МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт математики, механики и компьютерных наук
имени И. И. Воровича

Направление подготовки 02.03.02 — "Фундаментальная
информатика и информационные технологии"

Выпускная квалификационная работа
на степень бакалавра

Рахуба А.М., 4 курс, 9 группа

РЕШЕНИЕ ЗАДАЧИ АВТОМАТИЧЕСКОГО ОПРЕДЕЛЕНИЯ ЦЕНЫ НА ПРОДУКТЫ ПО ИХ ТЕКСТОВОМУ ОПИСАНИЮ

Научный руководитель:
к.ф.-м.н., доцент А. В. Абрамян

Допущено к защите:
Заместитель директора ИММКН
по направлению ФИИТ

_____ В.С. Пилиди

Ростов-на-Дону
2020

Содержание

Введение

В сегодняшнем мире оценить, сколько объективно стоит тот или иной товар, может являться довольно сложной задачей. Различные небольшие детали могут порождать большие различия в цене. Например, цены на одежду могут сильно колебаться в зависимости от сезонных трендов и влияний брендов, цены на электронику зависят от множества различных технических характеристик, а цены на покупки в интернете могут меняться в зависимости от времени и типа доставки и т.д. К примеру, один из этих свитеров стоит 335$, а другой 10$ (см. рис. 1).

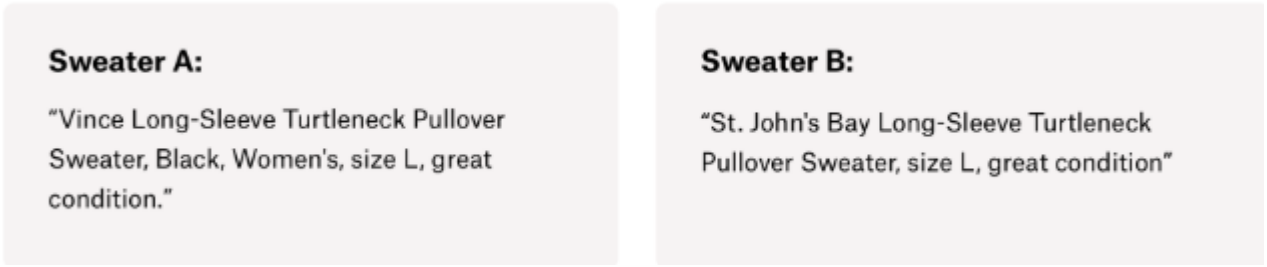

Рисунок 1 — Два свитера с большой разницей в цене

Смогли бы вы понять, какой свитер сколько стоит?
Mercari – крупнейшее в Японии приложение для совершения онлайн-покупок, в котором люди могут продавать и покупать различные новые и бывшие в употреблении товары разных брендов, начиная с одежды и электроники и заканчивая товарами ручной работы (см. рис. 2)
Mercari хотели бы предлагать пользователям объективную цену на их товар при подаче объявления. Это может помочь продавцам оценивать свой продукт до его фактической продажи. Но так как пользователи могут размещать на платформе Mercari практически любые товары, это может быть непростой задачей [1].

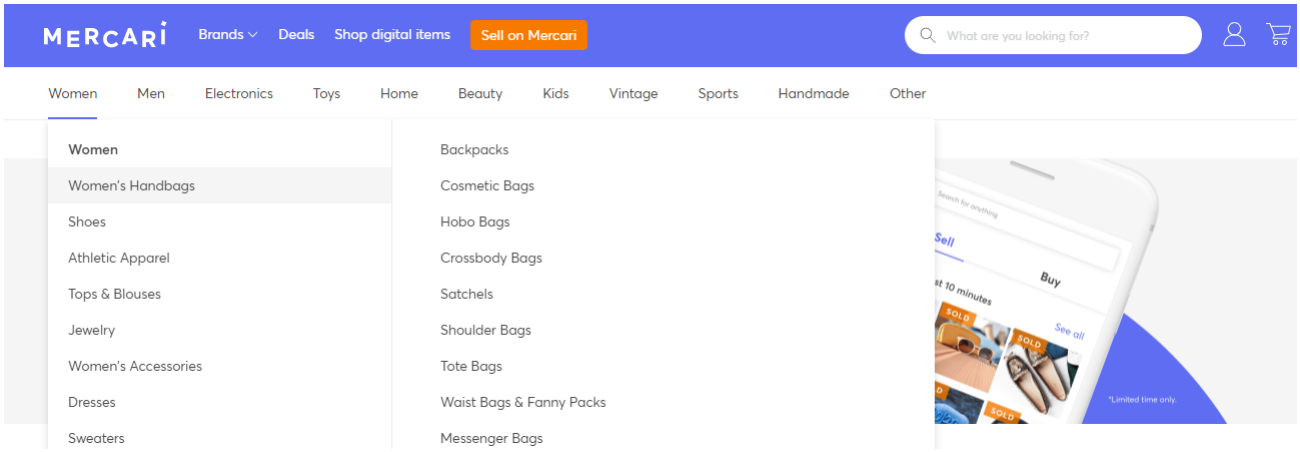

Рисунок 2 — Приложение Mercari

1. Постановка задачи

В рамках данной работы необходимо было реализовать модели решения с помощью языка программирования Python, которые могут автоматически прогнозировать цены на товары в зависимости от переданных им пользователями текстовых описаний продуктов, а также провести сравнение этих моделей [2].

2. Описание данных

	train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete with certificate of authenticity

Рисунок 3 — Несколько примеров из данных

Данные состоят из списка, содержащего 1482535 различных товаров. Каждый товар содержит в себе следующие поля:

- **train_id** - номер товара;
- **name** - текстовое название товара;
- **item_condition_id** ($1 \leqslant$ item_condition_id $\leqslant 5$) - состояние товара, которое указал продавец (1 - самое лучшее состояние, 5 - худшее);
- **category_name** - категория товара в виде трёхуровневой иерархии (разделенная символом /);
- **brand_name** - бренд товара;
- **price** - цена товара, по которой товар был продан (целевая переменная, которую будем предсказывать);
- **shipping** - тип доставки (1 - доставка оплачивается продавцом, 0 - доставка оплачивается покупателем);
- **item_description** - подробное текстовое описание товара.

3. Исследовательский анализ данных

Исследовательский анализ данных является важным процессом выполнения первоначальных исследований данных в целях выявления закономерностей и аномалий, характера и свойств анализируемых данных с помощью сводной статистики и графических представлений.

Для начала проанализируем целевую переменную **price**(см. рис. 4, 5).

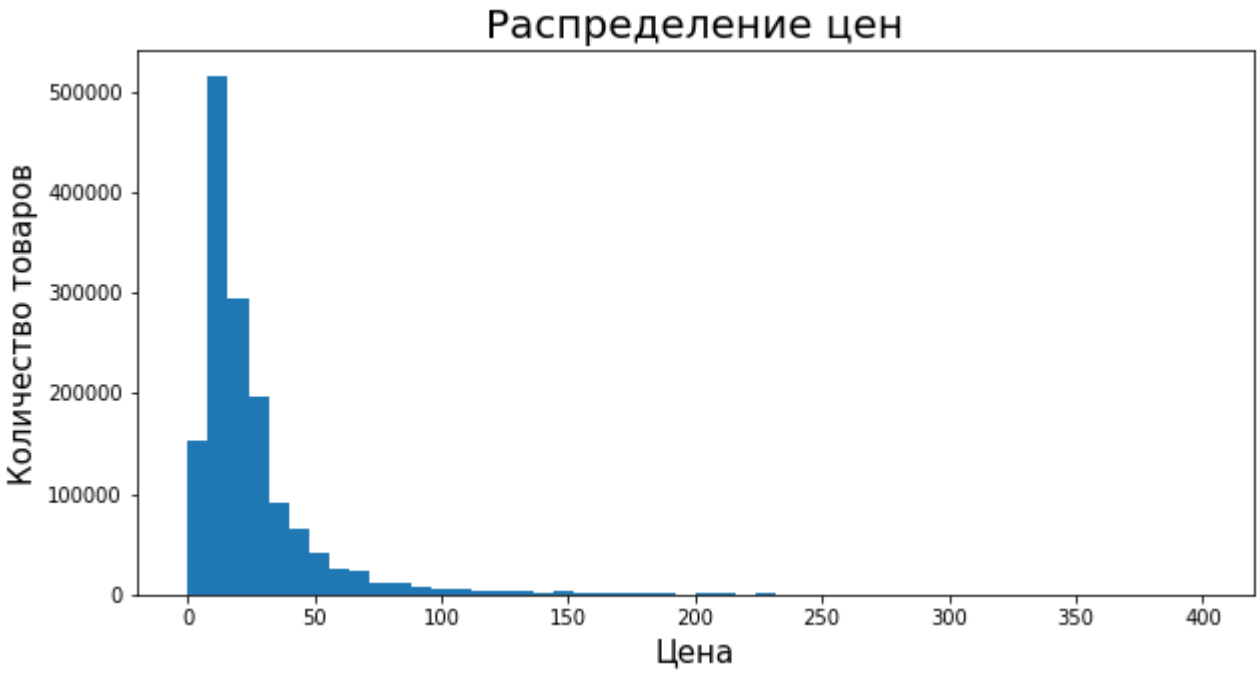

Рисунок 4 — Распределение целевой переменной

count	1.482535e+06
mean	2.673752e+01
std	3.858607e+01
min	0.000000e+00
25%	1.000000e+01
50%	1.700000e+01
75%	2.900000e+01
max	2.009000e+03

Рисунок 5 — Статистические данные

Таким образом, видим, что распределение цены сильно смещено к минимальной границе (75% всех товаров имеют цену ниже 29 долларов). Медианная цена равна 17 долларам. Так же во время анализа было определено, что данные содержат 874 товаров с ценой менее либо равной нуля. Данные товары были удалены из датасета.

Проанализируем **brand_name**. Видим, что сразу же в первых строках бренд не задан (значение NaN). В процентном соотношении бренд не

задан в 42.68 %. Проверим остальные признаки на наличие пропущенных значений (см. рис. 6).

```
В train_id пропущено 0.0 %
В name пропущено 0.0 %
В item_condition_id пропущено 0.0 %
В category_name пропущено 0.42676901388500105 %
В brand_name пропущено 42.675687251902986 %
В price пропущено 0.0 %
В shipping пропущено 0.0 %
В item_description пропущено 0.0002698081326916397 %
```

Рисунок 6 — Пропущенные значения в признаках

Как видим, пропущенные значения присутствуют в **brand_name**, **category_name**, **item_description**. Так как пропущенные значения в brand_name составляют значительную часть данных, сделаем это отдельной категорией, заменив их на текстовое значение 'none'. Товары с пропущенными значениями в category_name и item_description удалим.

Посмотрим на распределение товаров по их брендам (см. рис. 7).

Рисунок 7 — Распределение товаров по брендам

Можем заметить, что бренд не определен у абсолютного большинства товаров. Посмотрим, как определенность бренда влияет на цену(см. рис. 8).

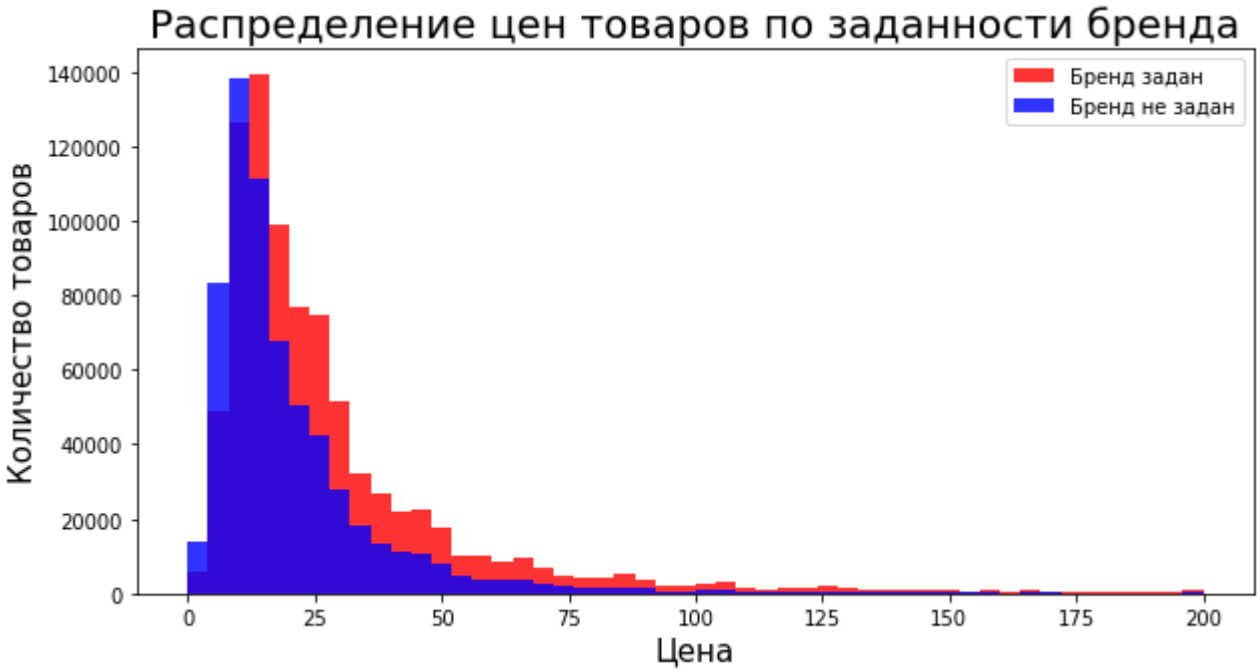

Рисунок 8 — Взаимосвязь определённости бренда и цены

Медианная цена товаров, у которых бренд задан: 20.0.
Медианная цена товаров, у которых бренд не задан: 14.0.
Как мы видим, медианная цена товаров заметно выше, если у них задан бренд. Добавим новый бинарный признак в датасет: 1 - бренд задан, 0 - бренд не задан.
Далее исследуем **shipping** и посмотрим на его связь с ценой.
0: 55.27 %
1: 44.73 %,
где 0 - доставка оплачивается покупателем, 1 - доставка оплачивается продавцом. То есть, доставка 55% товаров оплачивается покупателем, 45% продавцом.
Из графиков видно (см. рис. 9, 10), что медианная цена по одному типу доставки заметно выше другой [3].

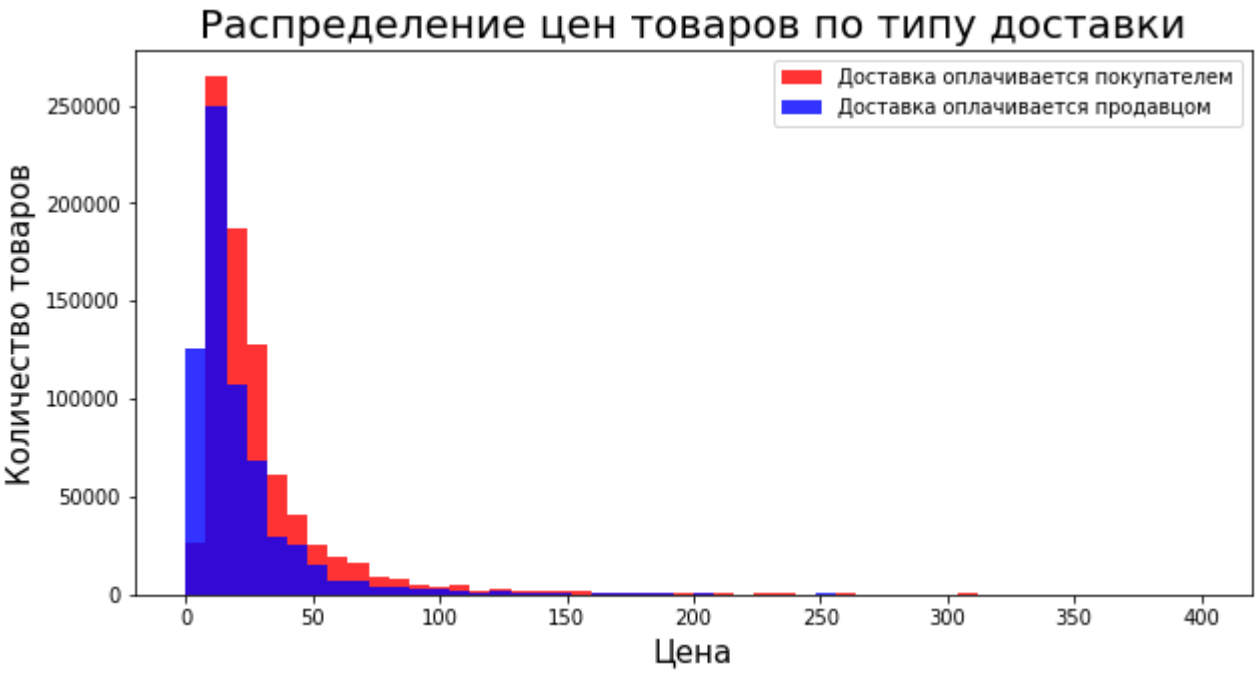

Рисунок 9 — Распределение товаров по типу доставки

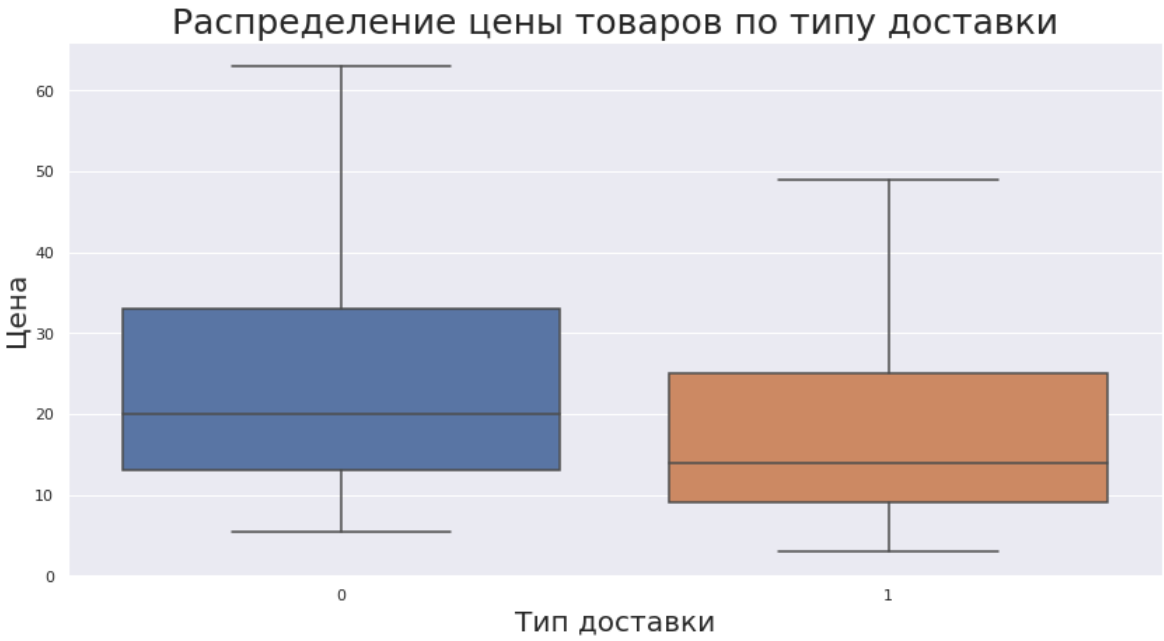

Рисунок 10 — Распределение цены товаров по типу доставки

Медианная цена товаров, доставку которых оплачивает покупатель: 20.0.
Медианная цена товаров, доставку которых оплачивает продавец: 14.0.
Исследуем **item_condition_id**:

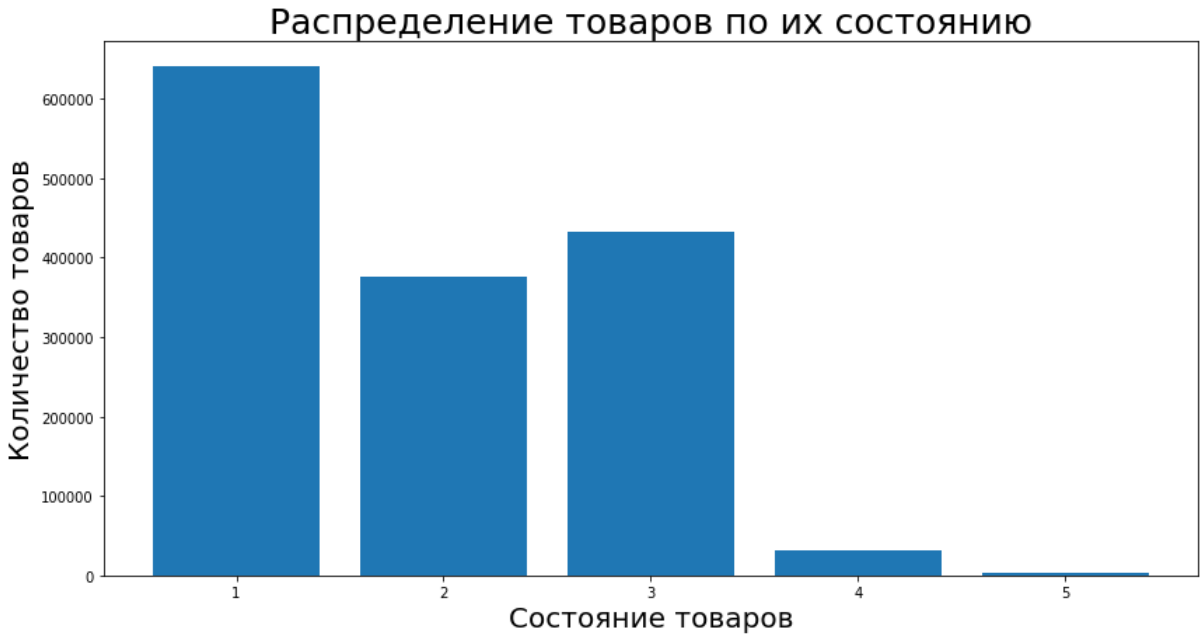

Рисунок 11 — Распределение товаров по их состоянию

1: 43.21 %
2: 25.33 %
3: 29.15 %
4: 2.16 %
5: 0.16 %, где 1 - самое лучшее состояние, 5 - худшее.
Таким образом, можем отметить, что большинство продавцов оценили состояние своего товара как наилучшее. И только 0.16% товаров имеют указанное худшее состояние.
Посмотрим на то, как распределяется цена товаров в зависимости от их состояния (см. рис. 12).

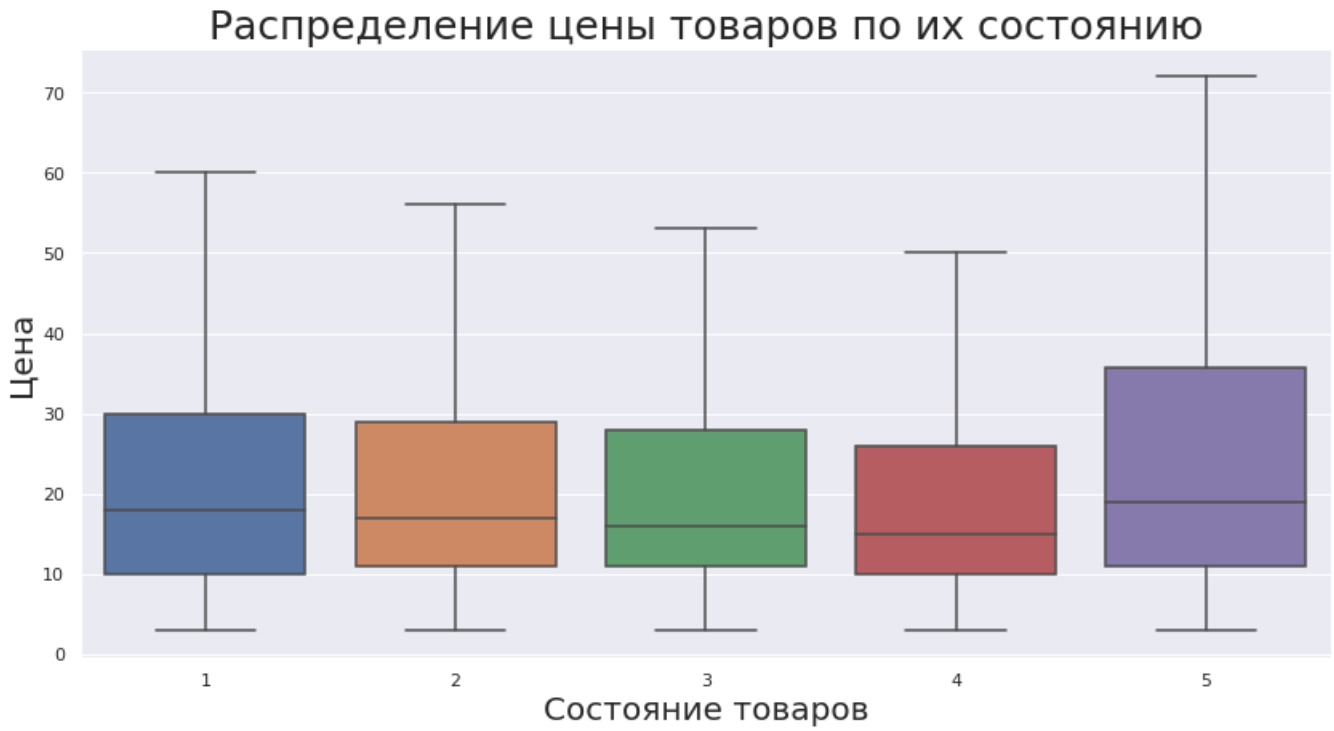

Рисунок 12 — Распределение цены товаров по их состоянию

Медианная цена товаров с состоянием 1: 18.0.
Медианная цена товаров с состоянием 2: 17.0.
Медианная цена товаров с состоянием 3: 16.0.
Медианная цена товаров с состоянием 4: 15.0.
Медианная цена товаров с состоянием 5: 19.0.

Медианная цена предсказуемо соотносится с состоянием товаров: чем лучше состояние, тем медианная цена выше. Исключение лишь составляют товары с самым худшим состоянием. Их медианная цена выше медианной цены товаров с более хорошим состоянием. После просмотра некоторых таких товаров становится ясно, что более высокая цена связана с тем, что эти товары могут являться винтажными вещами известных брендов, частями сломанной электроники и т.д (см. рис. 13).

653	653	Fossil vintage renewal purse	5	Women/Women's Handbags/Shoulder Bag	Fossil	36.0	0	No description yet

Gameboy advance sp ags-101 FOR PARTS	5	Other/Other/Other	none	24.0	0	For parts . Turns on and hold battery charge ....
Hobo International Lauren Wallet	5	Women/Women's Accessories/Wallets	none	24.0	0	Pretty wallet but the back closing clip is bro...

Рисунок 13 — Вещи с худшим состоянием и высокой медианной ценой

Исследуем **category_name**. Как мы могли заметить, категории состоят из троих частей в отношении вложенности. Таким образом, сначала разделим категории на составные части и добавим их в датасет.
Первая подкатегория содержит 10 уникальных значений.
Вторая подкатегория содержит 113 уникальных значений.
Третья подкатегория содержит 863 уникальных значений.
Проанализируем распределение подкатегорий в датасете и посмотрим на их связь с ценой.

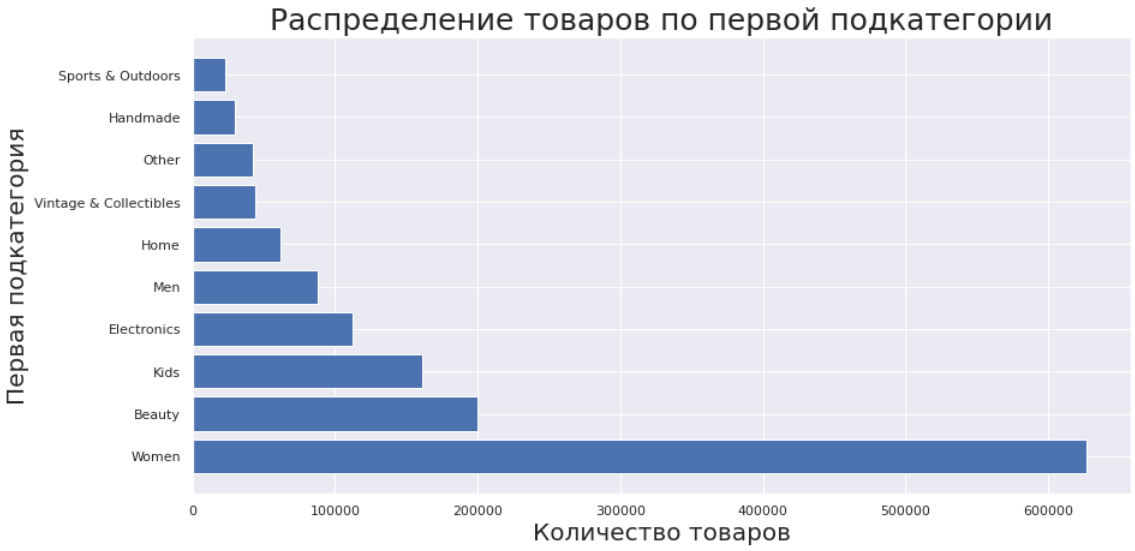

Рисунок 14 — Распределение товаров по первой подкатегории

Как можем заметить, подавляющее большинство товаров имеют первую подкатегорию Women.

```
Women: 45.1 %
Beauty: 14.39 %
Kids: 11.58 %
Electronics: 8.07 %
Men: 6.3 %
Home: 4.48 %
Vintage & Collectibles: 3.18 %
Other: 3.08 %
Handmade: 2.14 %
Sports & Outdoors: 1.67 %
```

Рисунок 15 — Процентное соотношение первых подкатегорий

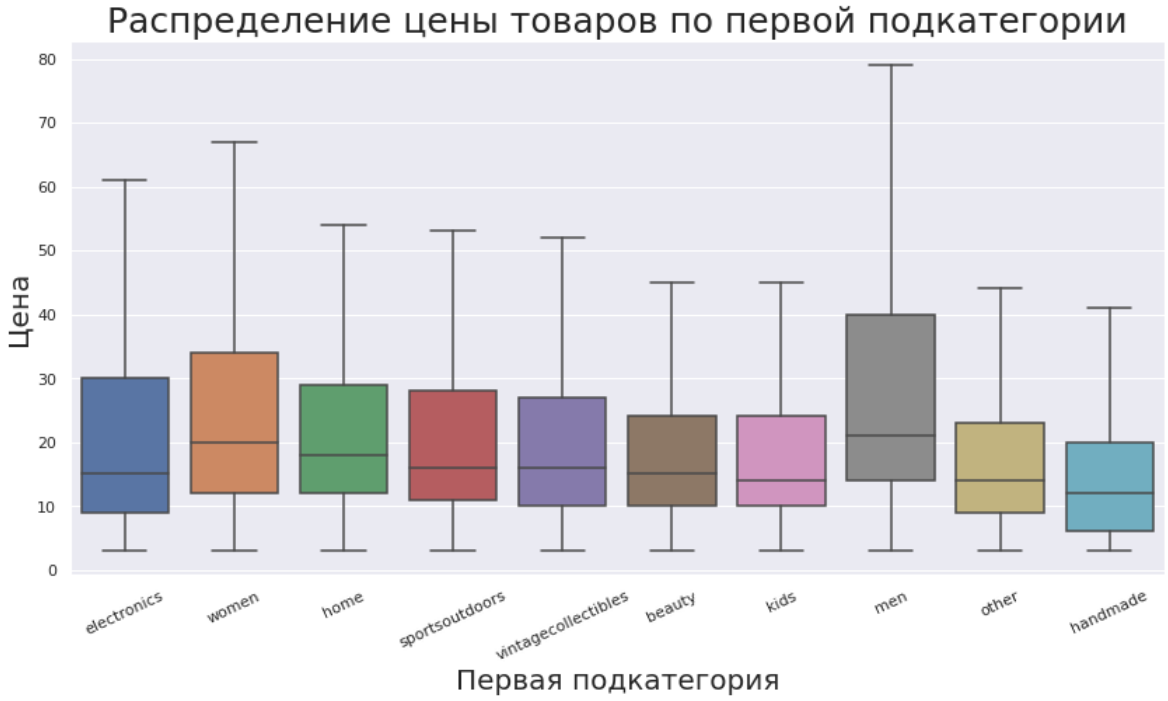

Рисунок 16 — Распределение цен товаров по первой подкатегории

Первые 10 вторых подкатегорий по популярности:

Рисунок 17 — Распределение товаров по второй подкатегории

```
Athletic Apparel: 9.06 %
Makeup: 8.41 %
Tops & Blouses: 7.21 %
Shoes: 6.78 %
Jewelry: 4.17 %
Toys: 3.92 %
Cell Phones & Accessories: 3.59 %
Dresses: 3.09 %
Women's Handbags: 3.09 %
Women's Accessories: 2.86 %
```

Рисунок 18 — Процентное соотношение вторых подкатегорий

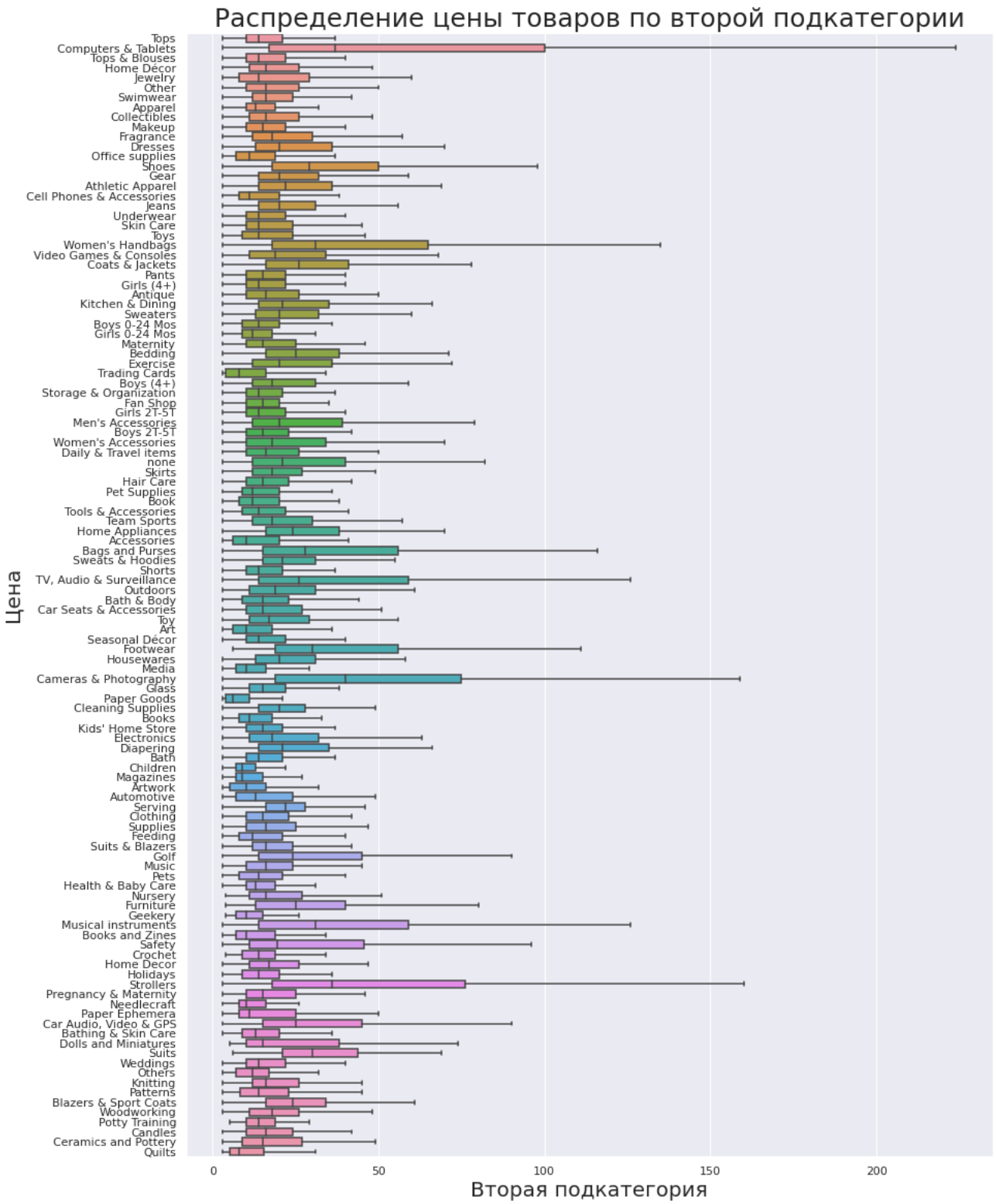

Рисунок 19 — Распределение цен товаров по второй подкатегории

4. Предобработка данных

Так как **item_description** и **name** содержат неструктурированные текстовые данные, то необходимо обработать их. Удалим пунктуационные знаки и символы, которые не представляют из себя никакой ценности для обучения. Приведем все слова к нижнему регистру для однообразности и удалим стоп-слова (союзы, междометия, артикли и т.д.), которые не несут полезной информации, а являются лишь "шумом". Затем приведем все слова к единой форме, используя стеммер Портера. Данный алгоритм отсекает у слова суффиксы и окончания, оставляя основу слова, применяя некоторые правила, которые опираются на особенности используемого языка [4].
Примеры правил для английского языка:

- отсечь ’ed’, если слово оканчивается на ’ed’;
- отсечь ’s’, если слово оканчивается на ’s’;
- отсечь ’ing’, если слово оканчивается на ’ing’.

Пример: описанные выше преобразования будут действовать следующим образом:
“The boy’s dogs are different size.” $\Longrightarrow$ “boy dog differ size”
Затем приведем категориальные признаки **category**, **brand_name**, **item_condition_id**, **shipping** к числовому виду, используя one-hot кодирование. В результате этого кодирования каждому признаку будет соответствовать N новых бинарных признаков, где N — число всех возможных категорий в этом признаке (единица на месте категории текущего признака и нули на всех остальных местах).

Листинг 4.1. Пример one-hot кодирования

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np
oh = OneHotEncoder(handle_unknown='ignore')
X = np.array(['dog',
              'cat',
              'mouse'])
X = oh.fit_transform(X.reshape(-1,1))
print(oh.categories_)
print(X.toarray())
```

```
[array(['cat', 'dog', 'mouse'], dtype='<U5')]
[[0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

Рисунок 20 — Вывод примера

Приведем признак **name** из текстового вида в числовой. Для начала токенизируем каждый признак в датасете на отдельные слова, на биграммы (пары соседних слов) и на триграммы (тройки соседних слов). Для векторизации наших признаков будем использовать представление мешок слов с подсчётом числа вхождений токенов в текущий документ. В результате применения этого представления каждому признаку будет соответствовать N новых признаков, где N — число уникальных токенов во всей коллекции документов.

Листинг 4.2. Пример представления текстовых данных в числовом виде с помощью мешка слов

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = ['This is the first document.',
          'This document is the second document.',
          'And this is the third one.',
          'Is this the first document?']
cv = CountVectorizer()
X = cv.fit_transform(corpus)
print(cv.get_feature_names())
print(X.toarray())
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

Рисунок 21 — Вывод примера

Приведем признак **item_description** из текстового вида в числовой, используя представление мешок слов с подсчётом частотности слов (чем чаще слово встречается во всём датасете - тем оно менее важно) с помощью алгоритма TF-IDF.

TF-IDF используется для оценки важности слова в контексте документа, являющегося частью коллекции документов. Вес некоторого слова пропорционален частоте употребления этого слова в документе (TF) и обратно пропорционален частоте употребления слова во всех документах коллекции (IDF).

$$TF_a = k/N,$$

где k - количество раз, когда термин a встретился в документе,

N - количество всех слов в документе.

$$IDF_a = log(D/m),$$

где D - общее количество документов, m - количество документов, в которых встречается термин .

$$TFIDF_a = TF_a * IDF_a$$

Сначала **item_description** токенизируем на отдельные слова, на биграммы (пары соседних слов) и на триграммы (тройки соседних слов), после чего применяем TF-IDF.

Листинг 4.3. Пример представления текстовых данных в числовом виде с помощью TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = ['This is the first document.',
          'This document is the second document.']
tv = TfidfVectorizer(ngram_range=(1,2), max_features=10)
X = tv.fit_transform(corpus)
print(tv.get_feature_names())
print(X.toarray())
```

```
['document', 'document is', 'first', 'first document', 'is', 'is the', 'second', 'second doc
[[0.33425073 0.         0.46977774 0.46977774 0.33425073 0.33425073
  0.         0.         0.33425073 0.33425073]
 [0.53594084 0.37662308 0.         0.         0.26797042 0.26797042
  0.37662308 0.37662308 0.26797042 0.26797042]]
```

Рисунок 22 — Вывод примера

5. Метрика оценки

Для оценки точности наших моделей решения данной задачи использовалась метрика RMSLE (Root Mean Squared Logarithmic Error).

$$RMSE(y_i, ypred_i) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - ypred_i))^2}{n}}$$

$$RMSLE(y_i, ypred_i) = \sqrt{\frac{\sum_{i=1}^{n}(\log(y_i + 1) - \log(ypred_i + 1))^2}{n}} =$$

$$= RMSE(\log(y_i + 1), \log(ypred_i + 1)),$$

где n - общее количество примеров в датасете,
y_i - фактическая цена,
$ypred_i$ - прогноз цены от модели.
Особенности данной метрики оценки:

- устойчива к воздействию выбросов;
  Товары, которые будут иметь очень высокое значение цены, будут иметь немного более высокую ошибку по сравнению с товарами с низкой стоимостью, потому что RMSLE имеет логарифмическое выражение в метрике, которое смягчает этот эффект. Если же оценивать с помощью RMSE, то такие выбросы могут исказить модель.

- учитывает только относительную ошибку, а не абсолютную;

  $$\log x - \log y = \frac{\log x}{\log y}$$

  Отсюда видно, что благодаря свойству логарифмов RMLSE можно рассматривать как относительную ошибку между прогнозируемыми и фактическими значениями, а масштаб ошибки не имеет значения.

- влечет за собой большее наказание за недооценку целевой переменной, чем за переоценку (см. рис. 23);
  Проще говоря, больше штрафа возникает, когда прогнозируемое значение меньше фактического значения, чем, когда прогнозируемое значение больше фактического значения. Это особенно может быть полезно для некоторых бизнес-проектов. Например, если модель регрессии, которую мы построили, даёт немного более высокую оценку цены для определенных товаров, чем фактическая, то эта небольшая переоценка является приемлемой. Если же мы недооцениваем цену товара, то пользователю, вероятно, не будет выгодно продавать свой продукт на нашей площадке [5].

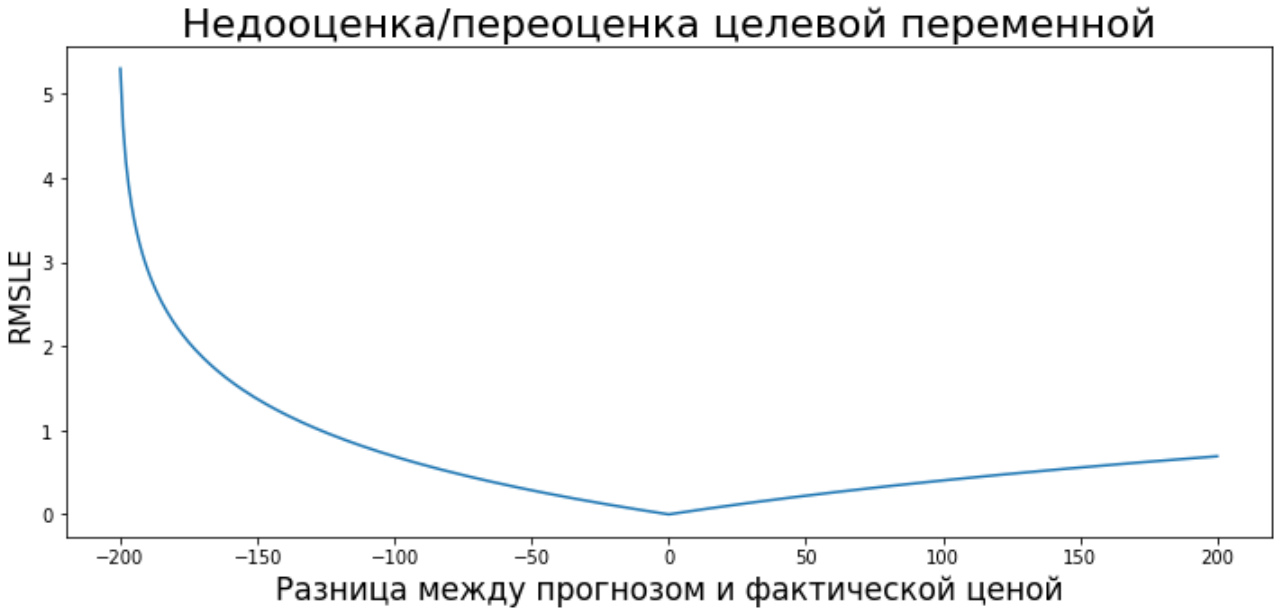

Рисунок 23 — Сравнение роста RMSLE при недооценке и переоценке целевой переменной

На левой половине графика RMSLE мы можем заметить, как быстро увеличивается ошибка по мере недооценки целевой переменной, в то время как на правой половине при переоценке ошибка растёт медленнее.

6. Моделирование решений

Объединяем все наши преобразованные признаки вместе в одну матрицу. Затем разделяем признаки и цены на тренировочный и тестовый наборы данных в отношении 0.8 к 0.2 соответственно. Так как в качестве функции оценки точности модели в этой задаче используется $RMSLE(y_i, ypred_i) = \sqrt{\frac{\sum_{i=1}^{n}(\log(y_i+1)-\log(ypred_i+1))^2}{n}}$, считаем $\log(p+1)$, где p - цена, для того, чтобы использовать среднеквадратичную логарифмическую ошибку $RMSE(y_i, ypred_i) = \sqrt{\frac{\sum_{i=1}^{n}(y_i-ypred_i))^2}{n}}$ для оценки моделей.

Данная задача представляет собой задачу регрессионного моделирования, так как целевая переменная имеет непрерывный характер. То есть, наше решение должно моделировать зависимость между независимыми переменными (нашими признаками) и зависимой переменной (наша целевая переменная price). Таким образом, модель решения рассматривает характеристики продуктов, которые были похожими, а также цену, по которой товар был продан, и предлагает схожие цены.

Для начала нам нужно задать основу для сравнения результатов. Так как мы работаем с задачей регрессии, то можем использовать медиану всех цен в тренировочном наборе в качестве результата для всех прогнозов. Эта основа будет служить для нас некоей базовой наивной моделью решения, с помощью которой мы посчитаем ошибку на тестовом наборе. После оценки точности данной модели была достигнута оценка RMSLE = 0.749506. Полученная оценка будет отправной точкой, которую мы будем улучшать с помощью более сложных моделей. Если мы не сможем получить точность больше, чем у нашей базовой модели, то это может говорить о том, что нужно выбрать другой подход к решению или что машинное обучение может быть вообще неприменимо для данной задачи [6].

Для подбора гиперпараметров для наших моделей использовалась

функция случайного поиска RandomizedSearchCV из sklearn [7]. При использовании случайного поиска создается сетка гиперпараметров, модель обучается и тестируется на некоторых случайных комбинациях гиперпараметров из этой сетки, и выбирается лучшая комбинация на основе оценки качества на тестовом наборе. Так как тестовый набор данных нам нужен для оценки итоговой проверки точности обученной модели, то нам необходимы еще тестовые выборки для проверки качества обученных моделей при подборе гиперпараметров. Для этого существует техника перекрестной проверки. При использовании перекрестной проверки мы разделяем тренировочный датасет на k частей. Затем на k-1 частях данных проводится обучение модели, а оставшаяся часть используется для тестирования. Данный процесс повторяется k раз. В результате каждая из k частей тренировочного набора используется для тестирования (см. рис. 24). После выполнения k раз вышеописанного процесса результаты тестирования, полученные на каждой итерации, усредняются.

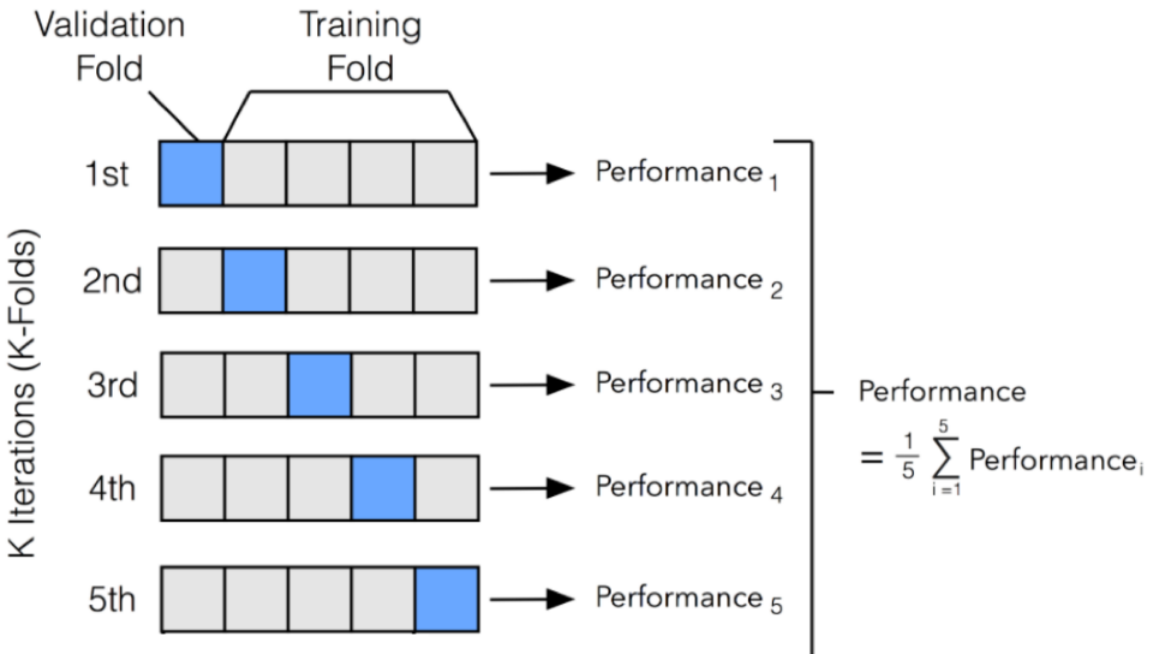

Рисунок 24 — Перекрёстная проверка

В качестве первой модели решения выбрана модель SGDRegressor из библиотеки sklearn. Это модель представляет собой модель линейной регрессии.

$$y = w_0 + \sum_{i=1}^{n}(w_i x_i) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n,$$

где y – целевая прогнозируемая переменная, x_i – признаки, w_i – коэффициенты. Обучение происходит с помощью алгоритма стохастического градиентного спуска. То есть, на каждом шаге выбирается случайный элемент из обучающей выборки, считается градиент функции потерь (направление наискорейшего роста функции), затем обновляются веса в направлении антиградиента, чтобы минимизировать функцию потерь. В результате реализации данной модели была достигнута оценка на тестовом наборе = 0.482615.

Следующей моделью решения была реализована модель Ridge из библиотеки sklearn (также известна как регуляризация Тихонова). Это регрессионная модель, которая в качестве функции потерь при обучении использует функцию наименьших квадратов вместе с L2-регуляризацией. То есть, другими словами, модель минимизирует следующую функцию ошибки:

$$E(y_i, ypred_i) = \sum_{i=1}^{N}(y_i - ypred_i)^2 + \lambda \sum_{j=0}^{k} w_j^2 =$$

$$= \sum_{i=1}^{N}(y_i - \sum_{j=0}^{k} x_j w_j)^2 + \sum_{j=0}^{k} w_j^2,$$

где слагаемое $\sum_{j=0}^{k} w_j^2$ - L2-регуляризация, N – число примеров в обучающей выборке, y_i – фактическое значение предсказываемой переменной, $ypred_i$ - предсказанное моделью значение, k – число признаков, x_i - признаки, w_i – обучаемые коэффициенты, λ - сила регуляризации.

Регуляризация помогает уменьшить степень переобучения модели. Переобучение - это ситуация, при которой с повышением сложности модели ошибка на обучающей выборке продолжает снижаться, а ошибка на тестовой выборке увеличивается (см. рис. 25).

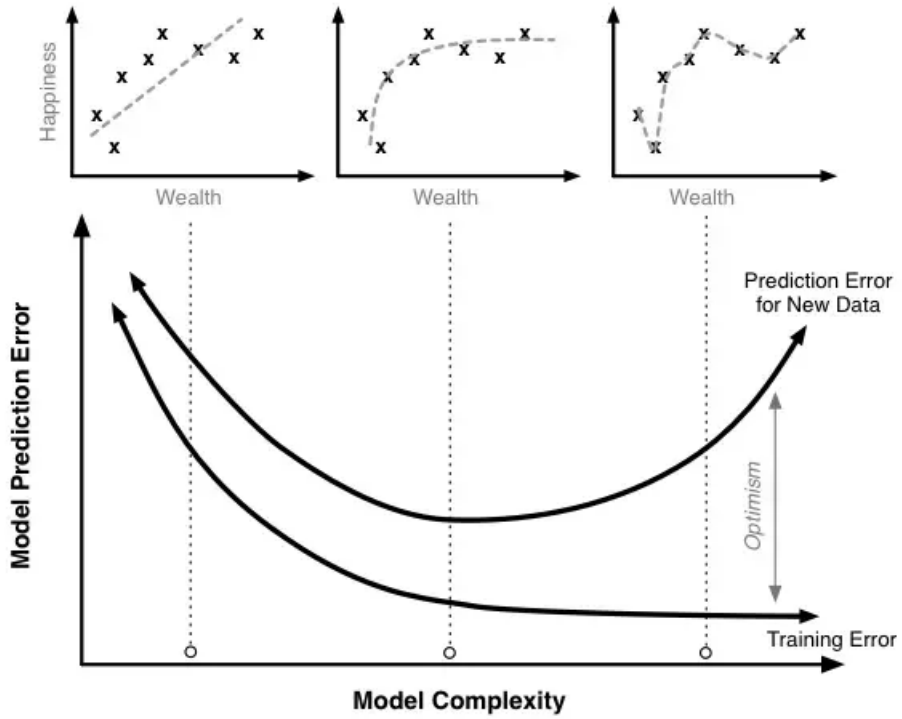

Рисунок 25 — Взаимосвязь сложности модели и ошибки предсказаний при недообучении и переобучении

То есть, при переобучении модель фактически «запоминает» данные из тренировочного набора, из чего следует плохая точность на новых данных из тестирующего набора, которые модель еще не видела. Чтобы значения коэффициентов модели не становились большими, при которых модель начинает запоминать данные, а не обобщать, регуляризация штрафует коэффициенты. L2-регуляризация штрафует коэффициенты модели, добавляя сумму их квадратов к ошибке.
С помощью данной модели была достигнута ошибка на тестовом наборе = 0.450486.
В качестве следующего решения была реализована модель LightGBM. LightGBM - это реализация алгоритма градиентного бустинга.

Градиентный бустинг - это метод машинного обучения, который создает модель прогнозирования в виде множества слабых предсказателей, обычно деревьев решений. Деревья решений - это техника разделения данных на основе признаков для прогнозирования некоторого значения. В каждой ветви в дереве решений данные разделяются на одну из двух групп. После чего каждому листу в дереве решений присваивается прогнозируемое значение (см. рис. 26).

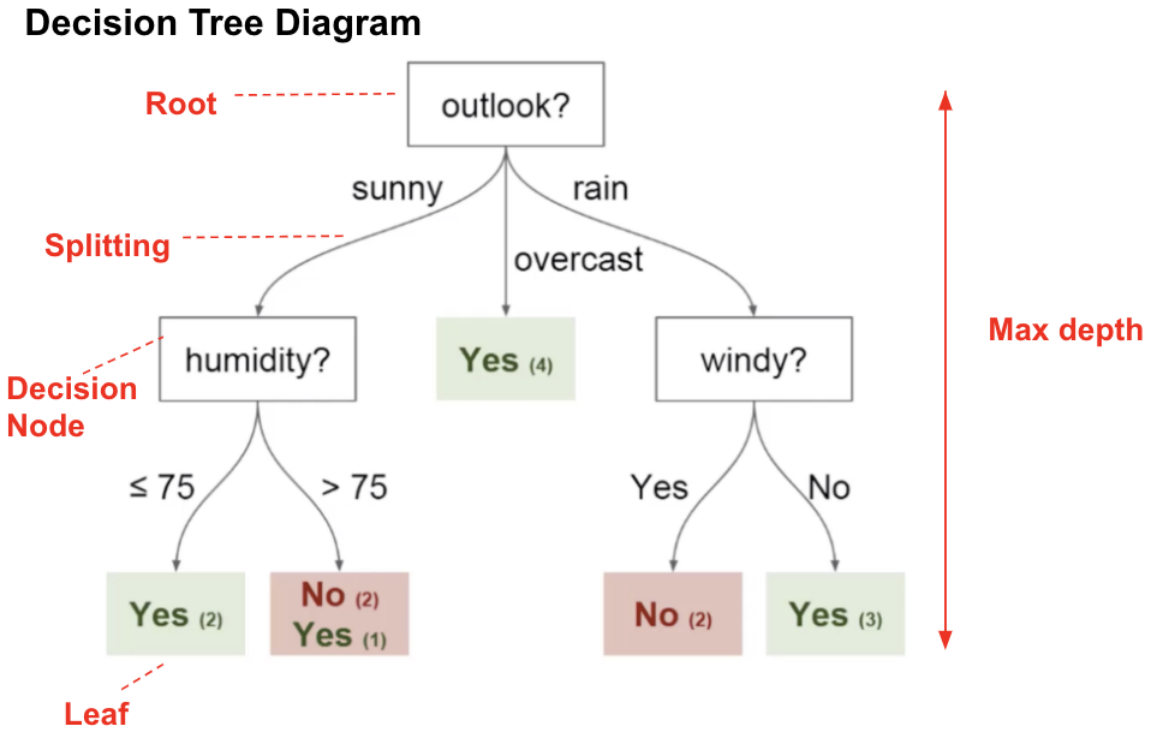

Рисунок 26 — Пример дерева решений

Однако одно дерево решений склонно к переобучению и поэтому не будет хорошо предсказывать на новых данных. Поэтому обычно вместо одного дерева решений используется множество деревьев. В основе градиентного бустинга лежит следующая идея: объединить предсказания нескольких деревьев решений, сложив их вместе. Например, в нашей задаче прогнозируемая цена для любого товара будет суммой предсказаний цен каждого отдельного дерева решений. Градиентный бустинг обучается итеративно, по одному дереву за итерацию. Дерево решений будет минимизировать функцию потерь с помощью ре-

курсивного разделения данных, пока не будет достигнут некоторый предел - например, глубина дерева или число листьев. Затем обученное дерево решений добавляется в модель, уже добавленные деревья в модели не изменяются. Для минимизации функции потерь при добавлении деревьев используется алгоритм градиентного спуска. Обучение прекращается, если в итоговую модель добавляется фиксированное количество деревьев или если потеря достигнет приемлемого уровня.
Достигнутая данной моделью ошибка на тестовом наборе = 0.44503.
В качестве последней модели были объединены модели Ridge и LightGBM. К данным были добавлены прогнозы Ridge и с помощью полученного датасета была обучена модель LightGBM. С помощью данного ухищрения была улучшена оценка, которая составила 0.43601.

7. Сравнение моделей

Таблица 1 — Сравнение реализованных моделей

Модель	Подбор гиперпараметров	Обучение	Время работы	RMSLE на обучающей выборке	RMSLE на тесте
Baseline	-	-	-	-	0.74950
SGDRegressor	2 мин	8.3 сек	0.05231 сек	0.47922	0.48261
Ridge	41.2 мин	1.7 мин	0.05109 сек	0.42595	0.45048
LightGBM	277.6 мин	22.6 мин	21.9 сек	0.39807	0.44503
Ridge+LightGBM	277.6 мин	21.6 мин	22.6 сек	0.37899	0.43601

Как можем заметить, самая быстро обучаемая модель показала худший результат предсказания. Самая долгая в обучении модель же напротив показала себя лучше всех в обобщении данных.

Заключение

В процессе работы было произведено исследование данных, с помощью которого были выявлены некоторые дополнительные за-

висимости между данными. В результате работы были реализованы несколько моделей решений задачи определения цены товаров по их текстовому описанию. Также было приведено сравнение этих моделей между собой.

Список литературы

1. Mercari price suggestion challenge. — 2018. — URL: https://www.kaggle.com/c/mercari-price-suggestion-challenge.
2. *Саммерфилд М.* Программирование на Python 3. Подробное руководство. — СПб. : СимволПлюс, 2009.
3. Understanding Boxplots. — 2018. — URL: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51.
4. Stemming. — 2019. — URL: https://en.wikipedia.org/wiki/Stemming.
5. All about the metric: RMSLE. — 2019. — URL: https://www.kaggle.com/c/ashrae-energy-prediction/discussion/113064.
6. How To Get Baseline Results And Why They Matter. — 2014. — URL: https://machinelearningmastery.com/how-to-get-baseline-results-and-why-they-matter/.
7. Библиотека scikit-learn для машинного обучения. — URL: https://scikit-learn.org/stable/.

A. Приложение

Ссылка на репозиторий: https://github.com/nisded/graduation_work