Стоимость подержанного автомобиля

анализ процесса разработки модели

Вера Мельникова

Студент DS+ 19

Постановка задачи

Принять участие в Мастерской, в рамках которой:

- поработать с реальными данными о продажах автомобилей на вторичном рынке
- разработать модель для предсказания стоимости автомобиля на вторичном рынке
- выяснить, какие характеристики больше всего влияют на итоговую стоимость автомобиля
- показать хорошие результаты в соревновании на Kaggle

Этапы решения

1. Предварительное знакомство с данными
2. Заполнение пропусков и обработка дубликатов
3. Разведывательный анализ
4. Преобразование характеристик, добавление новых
5. Подбор гиперпараметров моделей
6. Ансамбль моделей
7. Анализ важности признаков моделей
8. Общий вывод

Предварительное знакомство с данными

train.csv	test.csv

`<class 'pandas.core.frame.DataFrame'>`
`RangeIndex: 440236 entries, 0 to 440235`
`Data columns (total 15 columns):`
	`<class 'pandas.core.frame.DataFrame'>`
	`RangeIndex: 110058 entries, 0 to 110057`
	`Data columns (total 14 columns):`

Наличие пропусков в данных: year 0 make 8043 model 8123 trim 8337 body 10393 transmission 51461 vin 0 state 0 condition 9405 odometer 69 color 586 interior 586 seller 0 sellingprice 0 saledate 0 dtype: int64 Наличие дубликатов в данных: 0	Наличие пропусков в данных: year 0 make 2061 model 2079 trim 2114 body 2594 transmission 13011 vin 0 state 0 condition 2379 odometer 19 color 158 interior 158 seller 0 saledate 0 dtype: int64 Наличие дубликатов в данных: 0

Заполнение пропусков и дубликатов

Датасет, доступный для обучения, очень сильно по своим характеристикам похож на тестовый. Поэтому:

- будем **обрабатывать** обучающий датасет и тестовый **одновременно**;
- **закономерности** данных, необходимые для выработки правил заполнения пропусков, будем **вырабатывать на тренировочном** датасете, а **применять к обоим** датасетам
- В некоторых случаях кажется возможным **заполнить пропуски по** имеющимся **данным других столбцов**. Например, зная модель машины, можно заполнить пропуски в данных о типе кузова и т.п.

Обработка пропусков и дубликатов

Производитель

несколько вариантов написания.

Количество уникальных значений сократилось с 93 (86) до 55 (56)

пропуски заполнила unknown, так как модель и производитель обычно отсутствовали одновременно

```
# словарь для замены неявных дубликатов
make_to_replace = {
    "mercedes-b" : "mercedes",
    "mercedes-benz" : "mercedes",
    "land rover" : "landrover",
    "vw" : "volkswagen",
    "gmc truck" : "gmc",
    "dodge tk" : "dodge",
    "mazda tk" : "mazda",
    "ford truck" : "ford"
}

# заменяем повторяющиеся значения на одно
data['make'].replace(make_to_replace, inplace=True)
test['make'].replace(make_to_replace, inplace=True)
```

Обработка пропусков и дубликатов

Тип коробки передач

с 51 461

до 35 683

- 30 %

остальные пропуски заполнила unknown

```
# списки моделей по типу коробки передач
list_of_models_a = []
list_of_models_m = []

for model in data.query('transmission.isna()')['model'].unique():
    unique_transmissions = data[data['model'] == model]['transmission'].unique()
    if len(unique_transmissions) == 2: # т.е. только пап и еще одно
        if 'automatic' in unique_transmissions:
            list_of_models_a.append(model)
        elif 'manual' in unique_transmissions:
            list_of_models_m.append(model)
```

```
# поменяем пропуски на значение 'automatic'
indexes = data.query('transmission.isna() and model in @list_of_models_a').index
data.loc[indexes, 'transmission'] = 'automatic'
```

```
# поменяем пропуски на значение 'automatic'
indexes = test.query('transmission.isna() and model in @list_of_models_a').index
test.loc[indexes, 'transmission'] = 'automatic'
```

Обработка пропусков и дубликатов

Тип кузова

с 46

до 10

далее по аналогии с коробкой передач заполнила 12 % пропусков, остальные unknown

```
data['body'].replace('koup', 'coupe', inplace=True)
test['body'].replace('koup', 'coupe', inplace=True)
```

```
# напишем функцию, котрая убирает несущественные детали из значений столбца 'body'
def clean_body_type(row):
    '''
    сокращает название типа кузова автомобиля
    '''
    try:
        body_types = [
            'cab',
            'convertible',
            'coupe',
            'sedan',
            'van',
            'wagon'
        ]
        for b_type in body_types:
            if b_type in row:
                return b_type
        return row
    except:
        return row
```

Обработка пропусков и дубликатов

Цвет кузова и салона

два типа пропусков

Уровень отделки

пропуски заполнила unknown

```
# словарь для замены пропусков
color_to_replace = {
    "–" : "unknown",
    np.nan : "unknown"
}

# замена
data['color'].replace(color_to_replace, inplace=True)
test['color'].replace(color_to_replace, inplace=True)
```

```
data['trim'].fillna('unknown', inplace=True)
test['trim'].fillna('unknown', inplace=True)
```

Обработка пропусков и дубликатов

Пробег и **состояние авто**

заполнила медианным значением для каждого года выпуска

```
# найдем медианное значение пробега по годам
odometr_median = data.groupby('year')['odometer'].median()
odometr_median
```

```
# для каждого года заполняем пропуск медианой
for year in data['year'].unique():
    data.loc[(data['year'] == year) & (data['odometer'].isna()), 'odometer'] = odometr_median[year]
```

```
# для каждого года заполняем пропуск медианой
for year in test['year'].unique():
    test.loc[(test['year'] == year) & (test['odometer'].isna()), 'odometer'] = odometr_median[year]
```

Обработка выбросов

Удалила выбросы:

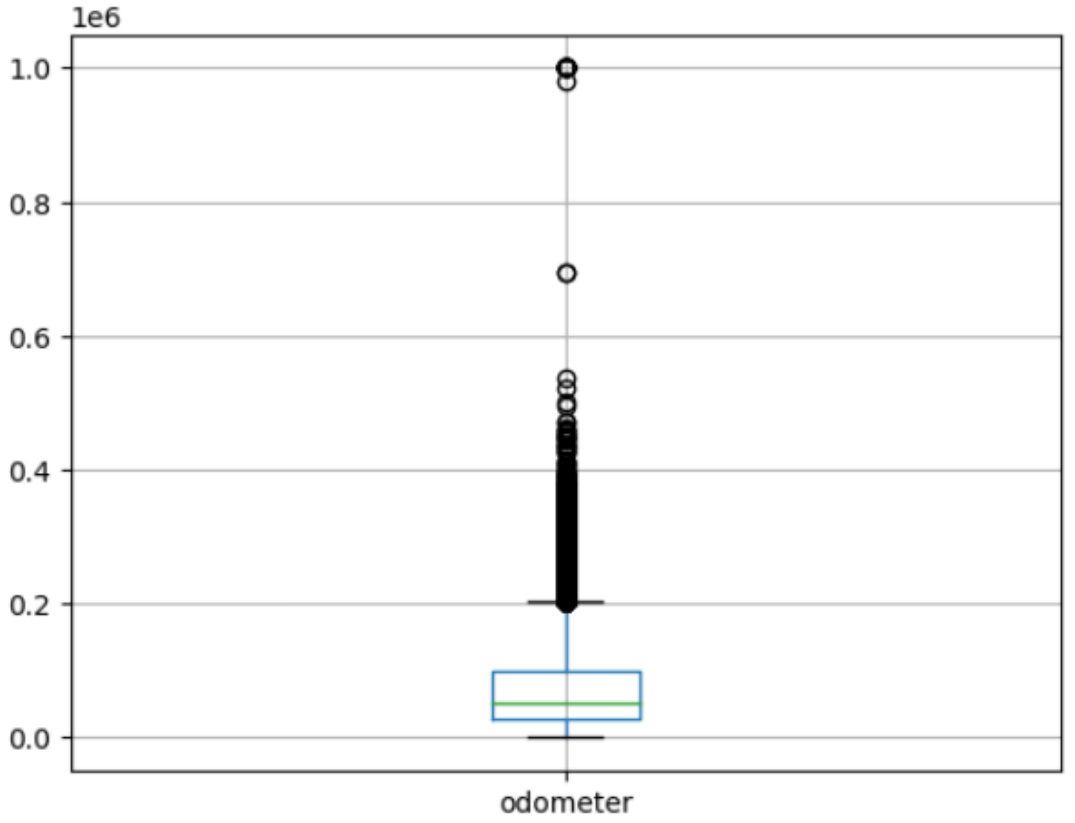
отрицательный **возраст** авто

состояние авто

пробег

цена

Цель: уменьшить зашумленность данных




```
# процент авто с пробегом более 200 000 км
len(data[data['odometer'] >= 200_000]) / data.shape[0] * 100
```

```
2.052874485340563
```

Генерация новых признаков

возраст авто

медианная цена по **штату**

медианная цена **модели**

медианная цена по **производителю**

медианная цена по **продавцу**

медианная цена по **году** производства

регион производства

```python
# функция определяет регион, где произведен автомобиль
def region_from_vin(row):
    if row[0] in '12345':
        return 1 # nourth america
    if row[0] in '67':
        return 2 # oceania
    if row[0] in '89':
        return 3 # south america
    if row[0] in 'abcdefgh':
        return 4 # africa
    if row[0] in 'jklmn':
        return 5 # asia
    else:
        return 6 # europe
```

Генерация новых признаков

Класс авто

Добавим 3 класса авто в зависимости от стоимости. Класс будем определять по медианной стоимости модели.

```
# треть диапазона разброса медианных цен разных моделей авто
(data['model_m_price'].max() - data['model_m_price'].min()) / 3
```

```
11366.666666666666
```

```
# для обучающего набора данных
data.loc[data['model_m_price'] > 22_800, 'model_class'] = 1
data.loc[((data['model_m_price'] > 11_400) & (data['model_m_price'] <= 22_800)), 'model_class'] = 2
data.loc[data['model_m_price'] <= 11_400, 'model_class'] = 3
```

```
# для тестового набора данных
test.loc[test['model_m_price'] > 22_800, 'model_class'] = 1
test.loc[((test['model_m_price'] > 11_400) & (test['model_m_price'] <= 22_800)), 'model_class'] = 2
test.loc[test['model_m_price'] <= 11_400, 'model_class'] = 3
```

Подготовка данных к обучению на них модели

посмотрела **корреляцию** признаков

удалила часть тех **признаков**, для которых рассчитала медианные значения по подгруппам

категориальные признаки: **OrdinalEncoder**

масштабирование: **StandardScaler**

```
data = data.drop([
    'year', # вместо него будем использовать year_m_price
    'age', # вместо него будем использовать year_m_price
    'vin', # неинформативный
    'state', # вместо него будем использовать stae_m_price
    'seller', # вместо него будем использовать seller_m_price
    'saledate', # вместо него будем использовать year_m_price, month_m_price
    'saledate_new', # вместо него будем использовать year_m_price, month_m_price
    'sale_month', # очень низкая корреляция с целевым признаком
    'sale_wd', # очень низкая корреляция с целевым признаком
    'sellingprice' # целевой признак
], axis=1)
```

```
# будем использовать OrdinalEncoder
enc = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
```

Подбор моделей

Попробовала несколько моделей без дополнительных настроек

отобрала те, которые дали **лучшие** результаты

настроила с помощью GridSearchCV

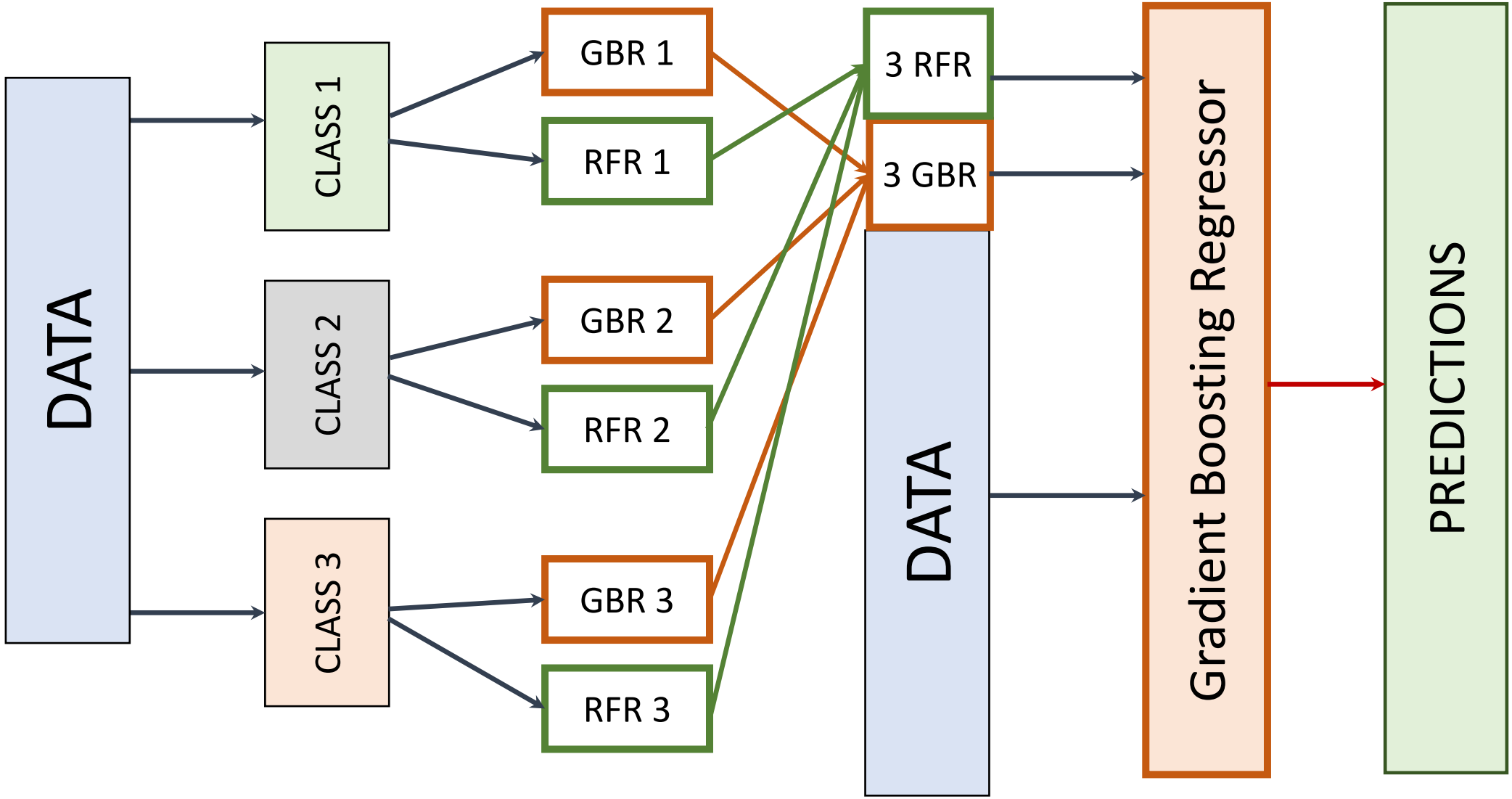
Модель	MAPE, %
GradientBoostingRegressor	29
RandomForestRegressor	29
SGDRegressor	42
ElasticNet	48
CatBoostRegressor	67
LInearRegression	70

```
mape_scorer = make_scorer(mean_absolute_percentage_error, greater_is_better=False)
```

Комментарий ревьюера ⚠:

Здесь все проще - указываем **scoring** = **'neg_mean_absolute_percentage_error'**, а полный перечень метрик, которые принимает GridSearchCV здесь 😊

Ансамбль моделей



Ансамбль моделей

Разбила датасеты на три и для каждого **обучила** свою **модель**, ориентированную на этот класс.

```
# разбиваем данные по классам
class_1 = data[data['model_class'] >= 0]
class_2 = data[(data['model_class'] >= -1) & (data['model_class'] < 0)]
class_3 = data[data['model_class'] < -1]

target_1 = target[class_1.index]
target_2 = target[class_2.index]
target_3 = target[class_3.index]
```

```
# формируем предсказания бустинга
gbr_class_1 = cross_val_predict(gbr_model, class_1, target_1, cv=3)
gbr_class_2 = cross_val_predict(gbr_model, class_2, target_2, cv=3)
gbr_class_3 = cross_val_predict(gbr_model, class_3, target_3, cv=3)
```

```
# формируем предсказания случайного леса
rfr_class_1 = cross_val_predict(rfr_model, class_1, target_1, cv=3)
rfr_class_2 = cross_val_predict(rfr_model, class_2, target_2, cv=3)
rfr_class_3 = cross_val_predict(rfr_model, class_3, target_3, cv=3)
```

Ансамбль моделей

Собрала три класса снова в один датасет и отсортировала индексы

на последнем этапе обучила **градиентный бустинг**

```
# инициализация модели
GBR = GradientBoostingRegressor(loss='absolute_error', n_estimators=500,
                                random_state=RANDOM_STATE)
```

```
# обучение
GBR.fit(features, target)
```

```
GradientBoostingRegressor(loss='absolute_error', n_estimators=500,
                          random_state=12345)
```

```
features = pd.concat([class_1, class_2, class_3])
features.head()
```

region	model_class	make	model	trim	body	transmission	color	interior	gbr	rfr
-0.575553	1.233457	0.913936	1.198527	-2.009489	-0.267017	-0.341462	-1.260447	-0.717286	9069.091398	10074.425356
-0.575553	1.233457	1.140201	-0.169427	-1.440718	1.232406	-0.341462	0.810587	-0.717286	6562.227203	5325.981453
1.523077	1.233457	-0.217384	-0.585761	-0.218085	-0.267017	-0.341462	-1.260447	2.230962	-363.358893	1129.432563
-0.575553	1.233457	1.894414	1.656037	1.253106	1.232406	-0.341462	-1.112516	0.511151	7594.217353	9670.233382
-0.575553	1.233457	-1.197862	0.265207	-0.648022	-0.267017	-0.341462	1.254380	2.230962	12490.827428	11994.253676

Эксперименты с ансамблем

способ расчёта	Kaggle
ансамбль моделей	20,50
передать финальной модели только два столбца с предсказаниями бустинга и леса	21,21
снова масштабировать два столбца с предсказаниями, чтобы они были в том же диапазоне, что и остальные	20,68
предоставлять решающему лесу вместе с остальными характеристиками предсказания градиентного бустинга	21,71

Анализ важности признаков моделей

```
# упорядоченный список весов характеристик финальной модели
sorted(zip(GBR.feature_importances_, features.columns), reverse=True)
```

```
[(0.8658465529132141, 'gbr'),
 (0.08865560786866611, 'rfr'),
 (0.011518952121683074, 'model_m_price'),
 (0.008029787026256906, 'trim'),
 (0.005389823507459146, 'seller_m_price'),
```

```
# градиентный бустинг
sorted(zip(gbr_model_2_class.feature_importances_, features.columns), reverse=True)
```

```
[(0.21134382696764864, 'odometer'),
 (0.19727984929786566, 'model_m_price'),
 (0.13092869141250227, 'seller_m_price'),
 (0.1037691974126957, 'year_m_price'),
 (0.08779321802559908, 'trim'),
```

```
# случайный лес
sorted(zip(rfr_model_2_class.feature_importances_, features.columns), reverse=True)
```

```
[(0.4664876456561101, 'odometer'),
 (0.18329540481213916, 'model_m_price'),
 (0.1664315430770359, 'year_m_price'),
 (0.06209058408391615, 'seller_m_price'),
 (0.029573284766815647, 'model'),
```

Сложности

сложности	пути решения
нехватка времени	использовать все то, что изучила на данный момент
незнание отрасли	видеоролики, извлечь максимум из имеющихся в данных связях
недостаток опыта	нет ошибок, есть опыт
разные стратегии при выполнении проекта для построения пригодной для использования модели и для победы в соревновании: • использование даты продажи • возможность протестировать решение на тестовой выборке	пока принимаются предсказания на сайте сделать все для повышения финального счета, потом оформить тетрадь для ревью
нехватка вычислительных мощностей	разбила задачу на этапы и каждый выполняла в своей тетрадке
сборка финальной тетради заняла большое время	некоторые этапы работы не получилось вставить в нее, только упомянула, без демонстрации кода

Что понравилось

- интересная задача, благодаря которой
  - систематизировались и закрепились уже имеющиеся знания
  - стали видны направления дальнейшей работы по изучению машинного обучения
- знакомство с Kaggle
- общение в Телеграм-канале
- полезные вебинары
- постоянная поддержка куратора (Артем, тебе особая благодарность!)

Благодарю за внимание

Вера Мельникова

Сайт	Телеграм	Дзен
https://mathkaleidoscope.ru/	https://t.me/mathkaleidoskope	https://dzen.ru/mathkaleido