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 (ИУ)

КАФЕДРА Системы обработки информации и управления (ИУ5)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Предсказание расхода топлива для машин

Студент ИУ5-31М _____ **Цветкова А.К.**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **Гапанюк Ю.Е.**
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2021 г.

Содержание

Цель работы

Целью научно-исследовательской работы является предсказание расхода топлива для машин в зависимости от ее характеристик. Будет проведена предобработка данных и применены методы машинного обучения к предобработанным и сырым данным.

Описание данных

В наборе данных содержатся данные для 398 машин и содержатся следующие столбцы: mpg (расход топлива, миль на галлон), cylinders (количество цилиндров), displacement, horsepower (количество лошадиных сил), weight (вес), acceleration (ускорение), model year (год выпуска), origin, car name (название модели).

Пример первых пяти строк датасета:

```
data = pd.read_csv('auto-mpg.csv')
data_raw = pd.read_csv('auto-mpg.csv')
data.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

```
data.shape
```

```
(398, 9)
```

Предобработка данных

Перед применением методов машинного обучения данные нужно очистить от аномальных данных, заполнить пропуски, выполнить масштабирование.

Замена ? на пропуски и пропусков на медиану

```
data["horsepower"].replace({"?": np.nan}, inplace=True)
```

```
data.isnull().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

```
def impute_column(dataset, column, strategy_param, fill_value_param=None):
    """
    Заполнение пропусков в одном признаке
    """
    temp_data = dataset[[column]].values
    size = temp_data.shape[0]

    indicator = MissingIndicator()
    mask_missing_values_only = indicator.fit_transform(temp_data)

    imputer = SimpleImputer(strategy=strategy_param,
                            fill_value=fill_value_param)
    all_data = imputer.fit_transform(temp_data)

    missed_data = temp_data[mask_missing_values_only]
    filled_data = all_data[mask_missing_values_only]

    return all_data.reshape((size,)), filled_data, missed_data
```

```
all_data, filled_data, missed_data = impute_column(data, 'horsepower', 'median')
```

```
data['horsepower']= all_data
```

Теперь нет пропусков

```
data.isnull().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

В наборе нет категориальных признаков:

```
data.dtypes
```

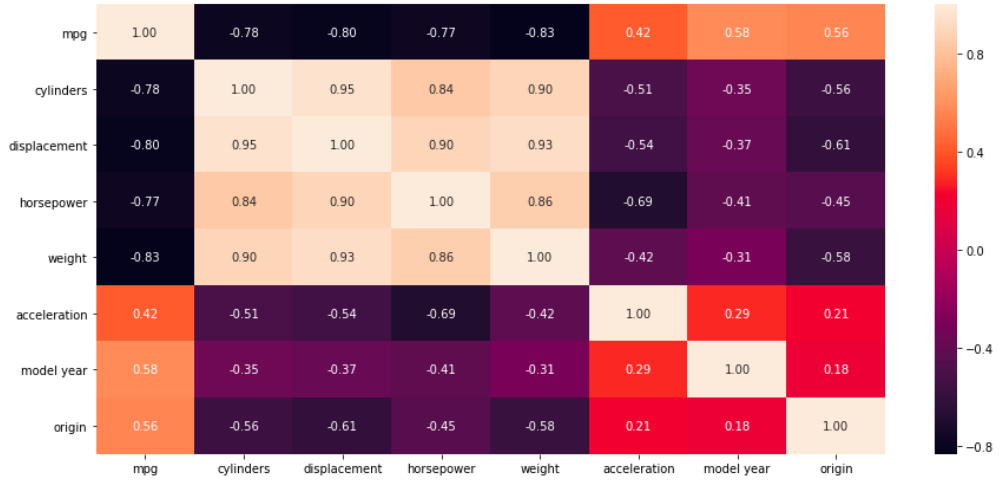
```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

На матрице корреляций можно посмотреть, какие признаки больше всего коррелируют с целевым. Так как в основном замечена отрицательная корреляция, некоторые признаки инвертируются.

mpg - целевой признак

```
fig, ax = plt.subplots(figsize=(15,7))
sns.heatmap(data.corr(method='pearson'), ax=ax, annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x130045978>
```



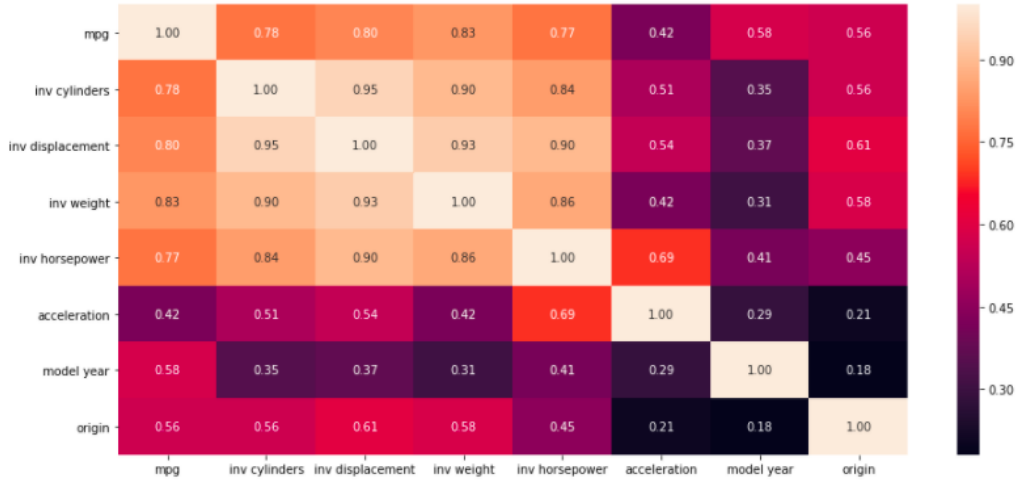
Инвертируем для получения положительной корелляции

```
data['inv cylinders'] = 0 - data['cylinders']
data['inv displacement'] = 0 - data['displacement']
data['inv weight'] = 0 - data['weight']
data['inv horsepower'] = 0 - data['horsepower']
```

```
data = data [['mpg', 'inv cylinders', 'inv displacement', 'inv weight', 'inv horsepower', 'acceleration', 'model year',
```

```
fig, ax = plt.subplots(figsize=(15,7))
sns.heatmap(data.corr(method='pearson'), ax=ax, annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x130278b38>
```



Также данные были масштабированы. Масштабирование - это изменение диапазона измерения признака с целью улучшения качества построения модели.

Масштабирование

```
from sklearn.preprocessing import MinMaxScaler
mmc = MinMaxScaler()
data[['inv cylinders']] = mmc.fit_transform(data[['inv cylinders']])
data[['inv displacement']] = mmc.fit_transform(data[['inv displacement']])
data[['inv weight']] = mmc.fit_transform(data[['inv weight']])
data[['inv horsepower']] = mmc.fit_transform(data[['inv horsepower']])
data[['acceleration']] = mmc.fit_transform(data[['acceleration']])
data[['model year']] = mmc.fit_transform(data[['model year']])
data[['origin']] = mmc.fit_transform(data[['origin']])
data[['mpg']] = mmc.fit_transform(data[['mpg']])
```

```
data.describe()
```

	mpg	inv cylinders	inv displacement	inv weight	inv horsepower	acceleration	model year	origin
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	0.386026	0.509045	0.675902	0.615133	0.683130	0.450482	0.500838	0.286432
std	0.207872	0.340201	0.269431	0.240103	0.207732	0.164148	0.308136	0.401027
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.226064	0.000000	0.498708	0.434363	0.570652	0.346726	0.250000	0.000000
50%	0.372340	0.800000	0.791990	0.662461	0.741848	0.446429	0.500000	0.000000
75%	0.531915	0.800000	0.906331	0.826836	0.836957	0.546131	0.750000	0.500000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Отбор признаков

Для выбора признаков, наиболее полезных для дальнейшего построения модели был использован алгоритм полного перебора с помощью класса ExhaustiveFeatureSelector из библиотеки MLxtend. Параметры min_features и max_features используются для задания диапазона количества перебираемых признаков. Были получены признаки 'inv displacement', 'inv weight', 'inv horsepower', 'model year', 'origin'. Можно заметить по матрице корреляций, что они хорошо коррелируют с целевым признаком.

Отбор признаков, наиболее подходящих для построения модели

```
from sklearn.neighbors import KNeighborsRegressor
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
```

```
data_x = data[['inv cylinders', 'inv displacement', 'inv weight', 'inv horsepower', 'acceleration', 'model year', 'orig
data_y = data['mpg']
```

```
knn = KNeighborsRegressor(n_neighbors=3)
```

```
%%time
efs1 = EFS(knn,
           min_features=2,
           max_features=7,
           scoring='neg_mean_squared_error',
           print_progress=True,
           cv=5)

efs1 = efs1.fit(data_x, data_y, custom_feature_names=data_x.columns)

#print('Best accuracy score: %.2f' % efs1.best_score_)
print('Best subset (indices):', efs1.best_idx_)
print('Best subset (corresponding names):', efs1.best_feature_names_)
```

```
Features: 120/120

Best subset (indices): (1, 2, 3, 5, 6)
Best subset (corresponding names): ('inv displacement', 'inv weight', 'inv horsepower', 'model year', 'origin')
CPU times: user 1.47 s, sys: 49.4 ms, total: 1.52 s
Wall time: 1.59 s
```

После отбора признаков разделяем выборку на тестовую и обучающую.

Разделение выборки ¶

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_
```

```
data_X_train, data_X_test, data_y_train, data_y_test = train_test_split(
    data_x, data_y, test_size=0.3, random_state=1)
data_X_train.shape, data_X_test.shape
```

```
((278, 7), (120, 7))
```

Метод ближайших соседей

Метод ближайших соседей — простейший метрический классификатор, основанный на оценивании сходства объектов. Классифицируемый объект относится к тому классу, которому принадлежат ближайшие к нему объекты обучающей выборки.

Подбор параметров – одна из важных задач для построения модели машинного обучения. Изменение параметров модели может принципиально повлиять на ее качество. Перебор этих параметров вручную может занять колоссальное количество времени. Однако, существует модуль GridSearchCV. GridSearchCV – это очень мощный инструмент для автоматического подбирания параметров для моделей машинного обучения. GridSearchCV находит наилучшие параметры, путем обычного перебора: он создает модель для каждой возможной комбинации параметров.

Подбор гиперпараметров и KNN

```
n_range = np.array(range(2,55,1))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
        19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
        36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
        53, 54])}]
```

```
%%time
clf_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=LeaveOneOut(), scoring='neg_mean_squared_error')
clf_gs.fit(data_X_train, data_y_train)
```

```
CPU times: user 1min 12s, sys: 711 ms, total: 1min 12s
Wall time: 1min 23s

GridSearchCV(cv=LeaveOneOut(), error_score='raise-deprecating',
             estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30,
                                           metric='minkowski',
                                           metric_params=None, n_jobs=None,
                                           n_neighbors=5, p=2,
                                           weights='uniform'),
             iid='warn', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
       53, 54])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
clf_gs.best_params_
```

```
{'n_neighbors': 3}
```

```
KNN = KNeighborsRegressor(n_neighbors=3)
KNN.fit(data_X_train, data_y_train)
target_KNN = KNN.predict(data_X_test)
```

```
mean_absolute_error(data_y_test, target_KNN)
```

```
0.057143912529550825
```

```
median_absolute_error(data_y_test, target_KNN)
```

```
0.03989361702127657
```

KNN на сырых данных

```
data_raw["horsepower"].replace({"?": np.nan}, inplace=True)
data_raw[['mpg']] = mmc.fit_transform(data_raw[['mpg']])
```

```
data_raw = data_raw.dropna(axis=0, how='any')
data_raw.shape
```

```
(392, 9)
```

```
data_raw_x = data_raw[['cylinders', 'displacement', 'weight', 'horsepower', 'acceleration', 'model year', 'origin']]
data_raw_y = data_raw['mpg']
```

```
data_raw_X_train, data_raw_X_test, data_raw_y_train, data_raw_y_test = train_test_split(
    data_raw_x, data_raw_y, test_size=0.3, random_state=1)
data_raw_X_train.shape, data_raw_X_test.shape
```

```
((274, 7), (118, 7))
```

```
KNN = KNeighborsRegressor(n_neighbors=3)
KNN.fit(data_raw_X_train, data_raw_y_train)
target_KNN = KNN.predict(data_raw_X_test)
```

```
mean_absolute_error(data_raw_y_test, target_KNN)
```

```
0.0850387666786873
```

```
median_absolute_error(data_raw_y_test, target_KNN)
```

```
0.0642730496453901
```

Autogluon

Автоматическое машинное обучение (AutoML) — процесс автоматизации сквозного процесса применения машинного обучения к задачам реального мира. В типичном приложении машинного обучения пользователь должен применить подходящие методы предварительной обработки данных, конструирования признаков, выделения признаков и выбора признаков, которые делают набор данных пригодным для обучения машин. После этих шагов работник должен осуществить выбор алгоритма и оптимизацию гиперпараметров для максимизации прогнозируемой производительности конечной модели. Поскольку многие из этих шагов не могут осуществить люди, не будучи экспертами, был предложен подход AutoML как основанное на искусственном интеллекте решение для всё возрастающей необходимости применения машинного обучения. Автоматизация сквозного процесса применения машинного обучения даёт преимущество получения более простых решений, более быстрого создания таких решений и моделей, которые часто превосходят модели, построенные вручную.

AutoGluon - это платформа AutoML с открытым исходным кодом, созданная AWS, которая позволяет легко использовать и легко расширять AutoML.

Autogluon

```
from autogluon.tabular import TabularPredictor
```

```
data_train, data_test = train_test_split(
    data, test_size=0.3, random_state=1)
data_train.shape, data_test.shape
```

```
((278, 8), (120, 8))
```

```
predictor = TabularPredictor(label='mpg').fit(train_data=data_train)
```

```
predictions = predictor.predict(data_test)
```

```
perf = predictor.evaluate_predictions(y_true=data_test['mpg'], y_pred=predictions, auxiliary_metrics=True)
```

```
Evaluation: root_mean_squared_error on test data: -0.0735609844055441
        Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "root_mean_squared_error": -0.0735609844055441,
    "mean_squared_error": -0.005411218426712701,
    "mean_absolute_error": -0.055652427647579855,
    "r2": 0.8691540220016181,
    "pearsonr": 0.938895424672481,
    "median_absolute_error": -0.04143354835662434
}
```

Autogluon на сырых данных

```
data_raw_train, data_raw_test = train_test_split(
    data_raw, test_size=0.3, random_state=1)
```

```
predictor = TabularPredictor(label='mpg').fit(train_data=data_raw_train)
```

```
predictions = predictor.predict(data_raw_test)
```

```
perf = predictor.evaluate_predictions(y_true=data_raw_test['mpg'], y_pred=predictions, auxiliary_metrics=True)
```

```
Evaluation: root_mean_squared_error on test data: -0.08133866038767745
        Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "root_mean_squared_error": -0.08133866038767745,
    "mean_squared_error": -0.0066159776736619295,
    "mean_absolute_error": -0.058148219517818046,
    "r2": 0.859312359334793,
    "pearsonr": 0.9303026577827065,
    "median_absolute_error": -0.0427735460565446
}
```

Вывод

После предобработки данных средняя и медианная ошибка составили 0.057143912529550825 и 0.03989361702127657 для KNN и 0.055652427647579855 и 0.04143354835662434 для Autogluon. Без предобработки данных средняя и медианная ошибка увеличивается: 0.0850387666786873 и 0.0642730496453901 для KNN и 0.08133866038767745 и 0.0427735460565446 для Autogluon. Можно сделать вывод о важности предобработки данных для решения задач машинного обучения.