Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ ПО КУРСУ ЧИСЛЕННЫЕ МЕТОДЫ НА ТЕМУ:

«Численный анализ данных о результативности преступлений с применением оружия»

Студент ИУ9-72Б _____ Сербин Д. А.

подпись, дата *фамилия, и.о.*

Научный руководитель _____ Домрачева А. Б.

подпись, дата *фамилия, и.о.*

2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире количество преступлений растет с каждым годом. Большинство преступлений происходят с применением оружия. Полиции становится все труднее и труднее успевать предотвратить преступление из-за постоянного роста их количества. Так же многие людям хотели бы знать, насколько они находятся в безопасности в тех или иных условиях. Поэтому особенно актуальна возможность предсказывать результативность преступления по различным признакам.

Чтобы разработать предсказывающую модель, которая обеспечит эти возможности, необходимо изучить теоретическую часть, выбрать данные для обучения, проанализировать и преобразовать их.

Целью данной работы является реализация подобной предсказывающей модели.

1. Теоретическая часть

1.1 Классификационная Модель

Чтобы предсказать результат инцидента необходимо воспользоваться предсказывающей моделью. При выборе модели нужно учитывать, что результатом предсказания является одно из двух значений, а именно пострадает ли человек в данных обстоятельствах или нет. Из этого был сделан вывод, что в данном случае нам максимально подходят алгоритмы классификации. Таким алгоритмом является логистическая регрессия.

Логистическая регрессия – один из статистических алгоритмов классификации, позволяющий по входным значениям получить ответ, который возможно однозначно классифицировать.

Логистическая регрессия – алгоритм классификации, который ищет взаимосвязи между входными данными и вероятностью результата.

Логистическая регрессия предсказывает вероятность двоичного события.

Для классификации используется функция сигмоиды:

$$sigmoid = \frac{1}{1 + e^{-z}}, \text{где } z - \text{входные значения}, sigmoid - \text{предсказанный результат}$$

На рисунке 1 представлен график функции сигмоиды. Сигмоида это s-образная кривая, которая дает значения в диапазоне от 0 до 1. При x стремящемся к бесконечности, сигмоида принимает значение 1 или 0. При результате сигмоиды больше 0.5 можно классифицировать ответ как 1. При результате сигмоиды меньше 0.5 можно классифицировать ответ как 0.

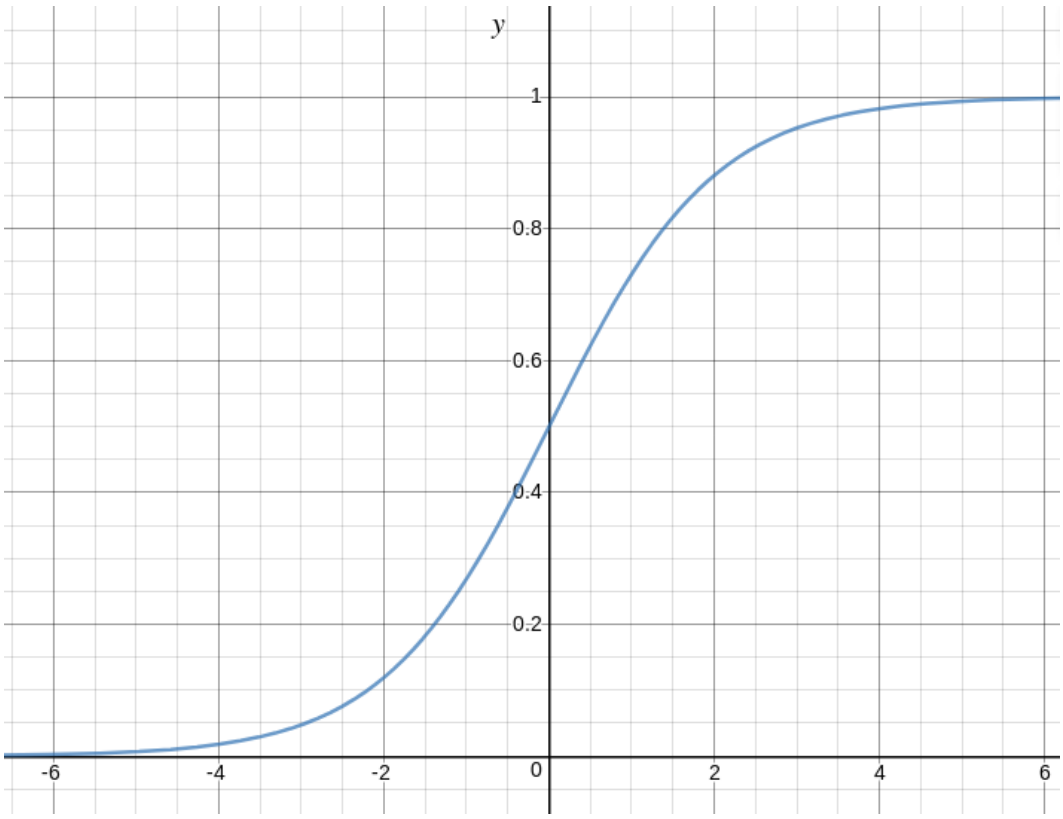

Рисунок 1 - График функции сигмоиды

Входным значением для логистической регрессии является функция:

$$z = \sum w_i x_i + b \text{, где } w_i - \text{подбирающиеся веса, } b - \text{дополнительный вес отклонения (смещение)}$$

После получения результата классификации используется стандартный алгоритм градиентного спуска для оптимизации параметров модели по формуле:

$$\theta_{i+1} = \theta_i - learning\_rate * \nabla L(f(x;\theta), y),$$

где θ_{i+1} – обновленное значение весов, θ_i – старое значение весов, $learning\_rate$ – скорость обучения, ∇L – функция потерь.

Функция потерь имеют формулу:

$$L = -\frac{1}{n}\sum_{1}^{n} y_{train} * \log(sigmoid) + (1 - y_{train}) * \log(1 - sigmoid),$$

где y_{train} − реальный результат,

$sigmoid$ − результат классификации

Для получения потерь весов и смещения используются формулы:

$$\frac{\partial L}{\partial w} = \frac{1}{n}(sigmoid - y_{train})x_i^T,$$

$\frac{\partial L}{\partial b} = \frac{1}{n}(sigmoid - y_{train})$, где y_{train} − реальный результат,

$sigmoid$ − результат классификации, x_i^T

− транспонированный вектор входных данных

1.2 Метрики

После обучения логистической регрессии, необходимо на тестовых данных провести анализ корректной работы модели. Для этого используют различные метрики.

В моделях классификации может возникнуть дисбаланс классов, когда один из предсказываемых результатов - классов в тренировочном наборе данных встречается чаще чем другой, что плохо сказывается на обучении модели. Классы не сбалансированы, когда их соотношение далеко от 1 к 1. При соотношении 1 к 1 классы являются сбалансированными. Существуют метрики для работы в случае дисбаланса классов[1]:

- матрица ошибок
- кривая точности и полноты

1.2.1 Матрица ошибок

Класс с меткой «истина» называется положительным, а с меткой «ложь» – отрицательным. Все ответы модели можно разделить на 4 типа:

- истинно положительные ответы (True Positive, TP): модель пометила объект меткой «истина», и его настоящая метка тоже «истина»
- истинно отрицательные ответы (True Negative, TN): модель пометила объект меткой «ложь», и его настоящая метка тоже «ложь»
- ложноположительные ответы (False Positive, FP): модель предсказала «истина», а действительное значение класса — «ложь»
- ложноотрицательные ответы (False Negative, FN): модель предсказала «ложь», а действительное значение класса — «истина»

Матрица ошибок показывает отношение ответов в одной матрице. На рисунке 2 показана матрица ошибок, где истинно положительные, истинно отрицательные, ложноположительные, ложноотрицательные ответы собираются в одну матрицу, которая формируется следующим образом:

- по горизонтали располагаются предсказания – метки алгоритма от 0 до 1
- по вертикали располагаются ответы — истинные метки класса от 0 до 1
- По главной диагонали выстроены правильные прогнозы, где истинно отрицательные ответы располагаются в левом верхнем углу, а истинно положительные ответы располагаются в правом нижнем углу
- Вне главной диагонали располагаются ошибочные варианты, где ложноположительные ответы располагаются в правом верхнем углу, а ложноотрицательные ответы располагаются в левом нижнем углу

Рисунок 2 - Формирование матрицы ошибок

1.2.2 Кривая точности и полноты

Для создания кривой точности и полноты воспользуемся матрицей ошибок.

Полнота выявляет, какую долю положительных среди всех ответов выделила модель. Полнота рассчитывается по такой формуле:

$$recall = \frac{TP}{TP + FN}\text{, где } recall - \text{полнота}$$

Полнота — это доля TP-ответов среди всех с истинной меткой «1». Хорошо, когда значение полноты близко к единице: модель хорошо ищет положительные объекты. Если ближе к нулю − модель надо перепроверить и починить. Метрика показывает долю правильных определенных положительных ответов из всех действительно положительных ответов. Чем меньше ложно отрицательных ответов, тем выше полнота модели.

Точность определяет, как много отрицательных ответов нашла модель, пока искала положительные. Чем больше отрицательных, тем ниже точность. Точность рассчитывается по такой формуле:

$$precision = \frac{TP}{TP + FP}$$, где $precision$ – точность

Метрика показывает долю правильно опрееделенных положительных ответов среди всех найденных положительных классификатором. Чем меньше ложноположительных ответов будет допускать модель, тем больше будет её точность.

На рисунке 3 представлен график точности и полноты. На графике по вертикали наносится значение точности, по горизонтали – полноты. Кривая показывает компромисс между точностью и полнотой для различных пороговых значений. Критерием качества модели является площадь под графиком. Чем выше кривая, тем лучше модель.

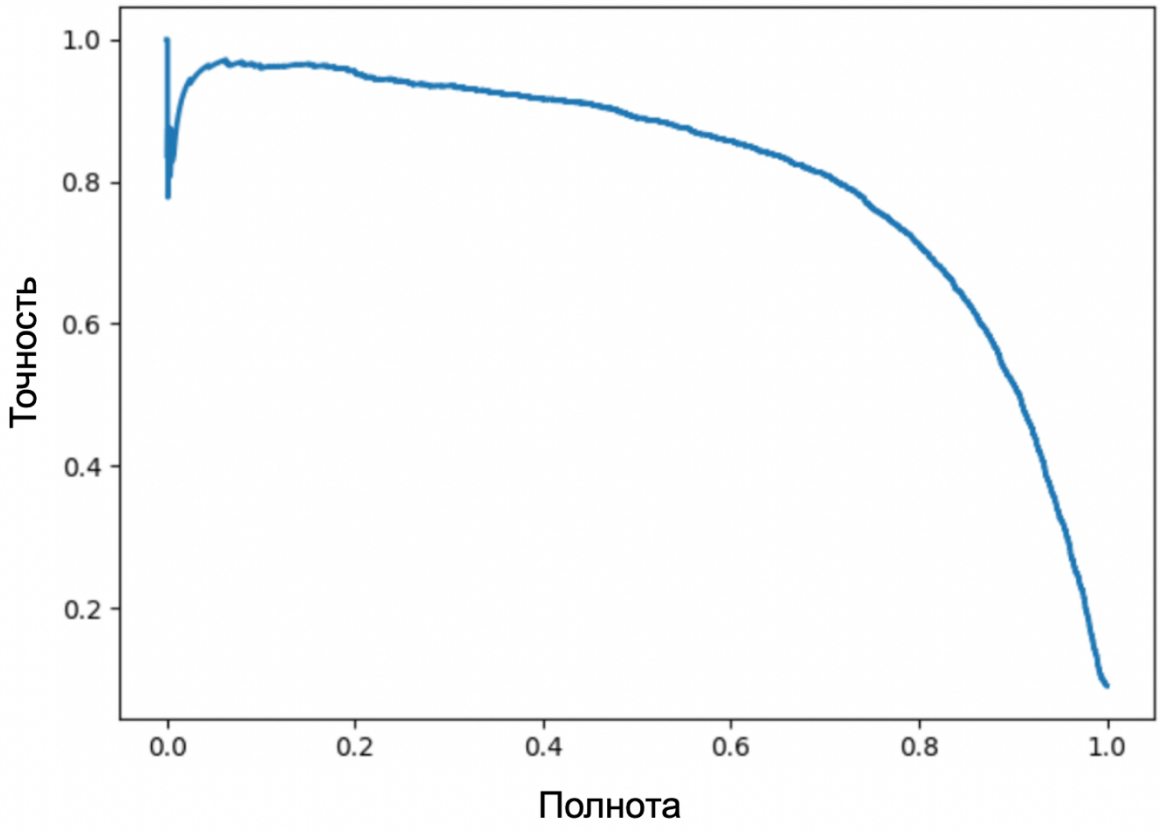

Рисунок 3 - Кривая точности и полноты

2. Проектирование и моделирование

2.1 Анализ набора данных

В первую очередь, прежде чем строить классификационную модель, нужно проанализировать наши данные, чтобы понять с чем мы имеем дело. Воспользуемся набором данных с Kaggle[2].

Для анализа и преобразования табличных данных воспользуемся библиотекой Pandas[3].

#	Column	Non-Null Count	Dtype
0	incident_id	239677 non-null	int64
1	date	239677 non-null	object
2	state	239677 non-null	object
3	city_or_county	239677 non-null	object
4	address	223180 non-null	object
5	n_killed	239677 non-null	int64
6	n_injured	239677 non-null	int64
7	incident_url	239677 non-null	object
8	source_url	239209 non-null	object
9	incident_url_fields_missing	239677 non-null	bool
10	congressional_district	227733 non-null	float64
11	gun_stolen	140179 non-null	object
12	gun_type	140226 non-null	object
13	incident_characteristics	239351 non-null	object
14	latitude	231754 non-null	float64
15	location_description	42089 non-null	object
16	longitude	231754 non-null	float64
17	n_guns_involved	140226 non-null	float64
18	notes	158660 non-null	object
19	participant_age	147379 non-null	object
20	participant_age_group	197558 non-null	object
21	participant_gender	203315 non-null	object
22	participant_name	117424 non-null	object
23	participant_relationship	15774 non-null	object
24	participant_status	212051 non-null	object
25	participant_type	214814 non-null	object
26	sources	239068 non-null	object
27	state_house_district	200905 non-null	float64
28	state_senate_district	207342 non-null	float64

dtypes: bool(1), float64(6), int64(3), object(19)

Рисунок 4 - Краткая информация о колонках набора данных

Из рисунка 4 видно, что в наборе данных находятся сырые данные, которые перед использованием необходимо подготовить. В наборе данных присутствует одна булевая колонка, 9 численных колонок и 19 строковых. Были рассмотрены колонки, которые потенциально могут быть полезны в дальнейшем:

- incident_id – номер строчки
- date – дата инцидента
- state – штат, где произошел инцидент
- city_or_country – страна или город, где произошел инцидент
- n_killed – количество убитых людей
- n_injured – количество пострадавших людей
- latitude – географическая широта
- longitude – географическая долгота
- participant_age – возраст участника преступления
- participant_gender – пол участника преступления

Все остальные колонки не являются полезными для дальнейшего использования в анализе данных.

2.2 Анализ временных рядов

Из рисунка 5 был сделан вывод, что в наборе данных присутствуют данные с 2013 по 2018 год. Но информации по 2013 году практически не представлено, а по 2018 году информации гораздо меньше, чем по другим годам.

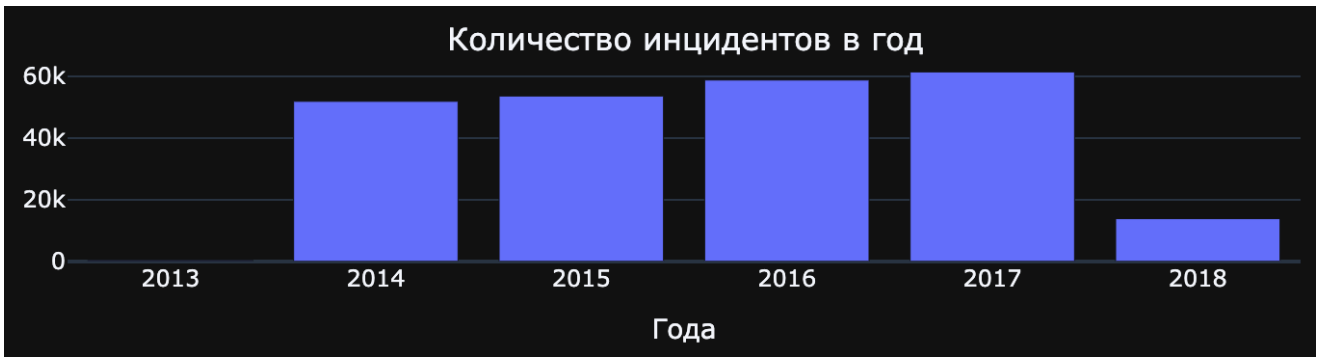

Рисунок 5 - Количество инцидентов в год

Из рисунка 6 был сделан вывод, что летом, в частности в июле и августе, происходит наибольшее количество инцидентов. В феврале меньше всего инцидентов, связано это с тем, что февраль самый короткий месяц в году.

Рисунок 6 - Среднее количество инцидентов в месяц

Из рисунка 7 был сделан вывод, что в выходные количество инцидентов увеличивается.

Рисунок 7 - Среднее количество инцидентов в день

На рисунке 8 представлен временной ряд количества убитых и раненных людей по дням. Был сделан вывод, что по 2013 году данных недостаточно, и

он не будет присутствовать в дальнейшем исследовании. По 2018 году можно понять, что информации достаточно, но в нем представлены не все месяцы, поэтому в анализе временных рядов по годам он представлен не будет.

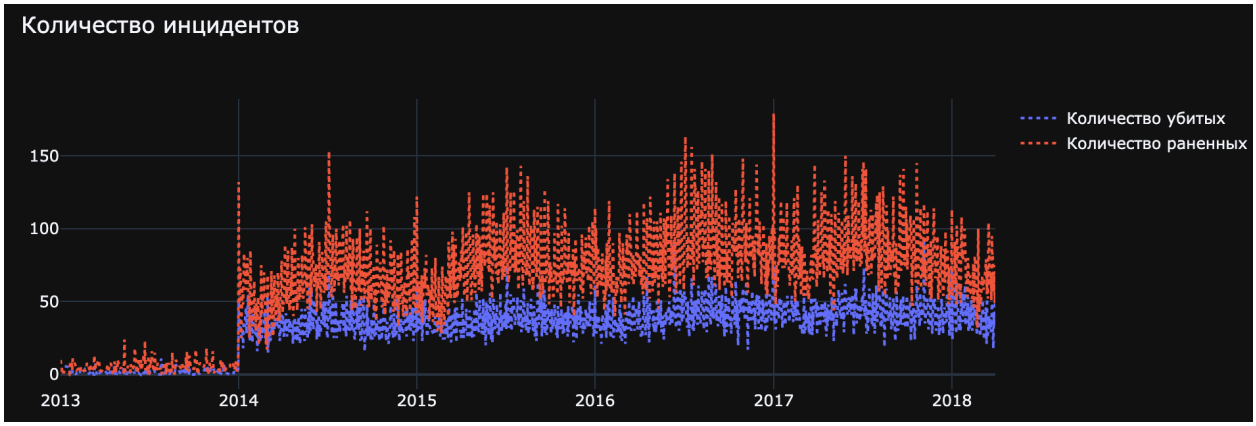

Рисунок 8 - Временной ряд по количеству инцидентов

Из рисунков 9, 10, 11, 12 был сделан вывод, что самый опасный месяц в году – июль. А самый опасный день – 4 июля:

- 4 июля 2014 – количество убитых: 48, количество раненых: 92, суммарное количество инцидентов за день: 192. При условии, что в 2014 – среднее количество убитых: 34, среднее количество раненых: 63, среднее количество инцидентов в день: 142.
- 4 июля 2015 – количество убитых: 60, количество раненых: 142, суммарное количество инцидентов за день: 211. При условии, что в 2015 – среднее количество убитых: 37, среднее количество раненых: 74, среднее количество инцидентов в день: 147.
- 4 июля 2016 – количество убитых: 52, количество раненых: 163, суммарное количество инцидентов за день: 224. При условии, что в 2016 – среднее количество убитых: 41, среднее количество раненых: 83, среднее количество инцидентов в день: 160.
- 4 июля 2017 – количество убитых: 62, количество раненых: 146, суммарное количество инцидентов за день: 248, При условии, что в 2014 – среднее количество убитых: 42, среднее количество раненых: 84, среднее количество инцидентов в день: 168.

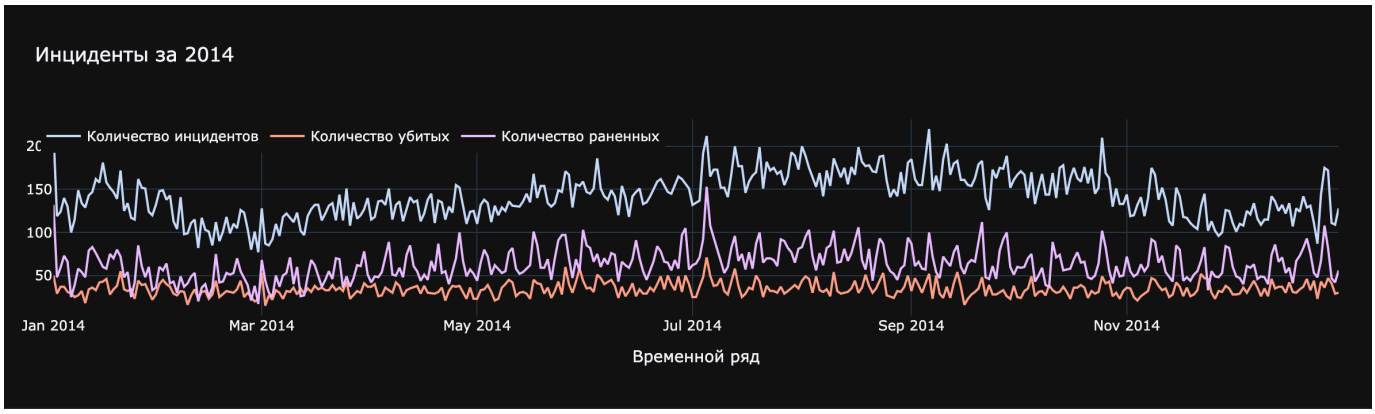

Рисунок 9 - Временной ряд по 2014 году

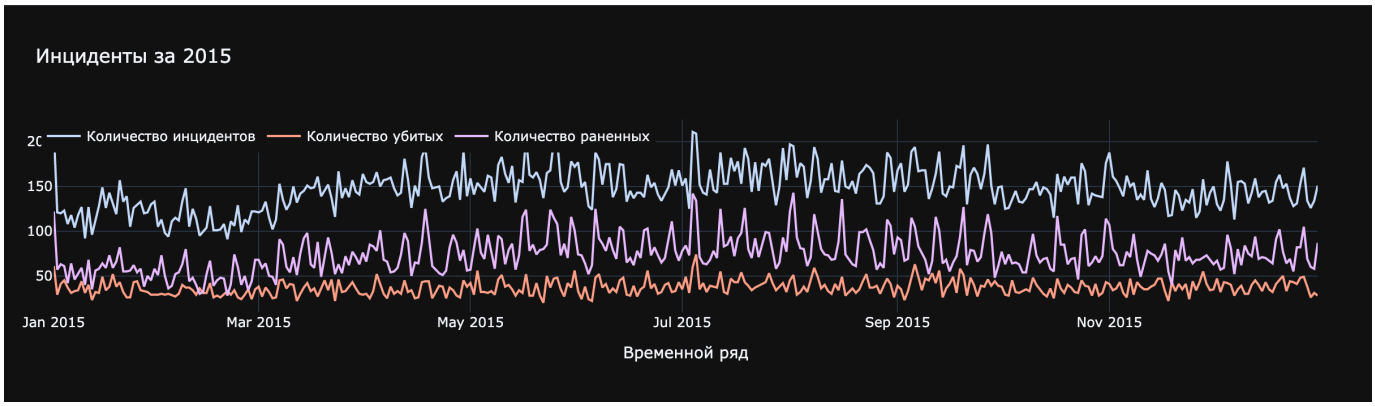

Рисунок 10 - Временной ряд по 2015 году

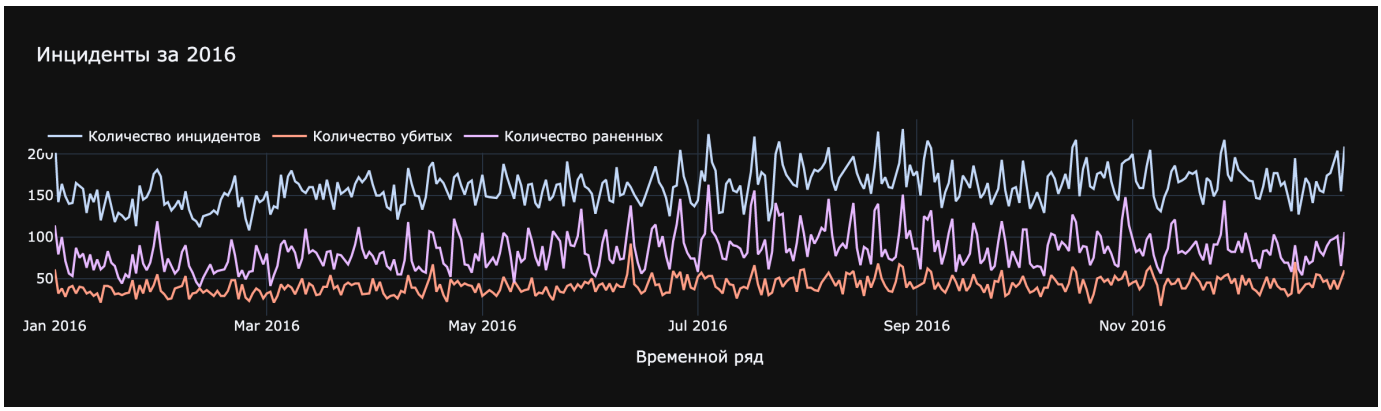

Рисунок 11 - Временной ряд по 2016 году

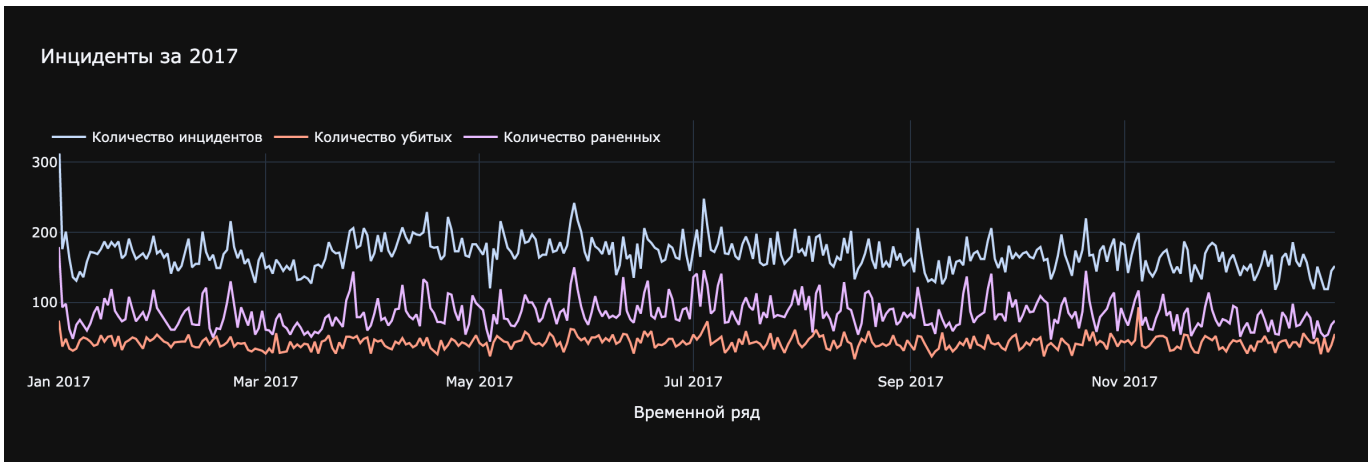

Рисунок 12 - Временной ряд по 2017 году

Отдельно был рассмотрен 2018 год. Из рисунка 13 можно увидеть, что информация представлена по апрель 2018 года. Самый опасных день произошел 15 февраля. В этот день произошло нападение в школе, где погибло больше 17 человек.

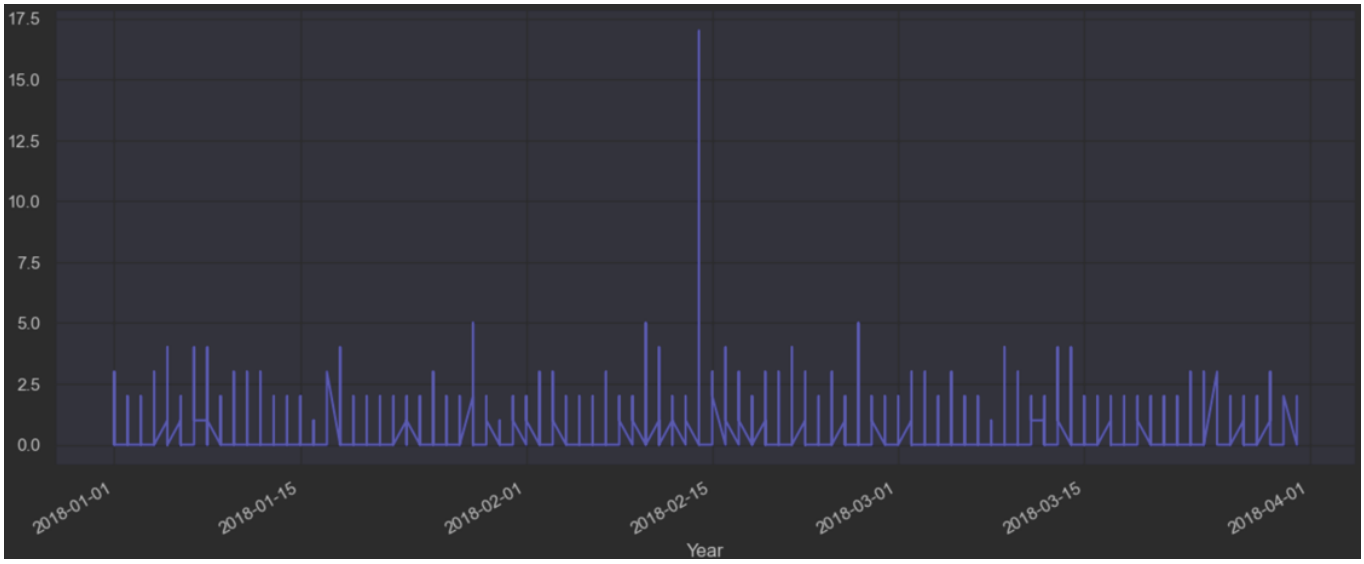

Рисунок 13 - Временной ряд по 2018 году

Дальше были рассмотрены тренды инцидентов. На рисунке 14 можно увидеть тренд возрастания количества инцидентов по месяцам.

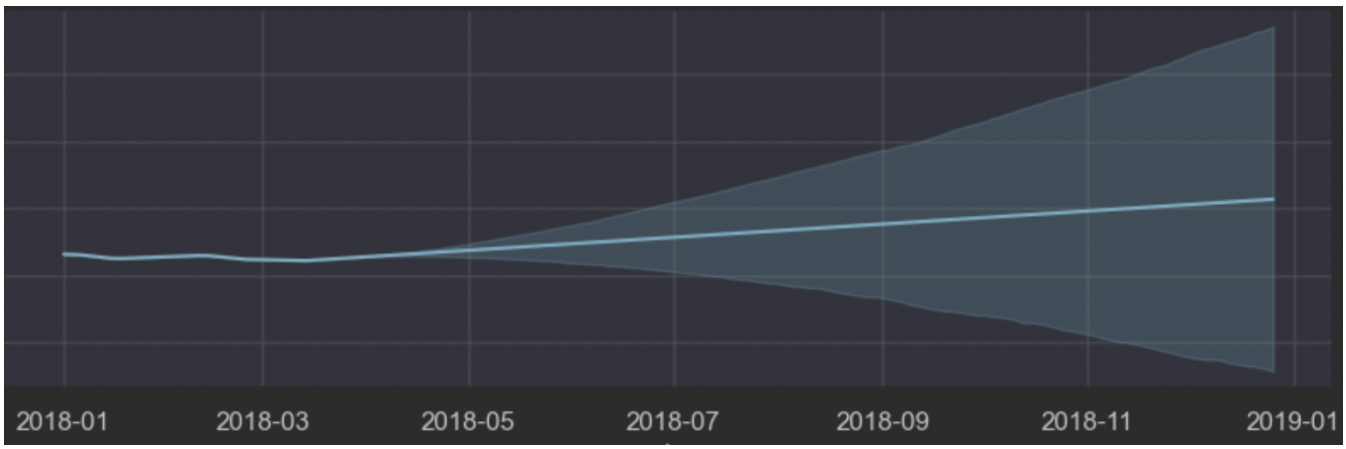

Рисунок 14 - Анализ тренда инцидентов по месяцам

На рисунке 15 видим, что в выходные дни увеличивается количество инцидентов, а со вторника по четверг количество инцидентов идет ровно.

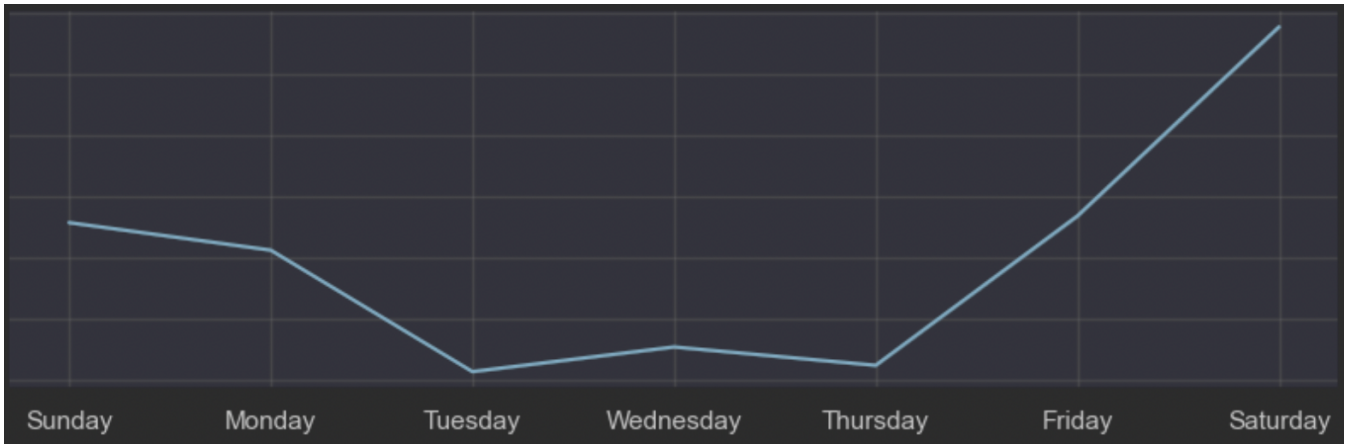

Рисунок 15 - Анализ тренда инцидентов по дням недели

2.3 Преобразование данных

Для правильной работы классификационной модели необходимо очистить данные от выбросов, дубликатов, ненужной информации и нулевых значений, а также преобразовать данные, которые хранятся в неподходящем виде.

В самом начале нужно избавиться от информации, которая нам не будет полезна в дальнейшем. Оставим только колонки: incident_id, date, state,

city_or_country, n_killed, n_injured, latitude, longitude, participant_age, participant_gender.

Так же необходимо избавиться от нулевых значений и дубликатах в полезных для нас значениях.

После первичной обработки необходимо поработать с колонками по отдельности, чтобы привести их к виду, подходящему для классификации.

Начнем с колонок n_injured и n_killed. Эти колонки являются для нас целевыми и их необходимо преобразовать. Каждую из этих колонок мы преобразуем таким образом, что при наличии в строке колонки цифры, означающей количество человек, мы преобразуем ее в булевое значение “истина”. Если в строке колонки находится ноль, то есть никто не пострадал, то мы преобразуем ее в булевое значение “ложь”. Получим колонки is_injured и is_killed. После преобразования объединим получившиеся колонки в одну с помощью логического “или” и получим колонку “is_injured_or_killed”.

Колонку date, содержащую в себе записи вида – год-месяц-день, разделим на 3 отдельные колонки для года, месяца и дня. Получим колонки year, month, day.

Для преобразования колонки city_or_country воспользуемся следующим алгоритмом:

- Создадим пустой словарь, где по названию города будет выдаваться его номер
- Если наш город уже лежит в словаре, то мы заменяем в колонке его название на его числовое значение из словаря
- Если наш город не лежит в словаре, то добавляем его в наш словарь, а его числовое значение будет равно длине словаря. После также заменяем в колонке его название на его числовое значение из словаря

Получим преобразованную колонку city_code.

Колонка participant_age состоит из записей вида – номер участника::возраст участника||следующий участник. Для преобразования колонки необходимо получить возраст участников и взять их среднее арифметическое значение. Получим колонку average_age.

Удалим из записи в наборе данных, где в колонке average_age средний возраст человека больше 83 и меньше 13.

Колонка participant_gender состоит из записей вида – номер участника::пол участника||следующий участник. Для преобразования колонки получим пол участников, посчитаем количество мужчин и количество женщин, запишем значение 1 если количество мужчин больше количества женщин. И запишем 0 если количество мужчин меньше количества женщин. Получим колонку gender_bool.

Для преобразования колонки state воспользуемся следующим алгоритмом:

- Создадим пустой строковой массив
- Добавим в него все названия штатов
- Преобразуем название город в записи на его номер в нашем массиве

Получим преобразованную колонку state_code.

После преобразования данных, удалим колонки date, state, city_or_country, n_killed, n_injured, participant_age, participant_gender, так как мы получили с помощью этих колонок получили другие подходящие для классификации колонки.

2.4 Работа с выбросами

Чтобы классификационная модель работала корректно также необходимо убрать выбросы из данных и сделать их более симметричными.

Для определения симметричности данных необходимо воспользоваться гистограммой для данных, а также некоторыми определениями из математической статистики.

Для правильного разбиения на интервалы воспользуемся методом, отображающим частоту переменных не высотой столбца в гистограмме, а его площадью. Площадь находится как площадь прямоугольника, где длину интервала умножают на высоту столбца. Полученная площадь – частота переменной, а ее высота – плотность частоты. Для оценки количества попавших значений в интервал будем брать два значения и искать площадь плотности гистограммы между ними.

Также воспользуемся метриками локации данных, а именно средним арифметическим значением и стандартным отклонением.

Для распределения используется правило 3 стандартным отклонений, что практически все значения находятся в промежутке $(\mu - 3\sigma, \mu + 3\sigma)$. Все данные, находящиеся вне интервала, являются выбросами.

Была рассмотрена колонка average_age. На рисунке 16 представлено распределение данных в колонке average_age. На рисунке красным пунктиром показаны границы интервала, где будут находится нужные для нас данные, а черным пунктиром показано среднее значение. Можно увидеть, что набор данных скошен вправо. Чтобы сделать его более симметричным воспользуемся логарифмированием всех данных из этой колонки.

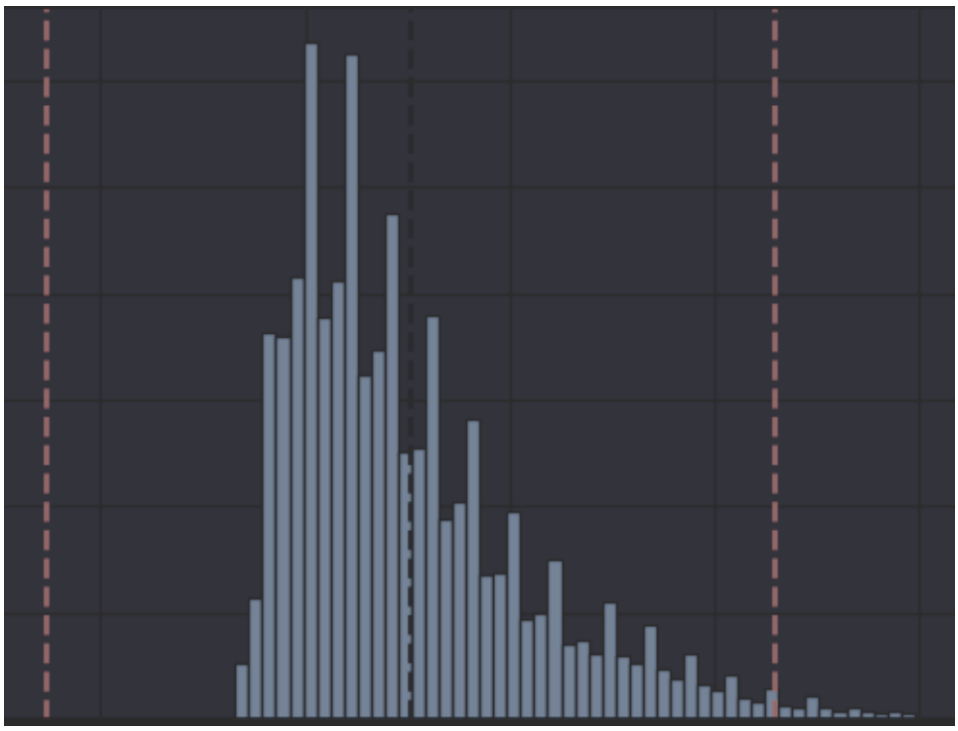

Рисунок 16 - Распределение данных в колонке average_age

На рисунке 17 представлено распределение данных после работы с выбросами. Набор данных стал более симметричным, выбросы за границей интервала исчезли.

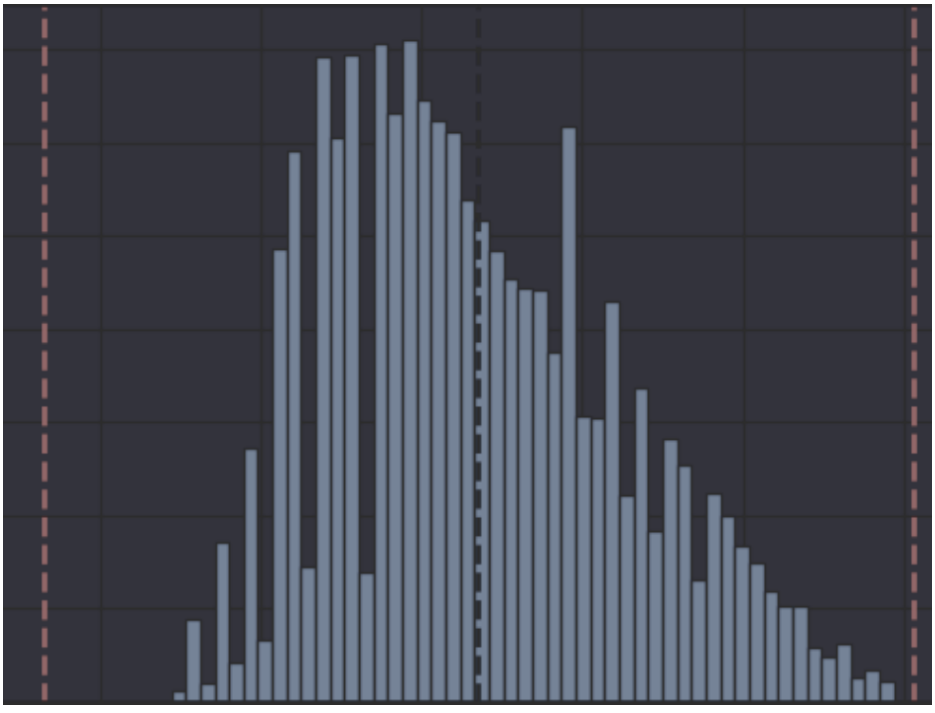

Рисунок 17 - Распределение данных после работы с выбросами

3. Реализация

3.1 Стек технологий

При реализации классификационной модели использовался следующий стек технологий:

1. Язык программирования Python.
2. Plotly и Matplotlib – библиотеки, позволяющие визуализировать данные.
3. Numpy[4] – библиотека, применимая для математических вычислений.

3.2 Реализация логистической регрессии

Сама логистическая регрессия будет классом. В листинге 1 представлена функция fit, с помощью которой положим данные в наш класс и произведем обучение с учителем.

Листинг 1 – функция тренировки логистической модели

```
def fit(self, x_train, y_train, learning_rate, num_iterations):

x = self.normalize(
    np.array(
        copy.deepcopy(
            x_train
        )
    )
)

y = np.array(
    copy.deepcopy(
        y_train
    )
)

self.weights, self.bias = self.init_w_b(
    size=x_train.shape[0],
    learning_rate=learning_rate
)

losses = []
```

```
        for i in range(num_iterations):

            loss, dw, db = self.grad_descend(
                x_train=x,
                y_train=y
            )

            losses.append(loss)

            self.update_w_b(
                dw=dw,
                db=db,
                learning_rate=learning_rate,
            )
```

Для обучения модели был использован градиентный спуск, где функция потерь высчитывается по формуле кросс-энтропии, представленная в листинге 2.

Листинг 2 – градиентный спуск

```
        def grad_descend(self, x_train, y_train):

            z = np.dot(self.weights.T, x_train) + self.bias

            y_head = self.sigmoid(z)

            one_losses = -y_train * np.log(y_head)

            zero_losses = (1 - y_train) * np.log(1 - y_head)

            loss_diff = one_losses - zero_losses

            loss = (np.sum(loss_diff)) / x_train.shape[1]

            dw = (np.dot(x_train, (y_head - y_train).T)) / x_train.shape[1]

            db = np.sum(y_head - y_train) / x_train.shape[1]

            return loss, dw, db
```

После обучения положим тестовые данные в функции predict, представленная в листинге 3. Функция predict выдает ответ 1 или 0 в зависимости от вероятности. Точность классификационной модели равна 0.79.

Листинг 3 – функция классификации

```
    def predict(self, x_test):

        x = self.normalize(
            np.array(
                copy.deepcopy(
                    x_test
                )
            )
        )

        z = self.sigmoid(
            z=np.dot(self.weights.T, x) + self.bias
        )

        predict = np.zeros((1, x.shape[1]))

        for i in range(z.shape[1]):

            if z[0, i] <= 0.5:
                predict[0, i] = 0
            else:
                predict[0, i] = 1

        return predict
```

Для анализа точности работы классификационной модели нам понадобятся метрик, для кривой точность-полнота необходим не результат классификации, а полученная вероятность, которая и будет являться нашей точностью. Функция predict_proba, представленная в листинге 3, выдает вероятность правильного ответа.

Листинг 4 – функция, предсказывающая вероятность

```
    def predict_proba(self, x_test):



        x = self.normalize(
            np.array(
                copy.deepcopy(
                    x_test
                )
            )
        )

        z = self.sigmoid(
            z=np.dot(self.weights.T, x) + self.bias
        )
        return z
```

3.3 Анализ классификационной модели с применением метрик

Проанализируем полученные данные с помощью метрик. После вычисление 4 типов ответов создадим матрицу ошибок, представленную на рисунке 18.

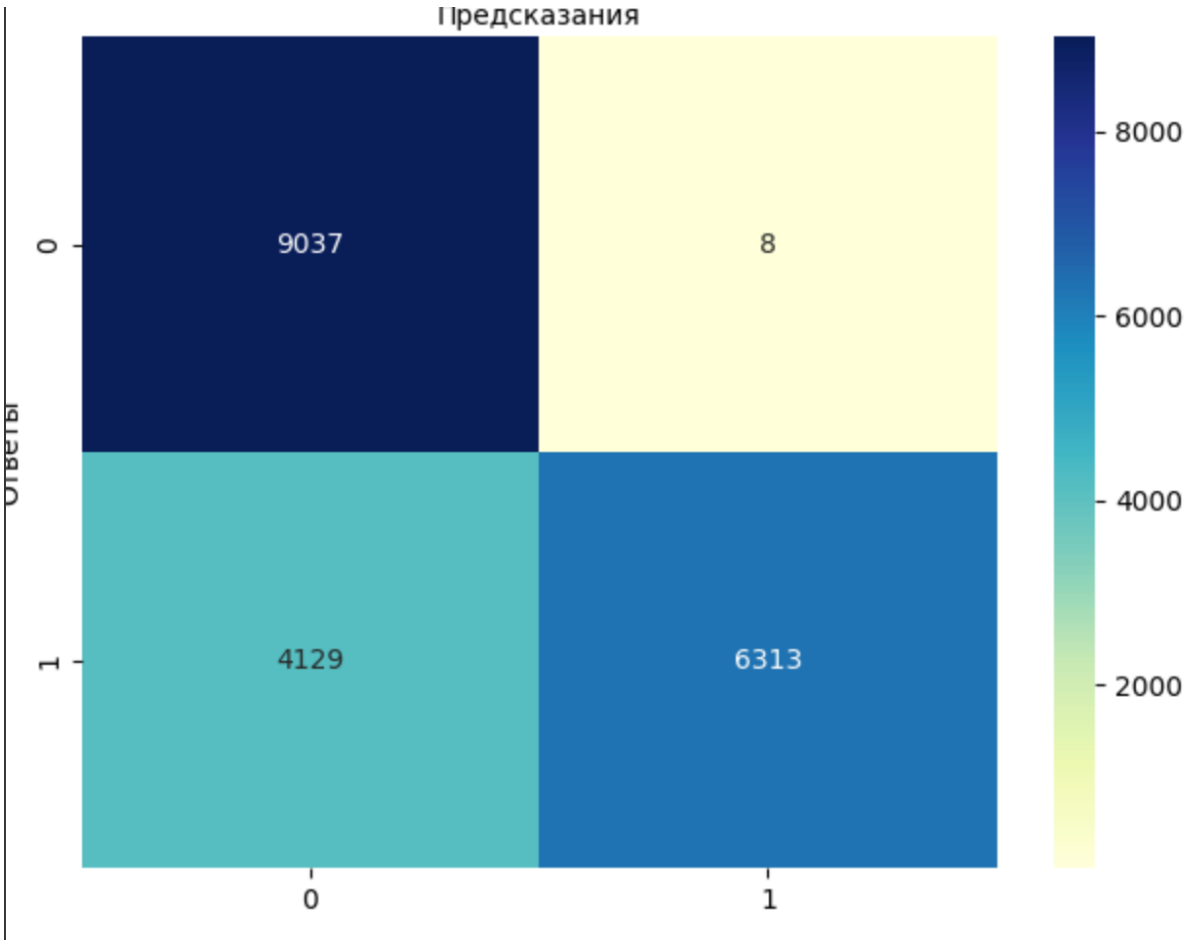

Рисунок 18 – Полученная матрица ошибок

Из матрицы видно, что модель отлично научилась определять положительный класс, где ложноположительные ответы являются погрешностью. Но отрицательный класс модель научилась определять с большой погрешностью, где истинно отрицательные ответы равные 9098, а ложноотрицательные ответы равны 4044. Точность определения отрицательных ответов составила 0.69.

На рисунке 19 представлена получившаяся кривая точности и полноты. Из графика видно, что при увеличении полноты снижается точность. А это значит, что чем больше увеличивается доля правильно положительно

определенных ответов среди всех действительно положительных ответов, тем чаще модель начинает ошибаться и выдавать отрицательные ответы за положительные, тем самым уменьшая точность. В идеальном случае кривая должна проходить через точку (1,1), но на реальных данных такое не встречается. Кривая показывает, что чем меньше неправильно определенных отрицательных ответов, тем больше модель выявляет неправильно определенных положительных ответов. Так же кривая показывает, что модель правильно давала верную классификацию пока доля правильно определенных положительных ответов ко всем действительно положительным ответам не стала больше 0.6, что показывает высокую точность классификационный модели для неискусственно созданных данных.

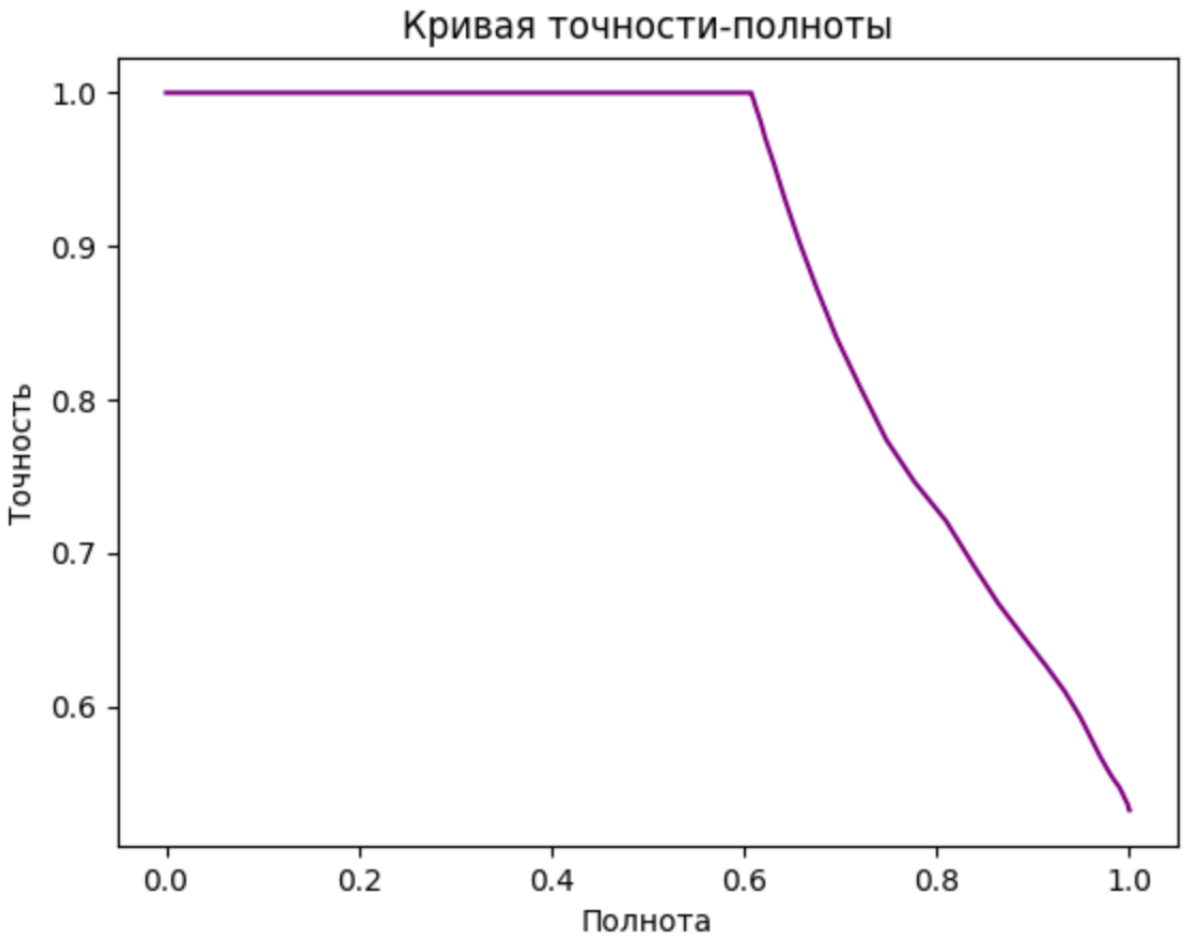

Рисунок 19 – Полученная кривая точности и полноты

4. Тестирование

Тестирование проводилось на оборудовании, оснащенном двух ядерным процессором Intel Core i5 с тактовой частотой 1.6 ГГц, встроенной графикой Intel HD Graphics 6000 1536 МБ и оперативной памятью 8 ГБ 1600 MHz DDR3 на операционной системе MacOS Big Sur.

Целью тестирования является определение корректности работы программы, определение возможностей работы программы в режиме реального времени, а также сравнение скорости с библиотечными реализациями.

Тестирование логистической модели производилось путем сравнения написанной самостоятельно модели с моделью из библиотеки Sklearn[5].

Сравним написанную модель и библиотечную. Точность собственной модели составила 0.7914, при этом точность библиотечной модели составила 0.7934. Разница в точности между моделями не значительна. Скорость обучения собственной модели составила 4.35 секунд, при этом скорость обучения библиотечной модели составила 3.89. Разница в скорости составляет почти полторы секунды. Разница меньше секунды, приходящаяся на обучение, не является существенной, так как обучение модели происходит всего один раз.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы был произведен численный анализ данных о результативности преступлений с применением оружия.

Также были реализованы классификационная модель, предсказывающая результативность преступлений и метрики анализа модели. Классификационная модель по средству сравнения с логистической регрессией из библиотеки Sklearn показала высокую точность и скорость классификации. Разница в их точности находится в пределах погрешности, а разница в скорости является несущественной при единоразовой обработке большого количества данных. Метрики качества показали, что классификационная модель с высокой точностью смогла классифицировать положительные ответы, но точность модели сильно падает при классификации отрицательных результатов. В ходе анализа полученной модели был сделан вывод, что модель с высокой точностью выявляет преступления с жертвами и с низкой – без них.

В ходе тестирования был сделан вывод, что сама классификационная модель реализована без ошибок, и проблема классификации отрицательного результата связана с набором данных.

В дальнейшем можно произвести дополнительные преобразования данных, увеличить количество входных параметров, расширить набор данных для улучшения прогноза преступлений без жертв. А также в перспективах исследования стоит сравнить точность работы логистической регрессии с другими моделями, например, с классификационным лесом, чтобы выявить самую эффективную модель для рассмотренного набора данных.

СПИСОК ЛИТЕРАТУРЫ

1. Учебник по машинному обучению Яндекс – URL: https://academy.yandex.ru/handbook/ml/article/metriki-klassifikacii-i-regressii (дата обращения: 23.12.2022))

2. Kaggle dataset gun-violence – URL: https://www.kaggle.com/datasets/jameslko/gun-violence-data (дата обращения: 23.12.2022)

3. Pandas documentation – URL: https://pandas.pydata.org/docs/development/index.html#development (дата обращения: 23.12.2022)

4. Numpy documentation - URL: https://numpy.org/devdocs/user/absolute_beginners.html (дата обращения 23.12.2022)

5. Sklearn Regression documentation – URL: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning (дата обращения: 23.12.2022)

ПРИЛОЖЕНИЕ А

Листинг А.1 – анализ данных

```
import copy
import time

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

pd.set_option('display.max_columns', None, 'display.width', -1)
gun_violence_df = pd.read_csv('gun-violence-data_01-2013_03-2018.csv')

gun_violence_df = gun_violence_df.dropna(subset='n_killed')

gun_violence_df = gun_violence_df.dropna(subset='n_injured')

gun_violence_df.drop_duplicates()

gv_df_clean = gun_violence_df.drop(columns=['incident_url', 'source_url',
'incident_url_fields_missing',
                                            'notes', 'participant_name',
'participant_type',
                                            'sources', 'gun_type',
'participant_age_group'])

gv_df_clean = gv_df_clean.dropna(subset=['state', 'participant_age',
'participant_gender',
                                         'date', 'state_house_district',
'state_senate_district',
                                         'congressional_district',
'latitude', 'longitude', ])

gv_df_clean = gv_df_clean.drop(columns=['gun_stolen', 'address',
'incident_characteristics', 'location_description'])

gv_df_clean = gv_df_clean.drop(columns=['n_guns_involved',
'participant_relationship', ])

gv_df_clean['is_killed'] = gv_df_clean.apply(lambda row: 1.0 if
row['n_killed'] > 0 else 0.0, axis=1)

gv_df_clean['is_injured'] = gv_df_clean.apply(lambda row: 1.0 if
row['n_injured'] > 0 else 0.0, axis=1)

# gv_df_clean['date'] = pd.to_datetime(gv_df_clean['date'])
#
# print(gv_df_clean['date'].year)
gv_df_clean['year'] = gv_df_clean.apply(lambda row:
float(pd.to_datetime(row['date']).year), axis=1)

gv_df_clean['month'] = gv_df_clean.apply(lambda row:
float(pd.to_datetime(row['date']).month), axis=1)

gv_df_clean['day'] = gv_df_clean.apply(lambda row:
float(pd.to_datetime(row['date']).day), axis=1)

print(gv_df_clean)

city_codes = {}
```

Продолжение листинга А.1

```
def add_city_code(city_str):
    if city_codes.get(city_str) is not None:
        return city_codes[city_str]
    else:
        city_codes[city_str] = len(city_codes)
        return city_codes[city_str]


gv_df_clean['city_code'] = gv_df_clean.apply(lambda row:
float(add_city_code(row['city_or_county'])), axis=1)


def parse_age(participant_age_str):
    ages_str = participant_age_str.replace('||', '|').split('|')
    ages = []
    for age_str in ages_str:
        right = age_str.replace('::', ':').split(':')
        ages.append(int(right[1]))
    sum = 0
    for i in range(0, len(ages)):
        sum += ages[i]
    return sum / len(ages)


def parse_gender(part_gender_str):
    genders_str = part_gender_str.replace('||', '|').split('|')
    genders = []
    for gender_str in genders_str:
        right = gender_str.replace('::', ':').split(':')
        genders.append(right[1])

    n_male = 0
    n_female = 0
    for i in range(0, len(genders)):
        if genders[i] == 'Male':
            n_male += 1
        elif genders[i] == 'Female':
            n_female += 1
    return 'Male' if n_male > n_female else 'Female'


gv_df_clean['average_age'] = gv_df_clean.apply(lambda row:
float(parse_age(row['participant_age'])), axis=1)
gv_df_clean['average_gender'] = gv_df_clean.apply(lambda row:
parse_gender(row['participant_gender']), axis=1)

gv_df_clean = gv_df_clean.drop(
    columns=['participant_age', 'participant_gender', 'participant_status',
'n_killed', 'n_injured'])

state_names = ["Alabama", "Alaska", "Arizona", "Arkansas", "California",
"Colorado", "Connecticut",
               "District of Columbia", "Delaware", "Florida", "Georgia",
"Hawaii", "Idaho", "Illinois", "Indiana",
               "Iowa", "Kansas", "Kentucky", "Louisiana", "Maine",
"Maryland", "Massachusetts", "Michigan", "Minnesota",
               "Mississippi", "Missouri", "Montana", "Nebraska", "Nevada",
"New Hampshire", "New Jersey", "New Mexico",
               "New York", "North Carolina", "North Dakota", "Ohio",
"Oklahoma", "Oregon", "Pennsylvania",
```

Продолжение листинга А.1

```
"Rhode Island", "South Carolina", "South Dakota", "Tennessee", "Texas",
"Utah", "Vermont", "Virginia",
               "Washington", "West Virginia", "Wisconsin", "Wyoming"]


def find_state_code(state):
    for i in range(0, len(state_names)):
        if state.upper().replace(' ', '') == state_names[i].upper().replace('
', ''):
            return i


gv_df_clean['state_code'] = gv_df_clean.apply(lambda row:
float(find_state_code(row['state'])), axis=1)
gv_df_clean['gender_bool'] = gv_df_clean.apply(lambda row: 1.0 if
row['average_gender'] == 'Male' else 0.0, axis=1)
gv_df_clean['is_injured_or_killed'] = gv_df_clean.apply(
    lambda row: 1.0 if row['is_killed'] or row['is_injured'] else 0.0,
    axis=1)

gv_ones = gv_df_clean[gv_df_clean['is_injured_or_killed'] !=
0.0].sample(frac=0.5, random_state=12345)

gv_df_clean = pd.concat([gv_ones] +
[gv_df_clean[gv_df_clean['is_injured_or_killed'] == 0.0]])

filter_age = gv_df_clean['average_age'] < 80.0
filter_age_2 = gv_df_clean['average_age'] > 13.0
gv_df_clean = gv_df_clean[filter_age & filter_age_2]

from sklearn.utils import shuffle

gv_df_clean = shuffle(gv_df_clean)

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

data_columns = ['state_code', 'average_age', 'gender_bool',
                'year', 'month', 'day',
                'city_code', 'is_injured', 'state_house_district',
                'state_senate_district', 'congressional_district',
                'latitude', 'longitude']

df_target = gv_df_clean.is_injured_or_killed
df_data = gv_df_clean[data_columns]
x_train, x_test, y_train, y_test = train_test_split(df_data, df_target,
test_size=0.25, random_state=125)
```

Листинг А.2 – логистическая регрессия и метрики

```
class MyLogisticRegression:

    def __init__(self):

        self.bias = None
        self.weights = None

    def fit(self, x_train, y_train, learning_rate, num_iterations):
```

Продолжение листинга А.2

```
        x = self.normalize(
                np.array(
                    copy.deepcopy(
                        x_train
                    )
                )
            )

            y = np.array(
                copy.deepcopy(
                    y_train
                )
            )

            self.weights, self.bias = self.init_w_b(
                size=x_train.shape[0],
                learning_rate=learning_rate
            )

            losses = []

            for i in range(num_iterations):

                loss, dw, db = self.grad_descend(
                    x_train=x,
                    y_train=y
                )

                losses.append(loss)

                self.update_w_b(
                    dw=dw,
                    db=db,
                    learning_rate=learning_rate,
                )

                if i % 10 == 0:
                    print(f"Loss on {i}: {loss}")

    def grad_descend(self, x_train, y_train):

        z = np.dot(self.weights.T, x_train) + self.bias

        y_head = self.sigmoid(z)

        one_losses = -y_train * np.log(y_head)
        zero_losses = (1 - y_train) * np.log(1 - y_head)

        loss_diff = one_losses - zero_losses
        loss = (np.sum(loss_diff)) / x_train.shape[1]

        dw = (np.dot(x_train, (y_head - y_train).T)) / x_train.shape[1]
        db = np.sum(y_head - y_train) / x_train.shape[1]

        return loss, dw, db

    def predict_proba(self, x_test):

        x = self.normalize(
            np.array(
                copy.deepcopy(
```

Продолжение листинга А.2

```
            x_test
                )
            )
        )

        z = self.sigmoid(
            z=np.dot(self.weights.T, x) + self.bias
        )
        return z

    def predict(self, x_test):

        x = self.normalize(
            np.array(
                copy.deepcopy(
                    x_test
                )
            )
        )

        z = self.sigmoid(
            z=np.dot(self.weights.T, x) + self.bias
        )

        predict = np.zeros((1, x.shape[1]))

        for i in range(z.shape[1]):

            if z[0, i] <= 0.5:
                predict[0, i] = 0
            else:
                predict[0, i] = 1

        return predict

    def accuracy(self, y_test, predict):

        y = np.array(
            copy.deepcopy(
                y_test
            )
        )

        print(f"test accuracy: {100 - np.mean(np.abs(predict - y)) * 100}")

    def init_w_b(self, size, learning_rate):

        return np.full((size, 1), learning_rate), 0.0

    def update_w_b(self, dw, db, learning_rate):

        self.weights -= learning_rate * dw
        self.bias -= learning_rate * db

    def normalize(self, x):

        for i in range(x.shape[0]):
            x[i] = (x[i] - x[i].mean(axis=0)) / x[i].std(axis=0)

        return x
```

Продолжение листинга А.2

```
    def sigmoid(self, z):

        return 1 / (1 + np.exp(-z))


class Metrics:

    def __init__(self):
        self.fp = 0
        self.fn = 0
        self.tp = 0
        self.tn = 0
        self.precision_positive = 0
        self.precision_negative = 0
        self.recall_positive = 0
        self.recall_negative = 0

    def confusion_matrix(self, actual_values, predicted_values):

        for actual_value, predicted_value in zip(actual_values,
predicted_values):

            if predicted_value == actual_value:
                if predicted_value == 1:
                    self.tp += 1
                else:
                    self.tn += 1

            else:
                if predicted_value == 1:
                    self.fp += 1
                else:
                    self.fn += 1

        return [
            [self.tn, self.fp],
            [self.fn, self.tp]
        ]

    def accuracy(self, predicted_values):
        return (self.tp + self.tn) / predicted_values.shape[0]

    def precision_recall_score(self):
        all_predicted_positives = self.tp + self.fp
        self.precision_positive = self.tp / all_predicted_positives

        all_predicted_negatives = self.tn + self.fn
        self.precision_negative = self.tn / all_predicted_negatives

        all_actual_positive = self.tp + self.fn
        self.recall_positive = self.tp / all_actual_positive

        all_actual_negative = self.tn + self.fp
        self.recall_negative = self.tn / all_actual_negative
        return [
            [self.precision_positive, self.precision_negative],
            [self.recall_positive, self.recall_negative]
        ]

    def f1_score(self):
        f1_pos = 2 * (self.precision_positive * self.recall_positive) \
```

Продолжение листинга А.2

```
                / (self.precision_positive + self.recall_positive)

        f1_neg = 2 * (self.precision_negative * self.recall_negative) \
                 / (self.precision_negative + self.recall_negative)
        return [f1_pos, f1_neg]

    def print_cnf(self):
        class_names = [0, 1]

        fig, ax = plt.subplots()

        tick_marks = np.arange(len(class_names))

        plt.xticks(tick_marks, class_names)

        plt.yticks(tick_marks, class_names)

        matr = np.array([
            [self.tn, self.fp],
            [self.fn, self.tp]
        ])

        sns.heatmap(
            pd.DataFrame(matr),
            annot=True,
            cmap="YlGnBu",
            fmt='g'
        )

        ax.xaxis.set_label_position("top")

        plt.tight_layout()

        plt.title('Confusion matrix', y=1.1)

        plt.ylabel('Actual label')

        plt.xlabel('Predicted label')
        plt.show()

    def print_pr_curve(self, y_test, probs):
        pr_scores = []
        recall_scores = []

        prob_thresholds = np.linspace(0, 1, num=100)

        for p in prob_thresholds:

            y_test_preds = []

            for prob in probs:
                if prob > p:
                    y_test_preds.append(1)
                else:
                    y_test_preds.append(0)

            pres, rec = self.calc_pr_rec_for_proba(y_test, y_test_preds)
            if pres == float('inf'):
                break
            pr_scores.append(pres)
            recall_scores.append(rec)
```

Продолжение листинга А.2

```
        self.print_curve(pr_scores, recall_scores)

    def print_curve(self, precisions, recalls):
        fig, ax = plt.subplots()
        ax.plot(recalls, precisions, color='purple')

        ax.set_title('Precision-Recall Curve')
        ax.set_ylabel('Precision')
        ax.set_xlabel('Recall')

        plt.show()

    def calc_pr_rec_for_proba(self, y_test, y_preds):

        tp = 0
        tn = 0
        fp = 0
        fn = 0

        for actual_value, predicted_value in zip(y_test, y_preds):

            if predicted_value == actual_value:
                if predicted_value == 1:
                    tp += 1
                else:
                    tn += 1

            else:
                if predicted_value == 1:
                    fp += 1
                else:
                    fn += 1

        all_predicted_positives = tp + fp
        if all_predicted_positives != 0:
            precision_positive = tp / all_predicted_positives
            all_actual_positive = tp + fn
            recall_positive = tp / all_actual_positive
            return precision_positive, recall_positive
        else:
            return float('inf'), float('inf')
```