**Санкт–Петербургский государственный университет**

***Полежаев Сергей Александрович***

# Выпускная квалификационная работа

# *Применение машинного обучения в задаче ценообразования*

Уровень образования: бакалавриат
Направление 02.03.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2015 «Программирование и информационные технологии»

Научный руководитель:
доцент, кафедра теории систем управления электрофизической аппаратурой, к.ф.-м.н. Козынченко Владимир Александрович

Рецензент:
профессор, кафедра теории управления, д.ф.-м.н. Котина Елена Дмитриевна

Санкт-Петербург
2019 г.

# Содержание

# Введение

Определение оптимальной цены на продаваемую продукцию является одной из важных задач в деятельности каждой компании. Современный рынок отличается высокой конкуренцией среди продавцов, которые борются за каждого покупателя. От правильности определения цены зависит количество клиентов, готовых купить данный товар, и прибыль, полученная от реализации продукции, а эти два фактора являются основными составляющими успеха компании.

Для выбора цены обычно проводится маркетинговое исследование с привлечением внешних специалистов и полным изучением рынка. Однако, у такого подхода есть несколько недостатков. Во-первых, это цена исследования. Проведение анализа может стоить слишком большую сумму для малого бизнеса. Во-вторых, при высокой волатильности рынка, большом числе постоянно меняющихся факторов проведения однократного анализа может быть недостаточно: исследование необходимо проводить постоянно. Даже при условии привлечения специалиста или трудоустройства его в компании крайне затруднительно учитывать все существенно влияющие на цену признаки и обновлять прогноз согласно актуальному положению рынка.

Различные подходы по автоматической оптимизации цены условно возможно разделить на две группы. В первую можно отнести такие методы как ARIMA, ARFIMA [1], GARCH[2], скрытые марковские модели [3], а во вторую группу - методы машинного обучения. Алгоритмы первой группы могут показывать приемлемые результаты и в каких-то случаях, возможно, даже демонстрировать лучшие показатели, чем методы второй группы. Однако, большинство из них предназначены для работы только с временными рядами, это сильно ограничивает и затрудняет работу с большим количеством признаков. Поэтому в данной работе в качестве основных были выбраны методы машинного обучения.

Машинное обучение - направление анализа данных, которое позволяет автоматизировать построение моделей. Это ветвь науки об искусственном интеллекте, основанная на идее того, что системы могут обучаться на

данных, обнаруживать структуры с минимальным вмешательством исследователя. Алгоритмы машинного обучения показывают высокие результаты на реальных задачах, поэтому в последние годы стали широко применяться в бизнесе и науке.

Различают два типа обучения:

1. Обучение с учителем (supervised learning) - способ машинного обучения, в ходе которого алгоритм обучается на основе размеченных данных. В качестве исходных данных выступает множество примеров с известными метками. Необходимо обучить алгоритм таким образом, чтобы на новых, неразмеченных данных, он смог бы предсказывать эту метку, т .е. относить пример к какому-либо классу (задача классификации) или предсказать скалярное значение целевой переменной (задача регрессии).

2. Обучение без учителя (unsupervised learning) - способ машинного обучения, при котором на вход алгоритму подаются неразмеченные данные. Как правило обучение без учителя применяется для задач кластеризации и обнаружения аномалий.

В данной работе рассматриваются различные алгоритмы обучения с учителем, так как были доступны размеченные данные.

## Постановка задачи

Целью выпускной квалификационной работы является решение задачи определения оптимальных цен на спортивное мероприятие до старта продаж.

Данная задача имеет ряд особенностей, однако возможно перенести полученные выводы и на другие, смежные области.

Оптимальной назовём такую цену $P_i$, что при определённом спросе $Q_i$ на товар $i$ максимальная возможная выручка $TR_i$ достигается при цене $P_i$. Предположим, что из всех факторов, которые влияют на $Q_i$, есть возможность менять только $P_i$. Таким образом, чтобы определить оптимальную цену, необходимо предсказать спрос на товар $i$ при различных ценах $P_i$ и выбрать ту, при которой $TR_i$ становится максимальной.

Общая схема подхода к решению данной задачи представлена на Рис. 1.

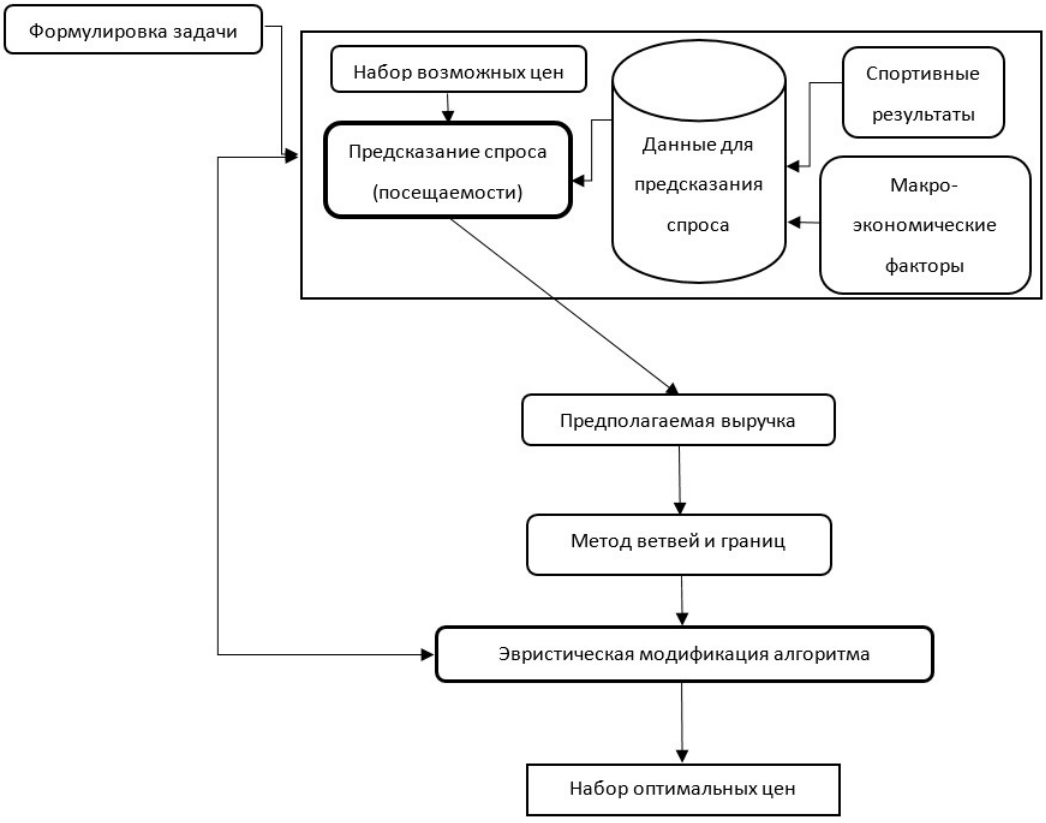



**Рис. 1:** Схема решения задачи

Из схемы очевидно, что самым трудоёмким и сложным элементом

является предсказание спроса (посещаемости). Именно эту проблему возможно решить с помощью машинного обучения.

Поставленную задачу можно решать как задачу классификации, так и как задачу регрессии. В первом случае необходимо научить алгоритм определять будет ли куплен конкретный товар (конкретное место на стадионе) или нет (задача бинарной классификации). С другой стороны также можно предсказывать какое количество товара определенной категории будет продано (какова будет заполняемость сектора), это задача регрессии.

Задача предсказания спроса выглядит следующим образом:

**Определение 1.** *Дано множество обучающих пар* $\left\{(x^i, d_i)\right\}_{i=1}^{k}$, *где* $x^i = \left(x_1^i, \ldots, x_n^i\right)$ *- обучающие признаки,* $d_i$ *- значение целевой переменной.* $d_i \in \{0, 1\}$ *в задаче классификации и* $d_i \in \mathbb{R}, d_i \geq 0$ *в задаче регрессии. Необходимо обучить такой алгоритм* $a$, *что* $L(d_i, y_i) \to min$, *где* $L$ *- функция потерь,* $y_i$ *- результаты работы алгоритма* $a$.

## Обзор литературы

Хотя задача оптимизации цены является одной из базовых в экономике, а машинное обучение в последнее время применяется практически в любой области науки, существует не так много научных работ, посвящённых этой проблеме. Однако, существующие показывают эффективность применения машинного обучения в похожих задачах. Например, в [4] успешно решается задача оптимизации цены в оффлайн-ритейле в полу-люкс сегменте.

Факторы, влияющие на спрос в спортивной индустрии описаны в статье [5]

Общий обзор современных методов машинного обучения представлен в книге "Pattern recognition and machine learning" [16]. В том числе там рассмотрены такие базовые алгоритмы как линейная регрессия [6] и машина опорных векторов [7].

Также для решения поставленной задачи необходимо было ознакомиться с более продвинутыми алгоритмами, такими как случайный лес [8] и градиентный бустинг [9]. В качестве современной реализации градиентного бустинга был выбран Catboost [10], одной из основных особенностей которого является способность более эффективно работать с категориальными переменными.

Алгоритм, используемый в библиотеке auto-sklearn подробно рассмотрен в [11]. Основной особенностью данного подхода является эффективная оптимизация гиперпараметров за счёт мета-обучения, а также автоматическое построение ансамблей из применённых методов.

Теория по используемым в работе свёрточным нейронным сетям подробно описана в [12] и [13].

# Глава 1. Работа с данными

## 1.1 Исходные данные

Для обучения был доступен корпус с историей транзакций по предыдущим 30 матчам. Данные содержали информацию о каждом проданном билете. Напрямую с такими данными работать весьма затруднительно, поэтому они были предобработаны с помощью библиотеки Pandas. Был получен формат, представленный в Таблице 1.

| **Сектор** | **Цена** | **Заполненность сектора** | **Клуб** | **Турнир** | **Дата** | **Время** |
|---|---|---|---|---|---|---|
| A101 | 600 | 0,89 | Челси | АПЛ | 01.01 | 16:30 |

**Таблица 1:** Обработанные данные

Однако, этих признаков может быть недостаточно для предсказания спроса. На спрос на спортивное мероприятие влияет множество факторов. Основываясь на [5] и после проведения нескольких опытов были отобраны внешние факторы, представленные в Таблице 2.

| **Признак** | **Пример** |
|---|---|
| Рейтинг команды соперника | $x \in [0, 1]$ |
| Результат предыдущего матча | $x \in \{0, 1, 2\}$ |
| Положение в турнирной таблице | $x \in [0, n]$, $n$ – количество команд в турнире |
| Выходной/не выходной день | $x \in \{0, 1\}$ |
| Нововведения в команде | $x \in \{0, 1\}$ |
| Погода за несколько дней до матча | $x \in \{0, 1, 2\}$ (солнечно, пасмурно, осадки) |

**Таблица 2:** Дополнительные факторы

## 1.2 Обработка данных

Поскольку практически все алгоритмы требуют, чтобы на входе в данных отсутствовали нечисловые значения, их необходимо предобработать. Самый простой способ имеет название Label Encoding. Он заключается в кодировании данных напрямую (т. е. присвоении следующего порядкового номера каждому новому значению в колонке в порядке встречи в данных). Однако, при таком подходе возникает сразу несколько серьёзных проблем. Во-первых, новое значение для той категории, которая впервые в данных встретилась позднее, окажется больше, чем то, которое встретилось раньше, т. к. кодирование напрямую зависит от того, в каком порядке данные будут даны кодировщику. Из-за этого алгоритм может переобучиться и найти зависимости в данных, которых на самом деле нет. Во-вторых, при такой кодировке будет отсутствовать какая-либо информация о зависимости признака с целевым значением. Для решения этих проблем были использованы более продвинутые кодировщики. О них подробнее будет рассказано ниже.

## 1.3 One-hot encoding

One-hot encoding (унитарное кодирование) - способ кодировки данных, при котором категориальная переменная бинаризируется. После кодировки этим методом на выходе получается матрица $M \times N$, где $M$ - количество примеров, $N$ - количество различных значений переменной. Каждый новый столбец (признак) $1 \dots N$ соответствует одному из значений категориальной переменной. Столбец $x$ принимает значения $x \in \{0, 1\}$. Значение 1 принимается в той строке (примере), где изначально было соответствующее этому столбцу значение категориальной переменной и 0 в другом случае. Соответственно, в каждой строке полученной матрицы может присутствовать только одно значение $x_i = 1$.

Плюсы данного подхода по сравнению с Label Encoding:

- Решается проблема с наличием зависимости от порядка, в котором данные подаются кодировщику.

Минусы данного подхода:

- Все ещё не решается проблема с отсутствием какой-либо зависимости кодировки от целевой переменной;
- В случае, если переменная принимает очень много значений, матрица может получиться слишком большой и будет занимать много места в памяти. Эта проблема частично решается хранением в виде разреженной матрицы, однако не все реализации алгоритмов поддерживают работу с таким типом данных;
- Проблематично добавлять новое значение категориальной переменной. Зачастую изначально добавляется лишний столбец и, в случае если появляется дополнительное значение (например, новая категория в тестовых данных), то 1 появляется именно в этом столбце. Но если добавляется сразу много новых значений, которых не было в обучающей выборке, то они будут просто объединяться в одну категорию, что в некоторых случаях может быть совершенно неприемлемо.

## 1.4 Target encoding

Target encoding - способ кодирования данных, в котором значение категориальной переменной заменяется средним значением целевой переменной для каждой категории. Предположим, что есть категориальная переменная $x$ и целевая $y$ (может быть как бинарной, так и иметь неограниченное число принимаемых значений). Для каждого принимаемого переменной $x$ значения высчитывается среднее $y$ для всех примеров, содержащих рассматриваемое значение $x$. После чего $x$ в примерах, содержащих рассмотренное значение, заменяются полученным средним. Рассмотрим пример из [14]. В Таблице 3 представлены категориальные признаки $x_0$ и $x_1$ и целевая переменная $y$.

| $x_0$ | $x_1$ | $y$ |
|---|---|---|
| a | c | 1 |
| a | c | 1 |
| a | c | 1 |
| a | c | 1 |
| a | c | 0 |
| b | c | 1 |
| b | c | 0 |
| b | c | 0 |
| b | c | 0 |
| b | d | 0 |

**Таблица 3:** Пример. Исходные данные

Заменим значения переменных $x_0$ и $x_1$ средними значениями $y$ по категориям. Получим Таблицу 4.

| $x_0$ | $x_1$ | $y$ |
|---|---|---|
| 0.8 | 0.444 | 1 |
| 0.8 | 0.444 | 1 |
| 0.8 | 0.444 | 1 |
| 0.8 | 0.444 | 1 |
| 0.8 | 0.444 | 0 |
| 0.2 | 0.444 | 1 |
| 0.2 | 0.444 | 0 |
| 0.2 | 0.444 | 0 |
| 0.2 | 0.444 | 0 |
| 0.2 | 0 | 0 |

**Таблица 4:** Пример. Закодированные данные

Плюсы данного подхода по сравнению с Label Encoding:

- Решается проблема с наличием зависимости от порядка, в котором данные подаются кодировщику;
- Также, появляется корреляция между значением категориальной переменной и целевой. Это может сильно помочь некоторым алгоритмам найти зависимости в данных;

- Данные довольно просто закодировать данным способом. Не возникает никаких новых столбцов, а также размер занимаемого места в памяти компьютера не увеличивается.

Основным минусом данного подхода является опасность переобучения в случае, если классы несбалансированны или количество примеров некоторых классов слишком мало. Так, в примере выше можно увидеть, что значение $d$ у переменной $x_1$ заменяется на 0, так как в данных присутствовал лишь один пример с таким значением. Однако, нельзя утверждать, что 0 является действительно средним целевой переменной для данного значения, поэтому есть большой риск переобучиться при таком кодировании.

Существует несколько способов бороться с данной проблемой. Самый простой - использовать кросс-валидацию и считать среднее на каждом фолде. Но все же это также не всегда может помочь. Даже во всей доступной выборке может присутствовать слишком мало значений для какой-то одной категории. В таком случае используется ещё один способ - аддитивное (Лапласово) сглаживание. Идея этого метода заключается в том, чтобы использовать в вычислении среднего также среднее по всей выборке. Это можно выразить в следующей формуле:

$$\mu = \frac{n * \bar{x} + m * \omega}{n + m},$$

где:

- $\mu$ - среднее, которое должно быть вычислено,
- $n$ - количество значений, которые присутствуют в выборке,
- $\bar{x}$ - среднее, вычисленное обычным образом, оценочное среднее,
- $m$ - гиперпараметр, «вес», который даётся общему среднему в данной формуле,
- $\omega$ - общее среднее.

Здесь с помощью параметра $m$ есть возможность контролировать, насколько сильно следует учитывать среднее по всей выборке. Так, применив данную формулу с параметром $m$, равным 10, получается результат, представленный в Таблице 5.

| $x_0$ | $x_1$ | $y$ |
|---|---|---|
| 0.6 | 0.526316 | 1 |
| 0.6 | 0.526316 | 1 |
| 0.6 | 0.526316 | 1 |
| 0.6 | 0.526316 | 1 |
| 0.6 | 0.526316 | 0 |
| 0.4 | 0.526316 | 1 |
| 0.4 | 0.526316 | 0 |
| 0.4 | 0.526316 | 0 |
| 0.4 | 0.526316 | 0 |
| 0.4 | 0.454545 | 0 |

**Таблица 5:** Пример. Закодированные данные модифицированным способом

При таком подходе закодированные значения лежат намного ближе к среднему значению по выборке (оно в данном случае равно 0.5), а также отсутствуют выбросы.

## 1.5 Дополнительная обработка данных

Помимо описанных выше кодировщиков к некоторым признакам были также применены другие. Порядковый кодировщик был применен для таких признаков как время начала мероприятия, дата (данные кодируются согласно порядку от меньшего к большему). Также для бинарных признаков была применена бинарная кодировка.

С помощью инструмента Pipeline из библиотеки scikit-learn было также проведено скалирование и трансформация данных бы с выбросами.

Код преобразований представлен ниже.

```
Pipeline(steps = [
    ('feature_processing',
     ColumnTransformer(transformers = [
            #numeric
            ('numeric', Pipeline([
                ('impute', SimpleImputer(
                      missing_values=np.nan,
                      strategy='mean')),
                ('scale', RobustScaler()),
                ('transform', QuantileTransformer(
                    output_distribution='normal')),
                ('engineer', PolynomialFeatures())]),
            numerical_features),

            #categorical
            ('categorical', Pipeline([
                ('impute', SimpleImputer(
                      missing_values=np.nan,
                      strategy='constant',
                      fill_value=0)),
          ('onehot', OneHotEncoder(
          handle_unknown='ignore',
          sparse=False))]),
              categorical_features),
    ])),]
)
```

# Глава 2. Алгоритмы машинного обучения

Как говорилось ранее, задачу предсказания спроса на спортивное мероприятие можно представить как в виде задачи регрессии, так и в виде задачи бинарной классификации. Поэтому ниже будут описаны использованные алгоритмы для обеих задач. Для алгоритмов регрессии данные группируются по секторам стадиона, а в целевой переменной хранится посещаемость соответствующего сектора. Задача алгоритма - предсказать посещаемость сектора при наличии всех остальных признаков, в том числе и цены.

Для алгоритмов классификации задача ставится в следующем формате. Обучающие примеры составляются из всех выше описанных признаков, однако целевая переменная $y$ принимает значения $\{0, 1\}$, где $y = 1$ означает, что место было продано, $y = 0$, что нет. Очевидно, что в точности предсказать, какие именно места продадутся, невозможно, однако, различные места имеют различную привлекательность для посетителей. Поэтому, например, места на задних рядах с меньшей вероятностью будут проданы на игру с меньшим спросом. Для сравнения результатов с алгоритмами регрессии полученные предсказания группируются по секторам. Перейдём к рассмотрению использованных алгоритмов.

## 2.1 Линейная регрессия

В качестве базового алгоритма была выбрана линейная регрессия [6]. Линейная регрессия — метод восстановления зависимости между двумя переменными.

Для заданного множества пар $(x_i, y_i)$, $i = 1, \ldots, m$ значений свободной и зависимой переменной требуется построить зависимость. Представлена линейная модель $y_i = f(\omega, x_i) + \epsilon_i$ с аддитивной случайной величиной $\epsilon$. Переменные $x, y$ принимают значения на числовой прямой $R$. Предполагается, что случайная величина распределена нормально с нулевым матожиданием и фиксированной дисперсией $\sigma^2$, которая не зависит от переменных $x, y$. При таких предположениях параметры $w$ регрессионной модели вычисляются с помощью метода наименьших квадратов.

Данный алгоритм не способен находить сложные зависимости в данных, поэтому используется лишь в качестве базового.

## 2.2 Алгоритм случайного леса

Дерево принятия решений [8] - средство поддержки принятия решений. Суть алгоритма заключается в построении структуры типа дерево на основе входных данных. Структура дерева включает в себя листья и ветви (рёбра). На рёбрах записаны атрибуты, на которых основывается целевая функция. В листьях хранятся значения целевой функции. В остальных узлах - атрибуты, по которым различаются случаи. Для использования построенного дерева необходимо согласно правилам спуститься до какого-либо листа и получить соответствующее значение.

Плюсы решающего дерева:

- Интерпретируемость построенного дерева,
- Быстрое вычисление
- Может работать с категориальными переменными

Минусы:

- Склонность к переобучению
- Длительность обучения очень высока.

Для борьбы с переобучением используются ансамбли решающих деревьев - случайный лес. Основная идея заключается в переобучении нескольких решающих деревьев, а затем в объединении полученных результатов.

Общий алгоритм классификации с помощью случайного леса выглядит следующим образом:

Обозначим за $N$ - размер обучающей выборки, за $M$ - количество обучающих примеров. Выберем $m$ случайных признаков (в задаче классификации обычно выбирается $m \approx \sqrt{M}$, в задаче регрессии $m \approx [M/3]$). Наиболее распространённый способ построения деревьев это бэггинг:

1. Выбирается случайная подвыборка с повторениями из всех обучающих примеров.

2. Строится решающее дерево, которое классифицирует объекты этой подвыборки. Оно строится на основе $m$ признаков. Для выбора следующего признака может использоваться стандартный критерий Джини или какой-либо другой критерий.

3. Дерево строится до самого конца, т. е. пока не закончатся все примеры подвыборки. Также, оно не подвергается отсечению, т. е. исключается любая борьба с переобучением для отдельно взятого дерева.

Чтобы в итоге классифицировать объект на основе построенного дерева проводится процедура голосования. Каждое дерево относит пример к одному из классов, выбирается тот класс, за который проголосовало большинство деревьев.

## 2.3 Алгоритм градиентного бустинга

Алгоритм случайного леса имеет 2 основные проблемы:

- Вычислительная сложность в случае большой выборки или большого числа признаков. При этом нельзя эти деревья упрощать, в таком случае эффективность алгоритма падает;

- Процесс построения деревьев ненаправленный: все деревья строятся случайно, и для действительно хороших результатов в некоторых случаях необходимо построить очень большое число деревьев.

Именно эти проблемы призван решить алгоритм градиентного бустинга [9]. Бустинг - подход к построению композиций, основными особенностями которого является тот факт, что базовые алгоритмы строятся последовательно, один за другим. Неформально можно описать, что задача каждого последующего алгоритма - исправить ошибки предыдущего. Таким образом, построение композиций направлено. Здесь в отличие от алгоритма случайных деревьев наоборот используются неглубокие деревья.

В градиентном бустинге строящаяся композиция выглядит следующим образом:

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

Функция потерь может быть выбрана следующим образом:

- Среднеквадратичная ошибка (для задачи регрессии):

$$L(y, z) = (y - z)^2$$

- Логистическая функция потерь (для задачи классификации):

$$L(y, z) = log(1 + exp(-yz))$$

Базовые алгоритмы строятся последовательно. Нужно уменьшить ошибку на обучающей выборке. Необходимо найти такой вектор сдвигов $s = (s_1, \ldots, s_l)$, $b_N(x_i) = s_i$, чтобы ошибка на примерах обучающей выборки была минимальной:

$$F(s) = \sum_{i=1}^{l} L(y_i, a_{N-1}(x_i) + s_i) \to min$$

Вектор $s$ должен быть таким, при котором бы минимизировалась функция F(s). Направление наискорейшего убывания функции задаётся направлением антиградиента, отсюда следует следующее выражение:

$$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{N-1}(x_1)), \\ \ldots \\ -L'_z(y_l, a_{N-1}(x_l)) \end{pmatrix}$$

Таким образом, на каждом шаге необходимо минимизировать не исходную функцию потерь, а «исправлять» ошибки алгоритма на предыдущем шаге.

Тогда общий алгоритм градиентного бустинга выглядит следующим образом:

**Алгоритм 1** Градиентный бустинг

Инициализация $a_0(x) = b_0(x)$

**for** t = 1 ... N **do**

$$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{N-1}(x_1)), \\ \dots \\ -L'_z(y_l, a_{N-1}(x_l)) \end{pmatrix}$$

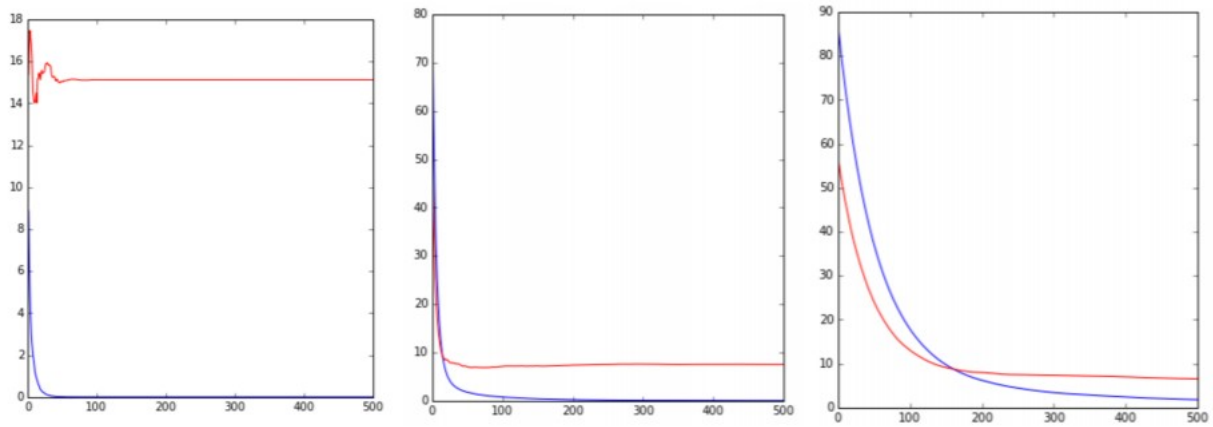
$b_n(x) = argmin_b \frac{1}{l} \sum_{i=1}^{l} (b(x_i) - s_i)^2$

Алгоритм $b_n(x)$ добавляется в композицию с «весом» $\eta$, т. е.

$a_N(x) = a_{N-1}(x) + \eta b_N(x)$

**end for**

Важно, что на последнем шаге новый алгоритм добавляется в композицию с определённой длиной шага $\eta$. Если этого не делать, то есть большой риск, что градиентный бустинг уйдёт в случайное блуждание и произойдёт переобучение. Ограничивая влияние каждого нового элемента в композиции, можно избавиться от нежелательного эффекта, заключающегося в том, что алгоритм не может верно аппроксимировать функцию. Действие этого коэффициента продемонстрировано на Рис. 2:



(a) Случай $\eta = 1$. (b) Случай $\eta = 0.1$. (c) Случай $\eta = 0.01$.

**Рис. 2:** Качество градиентного бустинга на обучающей (синий) и контрольной выборке(красный)

Существует много различных реализаций градиентного бустинга со своими особенностями. Одной из них является библиотека Catboost [10].

Отличием этой библиотеки можно назвать модифицированный подход борьбы с переобучением: использование упорядоченного бустинга, использование "забывчивых"(oblivious) деревьев в алгоритме. Основное отличие этих деревьев от обычных заключается в том, что один и тот же критерий используется во всех узлах на одном уровне. Такие деревья сбалансированы, менее склонны к переобучению, работают быстрее во время тестирования построенного алгоритма. Этот подход особенно хорошо позволяет бороться с переобучением на небольших корпусах данных.

Также важной особенностью алгоритма Catboost является улучшенная работа с категориальными признаками. Здесь используется подход, основанный на выше упомянутом Target Encoding. Однако среднее значение целевой переменной высчитывается для каждого объекта на основе исторических данных, т.е. лишь на тех данных, которые уже «видел» алгоритм. Данные постоянно случайно перемешиваются. Также в Catboost используется стандартный подход для создания новых признаков: комбинации существующих. Однако, они создаются жадным образом, что позволяет бороться с экспоненциальным ростом количества признаков.

Именно эта библиотека была использована для экспериментов, т. к. в задаче предсказания спроса присутствует довольно большое количество категориальных переменных, а также исторических данных для обучения доступно не очень много.

## 2.4 AutoML подход

В последнее время стали популярны различные алгоритмы, которые можно объединить под термином AutoML. Основная их идея заключается в подборе решающих функций, а также их гиперпараметров автоматически, без участия либо с минимальным участием исследователя. Этот подход может работать на простых выборках, с простейшими зависимостями, однако на более сложных данных результаты как правило не удовлетворительны. Но этот метод может быть полезен в качестве одного из элементов финальной композиции (ансамбля). К тому же, качественная предобработка данных совместно с AutoML может показать высокие результаты.

В качестве реализации была выбрана библиотека [11]. Её особенностью является использование Байесовской оптимизации для подбора параметров. Опишем задачу AutoML более формально:

Пусть $\mathcal{A} = \left\{A^{(1)}, \ldots, A^{(R)}\right\}$ - набор алгоритмов, а $\Lambda^{(j)}$ - пространство гиперпараметров алгоритма $\mathcal{A}^{(j)}$. Тренировочная выборка $D_{train}$ разбита на $K$ кросс-валидационных фолдов $\left\{D_{valid}^{(1)}, \ldots, D_{valid}^{(K)}\right\}$ и $\left\{D_{train}^{(1)}, \ldots, D_{train}^{(K)}\right\}$, таких что $D_{train}^{(i)} = D_{train} \backslash D_{valid}^{(i)}$ для $i = 1, \ldots, K$. $\mathcal{L}\left(A_{\lambda}^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)}\right)$ - значение функции потерь, которое алгоритм $A^j$ достигает на $D_{valid}^{(i)}$ будучи натренированным на $D_{train}^{(i)}$ с гиперпараметрами $\lambda$. Задача AutoML - найти набор алгоритмов и значения их гиперпараметров такие, что минимизируется функция потерь $\mathcal{L}$:

$$A^*, \lambda_* \in argmin_{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}} \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}(A_{\lambda}^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$$

Схема решения этой проблемы представлена на Рис. 3.

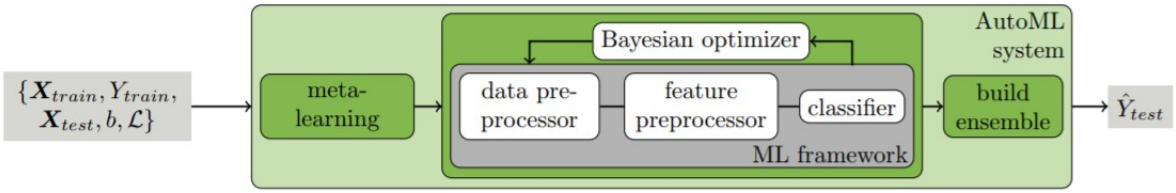



**Рис. 3:** AutoML подход (auto-sklearn)

Особенностями данного подхода можно назвать использование Байесовской оптимизации (Глава 3) для подбора оптимальных параметров, а также мета-обучение. С помощью мета-обучения решается проблема выбора начального приближения для поиска оптимальных гиперпараметров. Суть подхода заключается в следующем:

- Для набора известных датасетов (например, из репозитория OpenML [15]) подбираются оптимальные алгоритмы и их гиперпараметры. Также высчитываются мета-характеристики датасета.
- Подобранные параметры сохраняются для дальнейшего использования

- Затем для нового корпуса $\mathcal{D}$ высчитываются его мета-характеристики. Все обработанные ранее датасеты ранжируются согласно расстоянию $L_1$ до датасета $\mathcal{D}$ в пространстве мета-характеристик и выбирается $k = 25$ ближайших датасетов. Их оптимальные параметры используются в качестве начального приближения для старта Байесовской оптимизации.

Под мета-характеристиками датасета могут пониматься такие величины, как количество признаков, коэффициент ассиметрии, энтропия целевой переменной и т. д.

## 2.5 Нейронные сети

Одним из самых популярных разделов машинного обучения по праву являются нейронные сети.

Построение свёрточных нейронных сетей является одним из современных направлений в развитии нейронных сетей. Свёрточные нейронные сети состоят из набора свёрточных слоёв и слоёв пулинга. В качестве классификатора используется несколько полносвязных слоёв после свёрточных. Свёрточные нейронные сети могут использоваться как самостоятельно, так и в качестве одного из элементов большей нейронной сети. Теория свёрточных нейронных сетей подробно изложена в [12] и [13].

Одной из популярных библиотек для построения нейронных сетей является PyTorch [19]. Одной из причин, почему эта библиотека становится всё популярнее является наличие динамического вычислительного графа. Также, API данной библиотеки многими исследователями признаётся наиболее удобным для работы.

Выбранная архитектура нейронной сети представлена на Рис. 4.

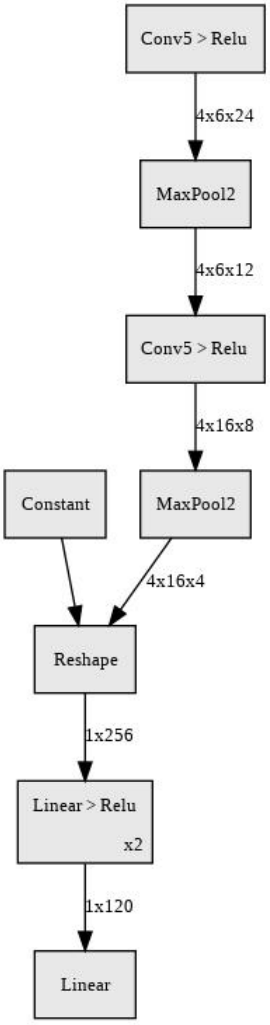



**Рис. 4:** Архитектура свёрточной нейронной сети

Архитектура взята не самая сложная, т. к. обучающих данных присутствовало не очень много, и в случае добавления дополнительных слоёв или выбора совершенно другой архитектуры (например, рекуррентных нейронных сетей), сети не хватало обучающих примеров, чтобы показать удовлетворительную метрику даже на тренировочных данных.

# Глава 3. Байесовская оптимизация

Важным аспектом качественной работы алгоритма машинного обучения являются правильно подобранные гиперпараметры. Есть несколько возможных подходов работы с параметрами различных алгоритмов:

- Ручной подбор параметров. Конечно, всегда можно подобрать оптимальные параметры вручную, однако этот процесс весьма трудоёмкий, и его результативность существенно зависит от квалификации исследователя;
- Подбор параметров по сетке. Вручную задаются параметры сетки. В случае большого числа гиперпараметров может работать очень долго;
- Байесовская оптимизация;
- Другие методы оптимизации.

Байесовская оптимизация [17] - это последовательная стратегия проектирования глобальной оптимизации функции типа «чёрный ящик», не требующая взятия производных.

**Определение 2.** ***Гауссовский случайный процесс.***
*Последовательность* $\{y_t\}_{t\in T}$ *называется гауссовским случайным процессом, если для любого конечного множества* $\{t_1, t_2, \ldots, t_n\}$ *случайные величины* $\{y_{t_1}, y_{t_2}, \ldots, y_{t_n}\}$ *имеют совместное многомерное нормальное распределение.*

Задача состоит не в том, чтобы найти саму функцию (что зачастую практически невозможно), а лишь найти аргумент доставляющий наибольшее значение. Необходимо ввести функцию выгоды (acquisition function), чтобы определить, какое значение необходимо выбрать далее.

Вероятность улучшения (Probability of Improvement):

$$PI(x) = \Phi\left(\frac{\mu(x) - \mu^+ - \epsilon}{\sigma(x)}\right)$$

где $\mu^+$ - лучшее рассмотренное значение, $\epsilon \sim N(0, \nu)$

Тогда функция $a$ (функция выгоды) будет иметь следующий вид:

$$a_{PI}(x; \mathcal{D}_n) = \mathbb{P}\left[v > \tau\right] = \Phi\left(\frac{\mu_n(x) - \tau}{\sigma_n(x)}\right)$$

Также можно использовать функцию ожидаемого улучшения (Expected Improvement)

Тогда алгоритм Байесовской оптимизации выглядит следующим образом:

На вход подается функция $f(\cdot)$, функция выгоды $a(\cdot)$, максимальное количество вычислений функции $T$, начальное множество значений $D = \{x_i, y_i\}_{i=1}^{k}$

**Алгоритм 2** Байесовская оптимизация

**for** t = 1 ... T **do**

$x_t = argmax_x a(x; D)$

$y_t = f(x_t)$

$D \leftarrow D \cup \{x_t, y_t\}$

**end for**

На Рис. 5, Рис. 6, Рис. 7 продемонстрированы результаты работы при $t = 2, t = 3$ и $t = 4$ соответственно.

Реализация данного подхода была взята из библиотеки bayesian-optimization [18].

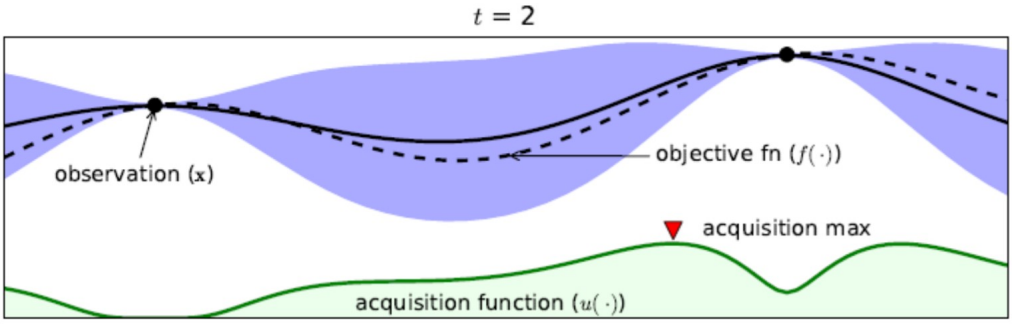



**Рис. 5**

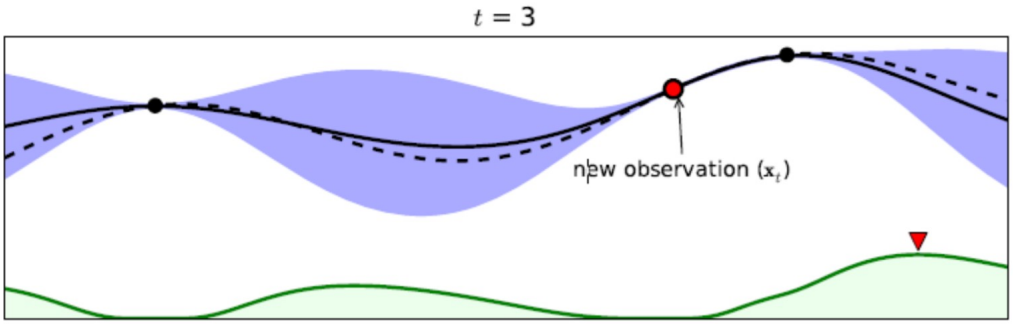

$t = 3$
new observation ($\mathbf{x}_t$)

**Рис. 6**

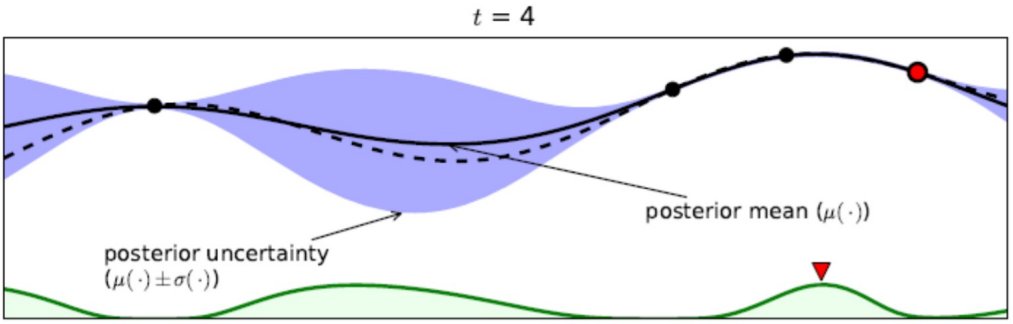

$t = 4$
posterior mean ($\mu(\cdot)$)
posterior uncertainty
($\mu(\cdot) \pm \sigma(\cdot)$)

**Рис. 7**

# Глава 4. Программные средства

Для проведения экспериментов использовался язык программирования Python и следующие библиотеки:

- scikit-learn версии 0.20.1 для тестирования линейной регрессии и алгоритма случайного леса, а также оценивания результатов работы алгоритмов
- pandas-profiling для удобного анализа входных данных и нахождения сильно скореллированных между собой признаков
- catboost для тестирования градиентного бустинга
- PyTorch для работы с нейронными сетями
- bayesian-optimization для Байесовской оптимизации параметров
- hiddenlayer для визуализации модели нейронной сети

# Глава 5. Анализ экспериментов

## 5.1 Сравнение алгоритмов

Для сравнения алгоритмов необходимо выбрать соответствующие метрики. Для задачи регрессии были выбраны 2 метрики:

- RMSE. Стандартной метрикой для сравнения различных алгоритмов между собой является **Root Mean Sqaure Error**. Её формула:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

- Также для более понятной человеческой интерпретации используется метрика $R^2$ (коэффициент детерминации). Это доля дисперсии зави-

симой переменной, объясняемая рассматриваемой моделью.

$$R^2 = 1 - \frac{V(y|x)}{V(y)} = 1 - \frac{\sigma^2}{\sigma_y^2}$$

$V(y|x) = \sigma^2$ - условная (по факторам x) дисперсия зависимой переменной (дисперсия случайной ошибки модели)

Для алгоритмов классификации использовались следующие метрики:

- Accuracy - доля правильных ответов. Однако данной метрики может быть недостаточно, так как нет гарантии сбалансированности классов.
- ROC-AUC - величина площади под кривой ROC (receiver operating characteristic)

Размер тренировочной выборки в задаче регресии составляет 941 пример. Тестовой выборки - 314 примеров.

Вначале было проведено сравнение базовых алгоритмов регрессии на тренировочной выборке. Результаты представлены в Таблице 6.

| **Алгоритм** | $RMSE$ | $R^2$ |
|---|---|---|
| Линейная регрессия | 91,915 | 0,571 |
| Случайный лес | 39,576 | 0,92 |

**Таблица 6:** Сравнение алгоритмов регрессии на тренировочной выборке

Затем на тестовой выборке. Результаты представлены в Таблице 7.

| **Алгоритм** | $RMSE$ | $R^2$ |
|---|---|---|
| Линейная регрессия | 23566218812,8 | $-2{,}5 \cdot 10^{16}$ |
| Случайный лес | 96,9 | 0,576 |

**Таблица 7:** Сравнение алгоритмов регрессии на тестовой выборке

Из полученных результатов видно, что линейная регрессия переобучилась, поэтому можно отвергнуть гипотезу о линейной зависимости. Алгоритм случайного леса также не показал приемлемых результатов.

Также было проведено сравнение результатов более продвинутых алгоритмов, представленное в Таблице 8 сначала на тренировочной выборке.

| **Алгоритм** | $RMSE$ | $R^2$ |
|---|---|---|
| Градиентный бустинг (CatBoost) | 26,8 | 0,76 |
| AutoML | 22,1 | 0,83 |
| Свёрточная нейронная сеть | 20,1 | 0,94 |

**Таблица 8:** Сравнение алгоритмов регрессии на тренировочной выборке

И на тестовой выборке(сравнение представлено в Таблице 9).

| **Алгоритм** | $RMSE$ | $R^2$ |
|---|---|---|
| Градиентный бустинг (CatBoost) | 74,3 | 0,71 |
| AutoML | 40,3 | 0,82 |
| Свёрточная нейронная сеть | 46,2 | 0,81 |

**Таблица 9:** Сравнение алгоритмов регрессии на тестовой выборке

Размер тренировочной выборки - 637296 примеров, тестовой - 212432. При разбиении применялась стратификация. Также, необходимо понимать, что каждый пример - отдельно взятое место, поэтому в данных присутствует значительное число крайне похожих друг над друга примеров.

Результаты работы алгоритмов классификации на тренировочной выборке представлены в Таблице 10.

| **Алгоритм** | *Accuracy* | *ROC – AUC* |
|---|---|---|
| Случайный лес | 0,88 | 0,87 |
| Градиентный бустинг (CatBoost) | 0,9 | 0,89 |
| AutoML | 0,91 | 0,9 |
| Свёрточная нейронная сеть | 0,92 | 0,89 |

**Таблица 10:** Сравнение алгоритмов классификации на тренировочной выборке

Результаты работы на тестовой выборке представлены в Таблице 11.

| **Алгоритм** | *Accuracy* | *ROC – AUC* |
|---|---|---|
| Случайный лес | 0,76 | 0,75 |
| Градиентный бустинг (CatBoost) | 0,88 | 0,87 |
| AutoML | 0,9 | 0,89 |
| Свёрточная нейронная сеть | 0,89 | 0,89 |

**Таблица 11:** Сравнение алгоритмов классификации на тестовой выборке

Из приведённых выше результатов можно сделать вывод, что все алгоритмы, кроме линейной регрессии, так или иначе применимы к данной задаче. Причём с точки зрения борьбы с переобучением лучше всего себя показал AutoML подход, так как ошибка на $D_{test}$ и на $D_{train}$ значительно не отличается.

Для сравнения алгоритмов классификации и регрессии полученные результаты алгоритмов классификации группировались по секторам после чего получался формат, который использовался в задачах регрессии. Результаты можно увидеть в Таблице 12.

| **Алгоритм** | $RMSE$ | $R^2$ |
|---|---|---|
| Градиентный бустинг (CatBoost) | 76,4 | 0,70 |
| AutoML | 38,2 | 0,84 |
| Свёрточная нейронная сеть | 39,1 | 0,83 |

**Таблица 12:** Представление результатов алгоритмов классификации в формате данных, используемых для решения задачи регрессии

Таким образом, алгоритмы классификации (кроме градиентного бустинга) для поставленной задачи классификации отработали лучше. Так произошло, скорее всего, из-за того, что в формате данных для классификации, где 1 примеру соответствует одно место, получается больше обучающих примеров.

## 5.2 Ансамблирование

Так как многие алгоритмы допускают ошибки в разных категориях, то имеет смысл провести ансамблирование лучших из них. Ансамблирование (ансамбль методов) - подход в машинном обучении, при котором используется несколько обучающих алгоритмов для получения лучшей предсказательной способности.

Существует несколько различных техник ансамблирования:

- Бэггинг. Такой же принцип применяется в алгоритме случайного леса;

- Бустинг. Этот принцип исплоьзуется в алгоритме градиентного бустинга. Строится ансамбль последовательными приращениями модели;

- Байесовский оптимальный классификатор. Каждой гипотезе даётся голос, который пропорционален вероятности того, что тренировочные данные будут выбраны из системы, если гипотеза была бы верна. Можно выразить следующим равенством:

$$y = \underset{c_j \in C}{\operatorname{argmax}} \sum_{h_i \in H} P(c_j|h_i)P(T|h_i)P(h_i)$$

$y$ - предсказанный класс, $C$ - множество всех возможных классов, $H$ - класс гипотез, $T$ - тренировочные данные.

Так как на практике разница в приросте точности между подходами получается не сильно большой, а задачи увеличить метрику на доли процента не стоит, то была выбрана самая простая в реализации техника ансамблирования - бэггинг. Она была применена для задачи классификации. На тестовых данных были получены результаты немного лучше достигнутых ранее. Эти результаты представлены в Таблице 13.

| **Алгоритм** | *Accuracy* | $ROC - AUC$ |
|---|---|---|
| Ансамбль (на основе бэггинга) | 0.92 | 0,9 |

**Таблица 13:** Результаты ансамблирования

# Глава 6. Оптимизация цены на основе предсказанного спроса

После предсказания спроса необходимо вычислить оптимальную цену. В случае, если возможных значений цены не так много (выборка дискретна), то можно просто провести полный перебор всех цен и выбрать ту, которая устраивает исследователя.

Поскольку цена на билеты на спортивные мероприятия имеет сильно ограниченный набор значений (разница между возможными ценами около 100 рублей), то в данном случае есть возможность провести поиск по сетке с целью найти оптимальную цену. Соответственно, для любой желаемой посещаемости можно выбрать оптимальную цену, чтобы получить максимальную выручку.

В таком случае график зависимости посещаемости от выручки будет выглядеть следующим образом:

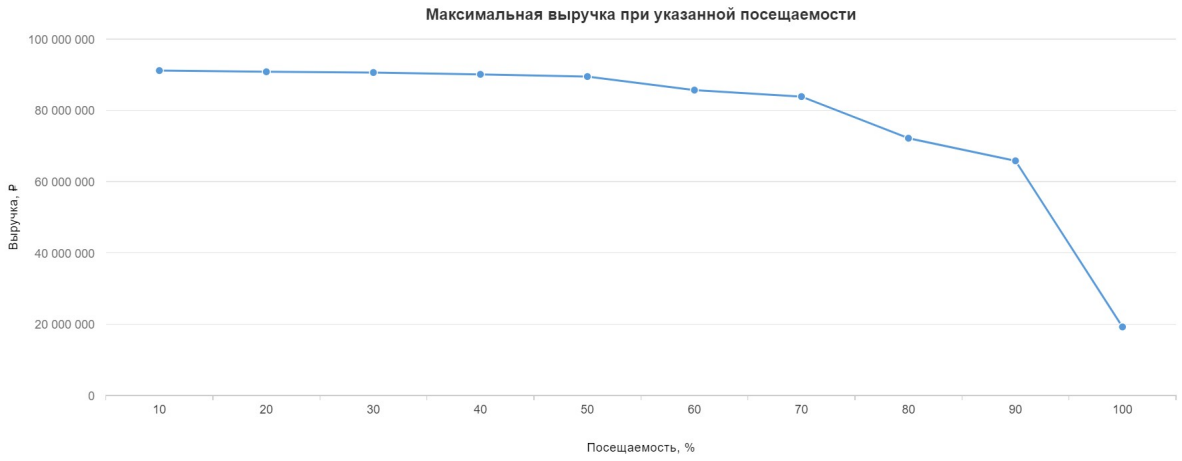



**Рис. 8:** Зависимость посещаемости от выручки

На первый взгляд полученный результат может смутить, однако он соответствует результатам анализа, проведённого в [5]: чем выше цены, тем ниже посещаемость и выше выручка. Однако, спортивный клуб заинтересован не только в повышении выручки от продажи билетов, но и в том, чтобы посещаемость преодолевала определённый порог, т. к. значительная часть дохода от мероприятия приходится также на продажу атрибутики и еды.

## Выводы

Зачастую, многие исследователи пренебрегают использованием различных алгоритмов машинного обучения и делают выбор сразу в пользу нейронных сетей. Однако, на примере данной задачи стало очевидно, что хоть нейронные сети и показали сравнимый или, возможно, даже немного более качественный результат, в случае необходимости можно было обойтись без их применения.

Проанализировав результаты, можно прийти к выводу, что с помощью алгоритмов машинного обучения возможно довольно точно предсказать спрос на спортивное мероприятие, а затем использовать полученные результаты для оптимизации цен. При этом подход, в котором эта задача формулируется как задача бинарной классификации оказался более подходящим по следующим причинам:

- Он показывает сравнимые результаты с алгоритмами регрессии, при этом обработка исходных данных требует меньших трудозатрат

- Результаты работы алгоритма позволяют предсказывать не только спрос, но и использовать их для анализа смежных проблем. Например, в примере с спортивным мероприятием, можно оценить, какие именно небольшие группы мест будут пользоваться наибольшим спросом. С помощью такой информации можно, например, выбирать не одну цену на всём секторе, а варьировать её в зависимости от конкретного места.

- Практически каждая реализация алгоритма машинного обучения позволяет предсказывать вероятность отнесения объекта к тому или иному классу (в нейронных сетях необходимо заменить последний слой на Softmax). Таким образом, результаты работы алгоритма можно использовать для ручного анализа, например, если известно, что на конкретный матч по какой-то причине, которую не мог учесть алгоритм, будет больший спрос. Необходимо просто «сдвинуть» порог, при котором считается, что место будет куплено.

Важной характеристикой достигнутых результатов является то, что гиперпараметры были подобраны не вручную, а с помощью Байесовской оптимизации. Таким образом, можно утверждать, что как минимум некоторые алгоритмы показали близкие к возможной границе результаты.

Также довольно важно было обработать исходные данные. Как и в практически любых реальных данных в них присутствовало довольно много выбросов и неточностей. Именно борьба с ними позволила достигнуть полученного качества предсказаний.

# Заключение

В результате выполнения данной выпускной квалификационной работы были достигнуты следующие результаты:

- Рассмотрены различные способы предобработки данных;
- Опробованы разные алгоритмы от самых простейших, таких как линейная регрессия, до нейронных сетей;
- Проведено сравнение результатов алгоритмов между собой;
- Построен ансамбль алгоритмов;
- Проведена оптимизация цен на основе предсказанного спроса.

# Список литературы


[1] Autoregressive integrated moving average // Wikipedia.org URL: https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average (дата обращения: 22.12.2018).

[2] Generalized autoregressive conditional heteroskedasticity // Wikipedia.org URL:

[3] Rabiner L.R. A tutorial on hidden Markov models and selected applications in speech recognition // Proceedings of the IEEE. 1989. P. 257-286.

[4] Qu T., Zhang J. H., Felix T. S. et al. Demand prediction and price optimization for semi-luxury supermarket segment // Computers and Industrial Engineering. 2017. No 113. P. 91–102.

[5] Silvestri F. Optimal ticket pricing in the sport industry. The case of the Italian Serie A // La Kivista di diritto ed economia dello sport. 2016. No 1. P. 12.

[6] Galton F. Regression Towards Mediocrity in Hereditary Stature // Journal of the Anthropological Institute. 1886. №15. P. 246-263.

[7] Hearst Marti A. Support Vector Machines // IEEE Intelligent Systems. 1998. №13. P. 18-28.

[8] Breiman L. Random Forests // Machine Learning. 2001. №45/1. P. 5-32.

[9] Friedman J. H. Stochastic Gradient Boosting // Computational Statistics & Data Analysis. 2002. №38/4. P. 367-378.

[10] Dorogush A. V., Ershov V., Gulin A. CatBoost: gradient boosting with categorical features

[11] Feurer M., Klein A. et al. Efficient and robust automated machine learning // Advances in Neural Information Processing Systems. Montreal. 2015. No 28.

[12] Schmidhuber J. Deep Learning in Neural Networks: An Overview P. 10-18, 2014 [Электронный ресурс] // arXiv.org. URL: https://arxiv.org/pdf/1404.7828.pdf (дата обращения: 26.02.2019).

[13] LeCun Y., Bengio Y., Hinton G. Deep learning. Nature,. 521(7553) изд. 2015 P. 436-444.

[14] Target encoding done the right way // URL: https://maxhalford.github.io/blog/target-encoding-done-the-right-way/ (дата обращения: 26.02.2019). support [Электронный ресурс] // arXiv.org. URL: https://arxiv.org/pdf/1810.11363.pdf (дата обращения: 26.02.2019).

[15] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. SIGKDD Explorations 15(2), P. 49-60, 2013.

[16] Christopher M. Bishop. Pattern Recognition and Machine Learning, 2006.

[17] Shallow Understanding on Bayesian Optimization // Towards Data Science URL: https://towardsdatascience.com/shallow-understanding-on-bayesian-optimization-324b6c1f7083 (дата обращения: 23.01.2019).

[18] Bayesian Optimization: Open source constrained global optimization tool for Python // URL: https://github.com/fmfn/BayesianOptimization (дата обращения: 21.01.2019). https://en.wikipedia.org/wiki/Autoregressive_conditional _heteroskedasticity#GARCH (дата обращения: 20.12.2018).

[19] Paszke A., Gross S., Chintala S. et al. Automatic differentiation in PyTorch // NIPS-W. 2017.