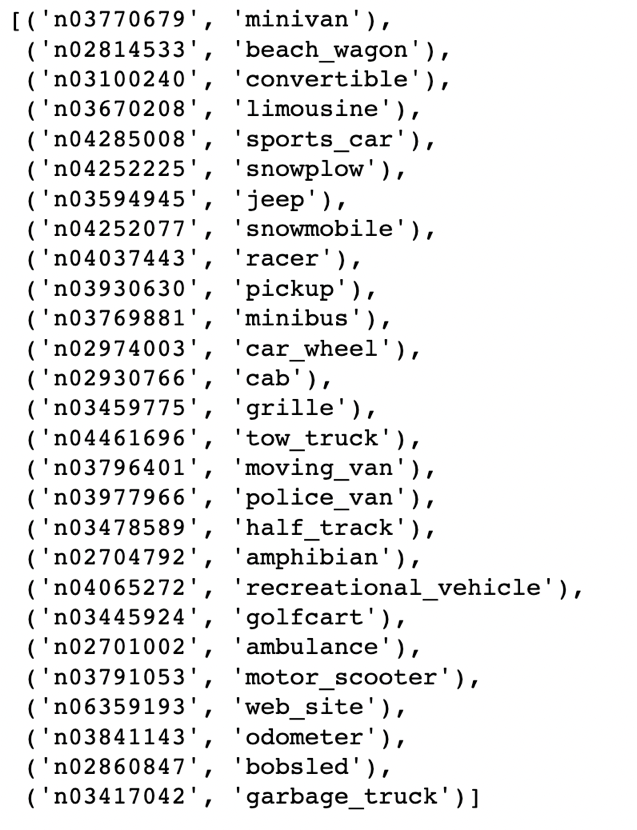


После нескольких часов работы созданного аппаратно-программный комплекса было получено 2241 файла формата JSON, содержащих большое кол-во информации о каждом объявлении, в т.ч. модель, марку автомобиля и url с фотографией автомобиля

## Фильтрация данных датасетов

Т.к. датасеты CIFAR-10 и датасет машин Стэнфорда уже были предварительно подготовлены, то речь в данным разделе пойдет исключительно о фильтрации собственноручно собранного датасета.

1. Все собранные JSON помещаются в один файл в одной папке. Размер собранного файла составляет 171,9 мегабайта и включет 35531 объявлений о продаже автомобиля.
2. Для каждого объявления выкачивается основное фото автомобиля. Дополнительные фото игнорируются, поле URL в JSON меняется на локальный путь к файлу
3. Определяем, что перед нами фото автомобиля. Иногда пользователи публикуют в качестве главного фото не фото самой машины, а приборной панели, вид из салона, документы. Для этого используется сторонняя сверточная сеть глубокого обучения VGG16 с огромным датасетом ImageNet, включающим 14 млн изображений и 1000 классов. Для нас только важно то, что ее точность очень высокая и её результатам можно доверять. Она используется из-за того, что перебирать вручную 35 тысяч изображений достаточно трудозатратная задача. Для всех изображений мы формируем классы, которые относятся к ним и считаем самые распространенные. Изображения, классы которого которые относятся к первым 27 популярным классам, изображают машину

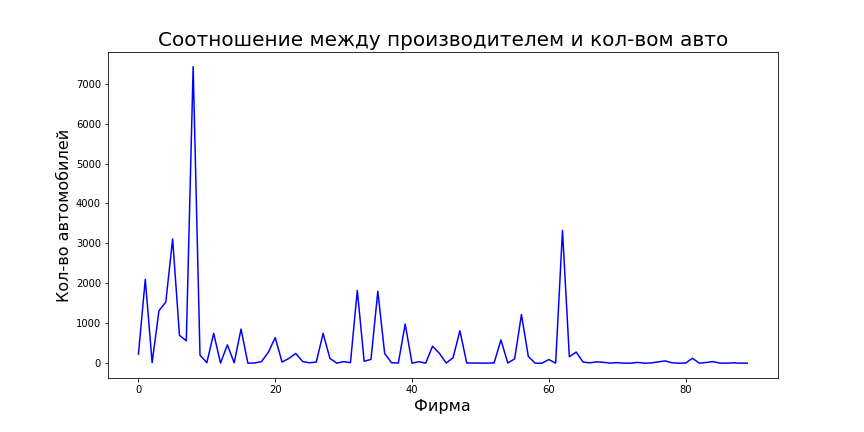


Таким образом было отфильтровано 784 записей и 34747 значений осталось. Использование моделей глубокого обучения для формирования новых датасетов вполне себе обычная практика на сегодняшний день.

## Предварительный анализ данных

### Анализ собственного датасета Drom.ru

Для начала необходимо посмотреть распределение классов между собой



На данном этапе данные совсем не подходят для обучения. Разброс огромный: от 1 фото для многих редких моделей (Диаграмма N) до нескольких тысяч для наиболее популярных автопроизводителей (Диаграмма N)

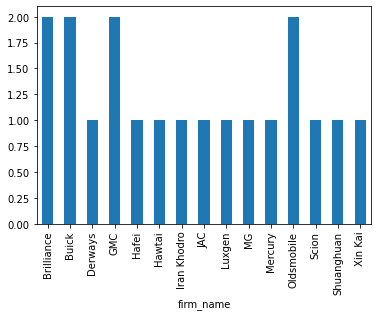


Диаграмма N. Наиболее редкие автомобили

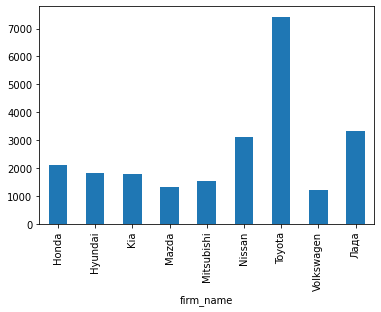


Диаграмма N. Наиболее популярные автомобили

Возьмем наиболее популярных автопроизводителей и сделаем так, чтоб в каждом классе было одинаковое кол-во изображений, как у самого малого класса (1220, как у Volkswagen), остальные просто исключаем из выборки.

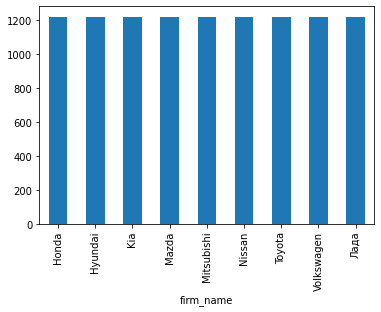


Диаграмма N. Наиболее популярные автомобили

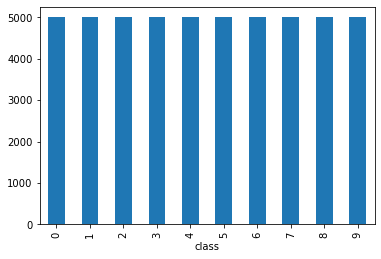
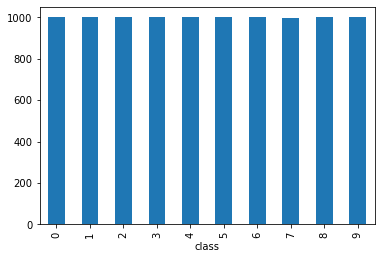
В итоге получилось 9 классов, в каждом по 1220 изображений, всего 10980. С такой моделью уже можно работать.

### Анализ датасета Стенфордского университета

Изначально информация ююю

### Анализ датасета CIFAR-10

Все данные в датасете идеально сбалансированы и уже поделены на тестовую и обучающую выборку. В тестовой выборке 10 классов по 1000 изображений, всего 10000. В обучающей выборке 10 классов по 5000 изображений, всего 50000.



## Преобразование данных

### Представление изображений в матричном виде

Ахахахаахахахахахахаххахахахахаха

### Перевод изображений в черно-белый формат

Ахахахаахахахахахахаххахахахахаха

### Нормализация изображений

АхахахаахахахахахахаххахахахахахаАхахахаахахахахахахаххахахахахаха

# IV. СОЗДАНИЕ АЛГОРИТМОВ И ОБУЧЕНИЕ МОДЕЛЕЙ

## Метод Байеса

Наивный байесовский классификатор является не только одним из самых простейших, используя удобную математическую модель, но и быстро работающим методом, который нередко используют в качестве базового. Также, как правило, этот метод показывает хорошие результаты в задачах классификации текстов. Алгоритм использует теорему Байеса

(7)

В данной формуле P(A) – вероятность наступления события A, P(B) – вероятность наступления события B, P(A|B) – вероятность наступления события A, если произошло событие B, P(B|A) аналогично вероятность наступления события B, если произошло A. В контексте нашей работы является элементом выборки, состоящей из признаков , которые в данном случае слова. – один из наших классов, в данном случае являющихся эмоциональным окрасом текста. Тогда нужно вычислить для каждого из рассматриваемых классов и отнести наши данные к тому классу, где данная вероятность будет больше

(8)

Данный алгоритм нередко называют наивным байесовым классификатором из-за того, что в нём приято считать, что каждое слово вносит свой независимый вклад в эмоциональный окрас текста, что с точки зрения естественного языка не совсем верно. Данный момент всё равно сохраняет высокую точность алгоритма, зато при независимых возможно преобразовать исходную формулу

(9)

In [14]:

#Реализация наивного байесовского метода

from sklearn.naive\_bayes import MultinomialNB

start\_time = time.time()

nb = MultinomialNB().fit(X\_train, y\_train)

y\_pred = nb.predict(X\_test)

print(metrics.classification\_report(y\_test, y\_pred))

print("Время выполнения", (time.time()-start\_time))

Out [14]:

precision recall f1-score support

0 0.70 0.73 0.71 3406

1 0.71 0.17 0.27 1667

2 0.65 0.84 0.73 3918

accuracy 0.67 8991

macro avg 0.68 0.58 0.57 8991

weighted avg 0.68 0.67 0.64 8991

Время выполнения 0.04143691062927246

Метод действительно показал достаточно высокие показатели точности и скорости выполнения. А использование не встроенного векторизатора, а также настройка параметров могли бы сделать результат ещё точнее.

## Метод k-ближайших соседей

В методе ближайших соседей не требуется фаза обучения, так как он производит сравнение интересующего нас объекта со всеми элементами обучающей выборки в результате чего находится косинус угла между векторами признаков или по-другому – расстояние:

(10)

На следующем шаге определяется параметр k (как правило от 1 до 100) и в обучающей выборке определяют k ближайших элементов к интересующему нас объекту b, вычисляется релевантность и объект относят к тому классу, у которого она выше заданного порога:

(11)

Код программы будет следующий:

In [15]:

# Реализация метода k-ближайших соседей

import sklearn.neighbors as neigh

start\_time = time.time()

kn = neigh.KNeighborsClassifier(n\_neighbors=90).fit(X\_train,y\_train)

y\_pred = kn.predict(X\_test)

print(metrics.classification\_report(y\_test, y\_pred))

print("Время выполнения", (time.time()-start\_time))

Out [15]:

precision recall f1-score support

0 0.93 0.07 0.13 3406

1 0.20 0.99 0.33 1667

2 0.91 0.10 0.19 3918

accuracy 0.26 8991

macro avg 0.68 0.39 0.22 8991

weighted avg 0.79 0.26 0.19 8991

Время выполнения 16.036214590072632

Метод выполнялся несколько дольше, показал схожие результатом в целом по точности, но очень слабо определил принадлежность к классу с нейтральным окрасом. Также отметим довольно низкие показатели полноты. В дальнейшем они окажутся самыми низкими.

## Логистическая регрессия

Логистическая регрессия получила своё название по историческим причинам, и данный метод не имеет отношения к регрессии, поскольку является подходом к проблемам классификации. Он основывается на вероятностях принадлежности данных к определённому классу, которые моделируются с помощью логистической функции, являющейся кумулятивной функцией распределения.

In [16]:

# Реализация логистической регресси

from sklearn.linear\_model import LogisticRegression

start\_time = time.time()

logr = LogisticRegression(max\_iter=500).fit(X\_train, y\_train)

y\_pred\_classification = logr.predict(X\_test)

print(metrics.classification\_report(y\_test, y\_pred))

print("Время выполнения", (time.time()-start\_time))

Out [16]:

precision recall f1-score support

0 0.79 0.82 0.80 3406

1 0.74 0.62 0.68 1667

2 0.82 0.85 0.84 3918

accuracy 0.80 8991

macro avg 0.78 0.76 0.77 8991

weighted avg 0.79 0.80 0.79 8991

Время выполнения 6.922231197357178

Логистическая регрессия показывает отличные показатели как скорости выполнения, так и точности. Лучше всего она справилась с положительной тональностью.

## Метод опорных векторов

В методе опорных векторов выборка принимает вид , в котором за обозначен i-ый текст, представленный в виде вектора размерности m, а – число, обозначающее класс, к которому принадлежит данный текст. Принцип работы заключается в поиске гиперплоскости, которая разделяет вектора классов, расстояние от которой до векторов обучающей выборки, находящихся ближе всего к гиперплоскости, или же до опорных векторов максимально. Данная гиперплоскость будет иметь вид , тогда как расстояние от неё до любого опорного вектора будет равно , где - скалярное произведение, а норма w. Таким образом от задачи поиска максимума переходим к поиску минимума , которую можно записать следующим образом:

(12)

Данная задача решается методом Лагранжа, в результате которого находятся w и v, с помощью которых определяется функция классификации, имеющая следующий вид:

(13)

Функция sign(x), возвращает 1, 0 или -1 в зависимости от знака x. В тех случаях, когда не предоставляется возможным разделить исходные данные гиперплоскостью, применяют отображение , которое так отображает элементы обучающей выборки, чтобы они стали линейно разделимы, и в таком случае функция классификации несколько изменит свой вид:

(14)

In [17]:

# Реализация метода опорных векторов SVM

import sklearn.svm as svm

start\_time = time.time()

svect = svm.LinearSVC(max\_iter=5000).fit(X\_train, y\_train)

y\_pred = svect.predict(X\_test)

print(metrics.classification\_report(y\_test, y\_pred))

print("Время выполнения", (time.time()-start\_time))

Out [17]:

precision recall f1-score support

0 0.81 0.82 0.81 3406

1 0.73 0.69 0.71 1667

2 0.84 0.85 0.85 3918

accuracy 0.81 8991

macro avg 0.80 0.79 0.79 8991

weighted avg 0.81 0.81 0.81 8991

Время выполнения 0.7738602161407471

Мы наблюдаем более высокие показатели метрик по сравнению с предыдущими методами и при этом значительно более быстрое выполнение алгоритма.

## Метод «случайного леса»

«Random forest» - метод, который был придуман Лео Брейманом и Адель Катлер ещё в XX веке и отличается своей универсальностью как по качеству работы (средне значение из разных источников – около 70%), так и по спектру выполняемых задач (регрессия, классификация, кластеризация, селекция признаков поиск аномалий и так далее). Случайный лес – это множество решающих деревьев. Если мы работаем с задачей классификации, то решение будет принято голосованием по большинству в отличии от задач регрессии, где ответы усредняются. Деревья являются независимыми и строятся по следующему алгоритму. Из обучающей выборки выбирается специально для каждого дерева своя подвыборка, по которой оно и строится. Далее строится расщепление в дереве, для чего просматривается определённое количество случайный признаком, причём для каждого расщепления свои признаки. Затем, используя заранее заданный критерий, определяются наилучшие признак и расщепление по нему. Построение дерева продолжается то тех пор, пока в листьях не останутся представители только одного класса, то есть до исчерпания выборки, но отметим, что современные реализации алгоритма включают в себя ограничение на число объектов в листья, высоту дерева, а также размер подвыборки, при котором проводится расщепление. Реализуем данный метод:

In [17]:

from sklearn.model\_selection import GridSearchCV

# Реализация метода случайного леса RandomForest

from sklearn.ensemble import RandomForestClassifier

start\_time = time.time()

max\_depth = [1,5,10,50]

n\_estimators = [5,10,100,500]

grid\_params ={'max\_depth':max\_depth,'n\_estimators':n\_estimators}

RandomFoest\_model = GridSearchCV(RandomForestClassifier(class\_weight = 'balanced'), grid\_params,

                  scoring = 'accuracy', cv=10,n\_jobs=-1, return\_train\_score=True)

RandomFoest\_model.fit(X\_train, y\_train)

y\_pred = RandomFoest\_model.predict(X\_test)

print(metrics.classification\_report(y\_test, y\_pred))

print("Время выполнения", (time.time()-start\_time))

Out [17]:

precision recall f1-score support

0 0.74 0.64 0.69 3406

1 0.49 0.78 0.60 1667

2 0.79 0.69 0.73 3918

accuracy 0.69 8991

macro avg 0.67 0.70 0.68 8991

weighted avg 0.72 0.69 0.69 8991

Время выполнения 832.5463447570801

Этот классификатор однозначно выделяет самое продолжительное время работы из всех рассматриваемых. При этом показатели его метрик не так высоки, чтобы отдать предпочтение данному методу.

## Линейный классификатор с SGD-обучением

Линейный классификатор представляет собой алгоритм, который базируется на построении линейной разделяющей плоскости, являющейся гиперплоскостью в случае двух классов, а если их больше, то кусочно-линейной. Наши объекты описываются числовыми признаками в количестве n, и получаем пространство признаковых описаний . Обозначим множество номеров\меток\имён за Y. Допустим, Y = {-1, 1}. В таком случае линейный алгоритм классификации будет иметь вид

(15)

В данной формуле – признак, - порог принятия решений, – вес j-ого признака, которые образуют вектор весов , < > - скалярное произведение. Если мы работаем с классификацией для произвольного числа классов, то рассматриваем вектор весов для каждого класса , а формула примет следующий вид:

(16)

Обучать линейный классификатор можно различными способами. Ранее уже были рассмотрены логистическая регрессия и метод опорных векторов, но сейчас речь пойдёт о стохастическом градиентном спуске, который даёт возможность решить проблему требовательности к ресурсам обычного градиентного спуска, ведь при больших наборах обучающей выборки, вычислять функцию ошибки для каждого её элемента довольно затратно. Для этого мы разбиваем выборку на K частей размера M.

(17)

(18)

Получим функцию и набор из K мини итераций градиентного спуска вместо одной:

(19)

(20)

Назовём эпохой каждую большую итерацию p, между которыми обучающая выборка перемешивается для обеспечения попадания каждому элементу в разные мини выборки. Получаем, что наши штрафные функции суммируются по разным поднаборам X и Y.

In [18]:

from sklearn.model\_selection import GridSearchCV

# Реализация линейного классификатора с SGD-обучением

from sklearn.linear\_model import SGDClassifier

start\_time = time.time()

alpha = [10\*\*x for x in range(-4,4)]

penalty = ["l1","l2"]

grid\_params ={'alpha':alpha,'penalty':penalty}

alpha\_log = [math.log(x,10) for x in grid\_params["alpha"]]

SGDClassifier\_model = GridSearchCV(SGDClassifier(class\_weight= 'balanced'), grid\_params,

                     scoring = 'accuracy', cv=10,n\_jobs=-1, return\_train\_score=True)

SGDClassifier\_model.fit(X\_train, y\_train)

y\_pred = SGDClassifier\_model.predict(X\_test)

print(metrics.classification\_report(y\_test, y\_pred))

print("Время выполнения", (time.time()-start\_time))

Out [18]:

precision recall f1-score support

0 0.81 0.76 0.78 3406

1 0.60 0.79 0.68 1667

2 0.85 0.79 0.82 3918

accuracy 0.78 8991

macro avg 0.75 0.78 0.76 8991

weighted avg 0.79 0.78 0.78 8991

Время выполнения 37.67648005485535

## Метод градиентного бустинга

Алгоритм XGBoost основывается на градиентном бустинге деревьев решений и используется в задачах классификации и регрессии. Этот метод машинного обучения строит модель предсказания, представленную в форме ансамбля слабых предсказывающих моделей, как деревья решений. Обучается данный алгоритм последовательно. Для начала на каждой итерации на обучающих данных определяют отклонения предсказаний, которые предсказывает новая модель, добавленная в ансамбль, и получается, что каждое добавленное предсказание нового дерева уменьшает среднее отклонение модели. Поэтому деревья добавляются до момента уменьшения ошибки. На языке python модель реализуется с использованием библиотеки:



In [19]:

# Реализация метода градиентного бустинга XGBoosted

from xgboost import XGBClassifier

start\_time = time.time()



XGBoost\_model = XGBClassifier(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,



       colsample\_bytree=1, gamma=0, learning\_rate=0.1, max\_delta\_step=0,

       max\_depth=7, min\_child\_weight=1, missing=None, n\_estimators=500,

       n\_jobs=1, nthread=None, objective='multi:softprob', random\_state=0,

       reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, seed=None,

       silent=True, subsample=1)

XGBoost\_model.fit(X\_train,y\_train)

y\_pred = XGBoost\_model.predict(X\_test)

print(metrics.classification\_report(y\_test, y\_pred))

print("Время выполнения", (time.time()-start\_time))

Out [19]:

precision recall f1-score support

0 0.77 0.77 0.77 3406

1 0.69 0.73 0.71 1667

2 0.83 0.81 0.82 3918

accuracy 0.78 8991

macro avg 0.76 0.77 0.77 8991

weighted avg 0.78 0.78 0.78 8991

Время выполнения 248.98635244369507

Данный метод тоже показывает довольно продолжительное время работы, но показатели метрик хорошие.

# V. СРАВНЕНИЕ МОДЕЛЕЙ

Тут мы с пацанами будем сравнивать модели

# V. ЗАКЛЮЧЕНИЕ

В ходе написания данной работы было разработано клиент-серверное приложение для возможности обмена текстовыми сообщениями между несколькими пользователями, включающее в себя сервер с использованием библиотеки Spring Boot и клиент с применением GUI библиотеки JavaFX.

Особенностями данной работы является её своеобразный стек разработки: со стороны сервера был использован интерфейс MongoRepository для доступа к данным NoSQL СУБД MongoDB через Spring ORM, а со стороны клиента работают несколько потоков для получения обновлений сообщений пользователя с помощью механизма лонгпулинга с последующей синхронизацией через главный поток JavaFX.

Приложение полностью готово к дистрибуции: чат-сервер Spring запакован в образ docker-контейнера с помощью плагина spring-boot в Maven и загружен в репозитории Docker Hub и GitHub Packages, а образы клиентов JavaFX скомпилированы в Java Runtime Image с использованием инструмента Jlink. Все методы решения могут генерировать автоматически составляемую документацию при помощи инструмента JavaDoc.

Созданное решение удовлетворяет всем требованиям и задачам: реализует CRUD-методы со стороны бекенда, а фронт осуществляет получение и отправку данных через RESTful API с помощью протокола http с последующим вводом/выводом данных на элементы управления графического интерфейса JavaFX.

Решение может модернизироваться и обновляться. Например, можно добавить возможность отправки любых типов файлов в чат-комнаты, в том числе фото и видео. Также можно внедрить голосовые сообщения, как это сделано в большинстве популярных мессенджеров. В дальнейшем можно написать клиенты под мобильные устройства и подключить к существующему серверу.

# X. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Законодательные и нормативные акты

1. ГОСТ 2.316-2008. Правила нанесения надписей, технических требований и таблиц на графических документах.
2. ГОСТ 7.1-2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.
3. ГОСТ 7.32-2001. Система стандартов по информацию, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: ИПК Издательство стандартов, 2001. – 21 с.

Ссылки

<https://ai.stanford.edu/~jkrause/cars/car_dataset.html> - датасет машин Стэнфордского униерситета

<https://www.cs.toronto.edu/~kriz/cifar.html> - датасет CIFAR-10

Учебная и научная литература

1. Володченкова Л.А., Козырев Д.В. Разработка серверной части программного приложения для удаленного хранения данных // МСиМ. 2020. №1 (53).
2. Байдыбеков А.А., Гильванов Р.Г., Молодкин И.А. СОВРЕМЕННЫЕ ФРЕЙМВОРКИ ДЛЯ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ // Интеллектуальные технологии на транспорте. 2020. №4 (24).
3. Гасанов Заурбек Зубаирович Анализ производительности многопоточных программ, написанных на языках Java и Go // Наука и образование сегодня. 2018. №6 (29).
4. Барабанов В.Ф., Донских А.К., Гребенникова Н.И., Кенин С.Л. ПОЛУЧЕНИЕ МЕТРИК JAVA-ПРИЛОЖЕНИЯ В КОНТЕЙНЕРАХ DOCKER // Вестник ВГТУ. 2020. №2.
5. О.А. Ляшенко, О.О. Конашков, Н.А. Солодкая СРАВНИТЕЛЬНЫЙ АНАЛИЗ ВЫПОЛНЕНИЯ ЗАПРОСОВ К СЕРВЕРАМ БАЗ ДАННЫХ MYSQL И MONGODB // Вестник Херсонского национального технического университета. 2019. №4 (71).

# XI. ПРИЛОЖЕНИЯ

## Приложение A. Диаграмма классов разработанного приложения-сервера с учётом иерархии модулей

Тут приложение хехех