Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Курсовая работа по дисциплине «Методы машинного обучения» на тему: «Классификация книжных отзывов»

ИСПОЛНИТЕЛЬ:

Фадеев Артем Александрович, ИУ5-31М

_____

"__"_____2021 г.

Москва 2021

Оглавление

Оглавление....2
Задание....3
Подготовка данных....4
- Загрузка датасета книг....4
- Устранение пропусков данных....4
- Обработка нестандартного признака....4
- Обработка категориальных признаков....5
- Нормализация числовых признаков....6
- Масштабирование признаков....6
- Отбор признаков....7

Результат работы моделей....9
AutoML....10

Задание

1. Поиск и выбор набора данных для построения модели машинного обучения. На основе выбранного набора данных строится модель для задачи классификации.
2. Для выбранного датасета решить следующие задачи:
    a. устранение пропусков в данных;
    b. кодирование категориальных признаков;
    c. нормализацию числовых признаков;
    d. масштабирование признаков;
    e. обработку выбросов для числовых признаков;
    f. обработку нестандартных признаков (которые не является числовым или категориальным);
    g. отбор признаков, наиболее подходящих для построения модели;
3. Обучить модель и оценить метрики качества для двух выборок:
    a. исходная выборка, которая содержит только минимальную предобработку данных, необходимую для построения модели (например, кодирование категориальных признаков).
    b. улучшенная выборка, полученная в результате полной предобработки данных в пункте 2.
4. Построить модель с использованием произвольной библиотеки AutoML.
5. Сравнить метрики для трех полученных моделей.

Подготовка данных

Загрузка датасета книг

```
In [2]: data = pd.read_csv('./books.csv', sep=',', encoding="utf-8")
        data.head(1)
```

Out[2]:

	Title	Authors	Average_rating	ISBN	ISBN13	Language_code	Num_pages	Ratings_count	Text_reviews_count	Publication_date	Publisher
0	Harry Potter and the Half-Blood Prince (Harry ...	J.K. Rowling/Mary GrandPré	4.57	0439785960	9780439785969	eng	652	2095690	27591	9/16/2006	Scholastic Inc.

Устранение пропусков данных

Датасет без пустых значений признаков, поэтому данный этап его подготовки я пропустил. Хотя теоретически можно было бы заменить пропуски, например, медианными значениями.

```
In [13]: data.isnull().sum()

Out[13]: BookID              0
         Title               0
         Authors             0
         Average_rating      0
         ISBN                0
         ISBN13              0
         Language_code       0
         Num_pages           0
         Ratings_count       0
         Text_reviews_count  0
         Publication_date    0
         Publisher           0
         dtype: int64
```

Обработка нестандартного признака

В данном блоке я переопределил столбец BookID, а также разделил дату публикации на месяц и год публикации, сделав из необычного признака два числовых. Из «месяцев» теперь легко выделить классы.

In [9]:
```
data = data.reset_index(drop=True)
enc = OrdinalEncoder(categories='auto', dtype=int)
data[["BookID"]] = enc.fit_transform(data[["BookID"]])
data["Authors"] = data["Authors"].str.split('/',expand=True)[[0]]
data.rename(columns={"Authors":"Author"}, inplace=True)
date_data = data["Publication_date"].str.split('/',expand=True)
date_data.columns = ["Publication_month", "Publication_day", "Publication_year"]
date_data = date_data[["Publication_month", "Publication_year"]]
date_data.head(1)
data["Publication_month"] = pd.to_numeric(date_data["Publication_month"])
data["Publication_year"] = pd.to_numeric(date_data["Publication_year"])
data = data.drop(columns="ISBN")
data = data.drop(columns="ISBN13")
data = data.drop(columns="Publication_date")
data.head(1)
```

Out[9]:

	BookID	Title	Author	Average_rating	Language_code	Num_pages	Ratings_count	Text_reviews_count	Publisher	Publication_month	Publication_year
0	0	Harry Potter and the Half-Blood Prince (Harry ...	J.K. Rowling	4.57	eng	652	2095690	27591	Scholastic Inc.	9	2006

Обработка категориальных признаков

Для данного этапа я не пользовался стандартными библиотеками, а создал собственный способ, который можно считать кодированием по частоте. Этот принцип, и его реализация в данной работе, предполагает замену какого-либо строкового значения количеством вхождений такого значения в данный признак (см. пару Publisher - Publisher_Freq).

In [15]:
```
agg_data = data.groupby("Title").BookID.count().sort_values(ascending=False).reset_index()
agg_data.columns = ["title", "Title_Freq"]
data = pd.merge(data, agg_data, left_on="Title", right_on="title").drop(columns="title")
agg_data = data.groupby("Author").BookID.count().sort_values(ascending=False).reset_index()
agg_data.columns = ["author", "Author_Freq"]
data = pd.merge(data, agg_data, left_on="Author", right_on="author").drop(columns="author")
agg_data = data.groupby("Language_code").BookID.count().sort_values(ascending=False).reset_index()
agg_data.columns = ["language_code", "Language_code_Freq"]
data = pd.merge(data, agg_data, left_on="Language_code", right_on="language_code").drop(columns="language_code")
agg_data = data.groupby("Publisher").BookID.count().sort_values(ascending=False).reset_index()
agg_data.columns = ["publisher", "Publisher_Freq"]
data = pd.merge(data, agg_data, left_on="Publisher", right_on="publisher").drop(columns="publisher")
data = data.sort_values(by=["BookID"]).reset_index().drop(columns="index")
data["Publication_monthName"] = pd.to_datetime(data["Publication_month"], format='%m').dt.month_name().str.slice(stop=3)
data.head(10)
```

Out[15]:

:ount	Text_reviews_count	Publisher	Publication_month	Publication_year	Title_Freq	Author_Freq	Language_code_Freq	Publisher_Freq	Publication_monthName
5690	27591	Scholastic Inc.	9	2006	2	24	8908	13	Sep
3167	29221	Scholastic Inc.	9	2004	1	24	8908	13	Sep
6333	244	Scholastic	11	2003	2	24	8908	33	Nov
9585	36325	Scholastic Inc.	5	2004	2	24	8908	13	May
1428	164	Scholastic	9	2004	1	24	8908	33	Sep

Нормализация числовых признаков

Для этого использую стандартный метод, предварительно убрав лишние, уже обработанные признаки.

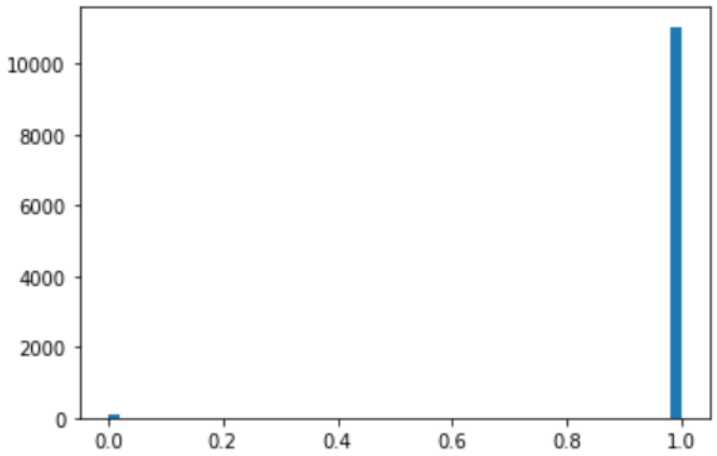
```
In [19]: clear_data = data[['Average_rating','Num_pages','Ratings_count','Text_reviews_count','Publication_month','Publication_year','Tit
         clear_data.head(1)
```

Out[19]:

erage_rating	Num_pages	Ratings_count	Text_reviews_count	Publication_month	Publication_year	Title_Freq	Author_Freq	Language_code_Freq	Publisher_Freq
4.57	652	2095690	27591	9	2006	2	24	8908	13

```
In [23]: norm = Normalizer()
```

```
In [24]: ex_data = norm.fit_transform(clear_data[['Ratings_count']])
         plt.hist(ex_data, 50)
         plt.show()
```



Аналогичную операцию произведу со всеми признаками.

```
In [33]: n_data = pd.DataFrame(norm.fit_transform(clear_data), columns=['Average_rating','Num_pages','Ratings_count','Text_reviews_count'
         n_data.head(1)
```

Out[33]:

rage_rating	Num_pages	Ratings_count	Text_reviews_count	Publication_month	Publication_year	Title_Freq	Author_Freq	Language_code_Freq	Publisher_Freq
0.000002	0.000311	0.999904	0.013164	0.000004	0.000957	9.542478e-07	0.000011	0.00425	0.000006

Масштабирование признаков

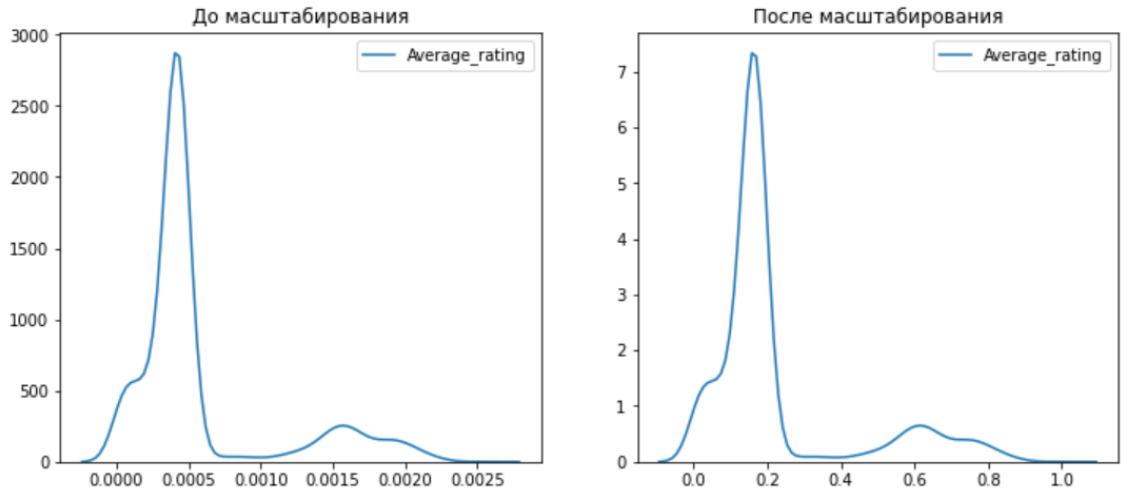
Для этого использую стандартный метод, предварительно убрав лишние, уже обработанные признаки.

In [44]:
```
mmscale = MinMaxScaler()
mm_data = pd.DataFrame(mmscale.fit_transform(n_data), columns=cols)
mm_data.describe()
```

Out[44]:

	Average_rating	Num_pages	Ratings_count	Text_reviews_count	Publication_month	Publication_year	Title_Freq	Author_Freq
count	11123.000000	11123.000000	11123.000000	11123.000000	11123.000000	11123.000000	11123.000000	11123.000000
mean	0.226416	0.082361	0.302589	0.089470	0.160205	0.294131	0.049446	0.034233
std	0.200126	0.101209	0.356812	0.117652	0.176934	0.257334	0.060816	0.070048
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.143078	0.027479	0.016831	0.007831	0.053947	0.196206	0.027407	0.003015
50%	0.165149	0.051235	0.108492	0.039127	0.109857	0.217774	0.029954	0.011446
75%	0.181455	0.089558	0.569349	0.129710	0.183031	0.219181	0.040886	0.032955
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [53]:
```
draw_kde('Average_rating', n_data, mm_data, 'До масштабирования', 'После масштабирования')
```



Отбор признаков

Для этого использую метод фильтрации, основанный на удалении константных или почти константных признаков.

По итогу можно избавиться от двух колонок: Заглавие книги. Средний рейтинг является будущим целевым признаком, поэтому его не удаляю.

In [78]:
```
from sklearn.feature_selection import VarianceThreshold

selector = VarianceThreshold(threshold=0.15)
selector.fit(clear_data)
# Значения дисперсий для каждого признака - великолепные признаки
for i in range(len(selector.variances_)):
    print(selector.variances_[i].round(2), '\t', clear_data.columns[i])
```

```
0.12     Average_rating
58149.36         Num_pages
12654921703.78   Ratings_count
6638371.64       Text_reviews_count
11.65    Publication_month
68.01    Publication_year
0.77     Title_Freq
189.39   Author_Freq
10166643.46      Language_code_Freq
4906.97          Publisher_Freq
```

In [80]:
```
f_data = mm_data.drop(['Title_Freq'], axis=1)
f_data.head(1)
```

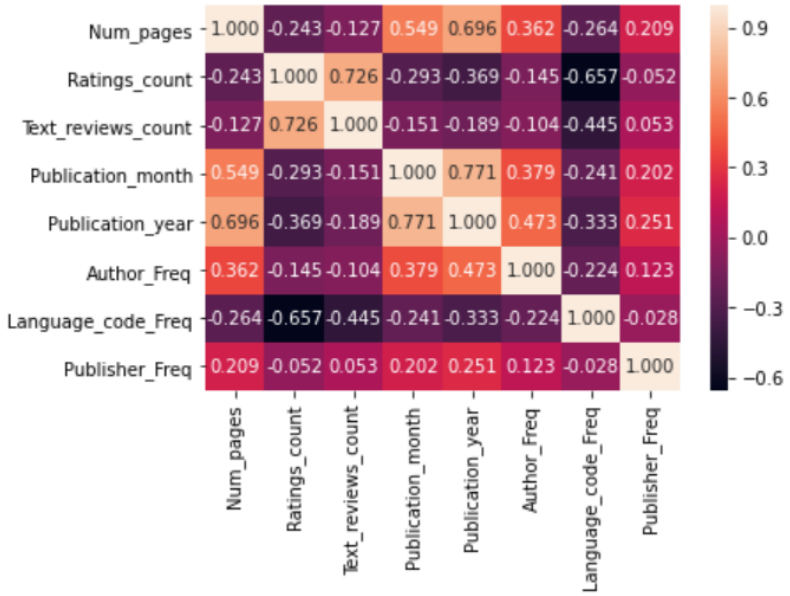
Out[80]:

	Average_rating	Num_pages	Ratings_count	Text_reviews_count	Publication_month	Publication_year	Author_Freq	Language_code_Freq	Publisher_Freq
0	0.000854	0.00053	0.999934	0.078507	0.00065	0.000521	0.000262	0.004127	0.000037

Также для фильтрации можно применить отсев признаков, состоящих в какой-либо группе коррелирующих признаков (например, у которых корреляция между друг другом > 0,9). Однако, в данном наборе данных таких признаков нет, поэтому не можем применять этот метод.

In [82]:
```
sns.heatmap(X_train_df.corr(), annot=True, fmt='.3f')
```

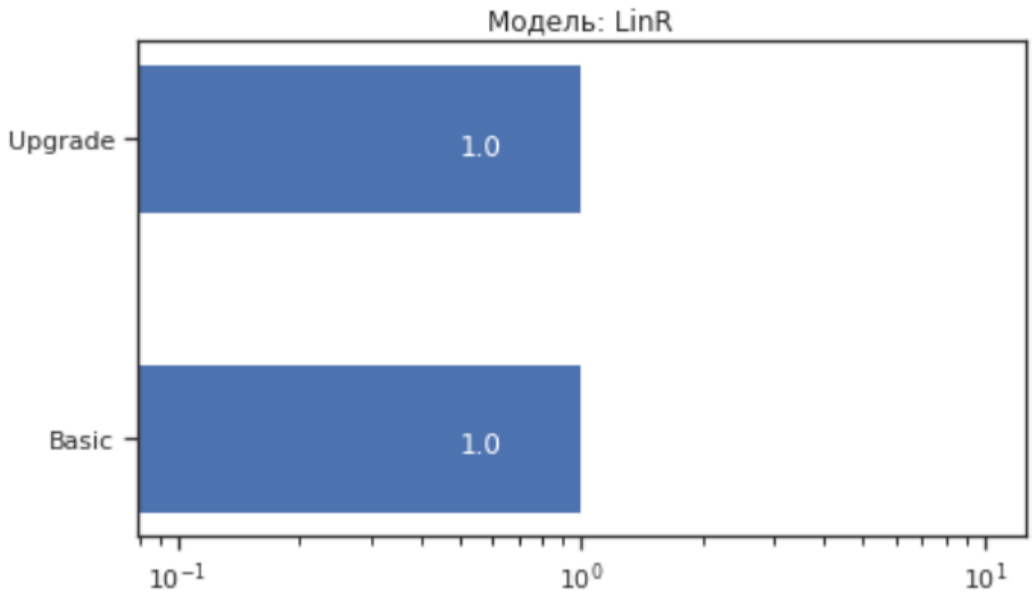
Out[82]: <AxesSubplot:>



Результат работы моделей

In [125]:
```
clas_models_dict = {'LinR': LogisticRegression(),
                    'KNN_5':KNeighborsClassifier(n_neighbors=5),
                    'Tree':DecisionTreeClassifier(random_state=1),
                    'GB': GradientBoostingClassifier(random_state=1),
                    'RF':RandomForestClassifier(n_estimators=20, random_state=1)}
```

In [129]:
```
for model in clas_models_dict:
    logger.plot('Модель: ' + model, model, figsize=(7, 4))
```



Для KNN_5 - 0.94. Для остальных - 1.

AutoML

```
In [19]: train = f_data.copy()
```

```
In [21]: from supervised.automl import AutoML
         automl = AutoML()
```

```
In [23]: automl.fit(train[train.columns[:-1]], train['Average_rating'])
```

```
Linear algorithm was disabled.
AutoML directory: AutoML_1
The task is regression with evaluation metric rmse
AutoML will use algorithms: ['Baseline', 'Decision Tree', 'Random Forest', 'Xgboost', 'Neural Network']
AutoML will ensemble available models
AutoML steps: ['simple_algorithms', 'default_algorithms', 'ensemble']
* Step simple_algorithms will try to check up to 2 models
1_Baseline rmse 0.200038 trained in 0.24 seconds
2_DecisionTree rmse 0.020906 trained in 13.62 seconds
* Step default_algorithms will try to check up to 3 models
3_Default_Xgboost rmse 0.002411 trained in 4.68 seconds
4_Default_NeuralNetwork rmse 0.001611 trained in 1.31 seconds
5_Default_RandomForest rmse 0.010195 trained in 8.06 seconds
* Step ensemble will try to check up to 1 model
Ensemble rmse 0.001359 trained in 0.29 seconds
AutoML fit time: 36.6 seconds
AutoML best model: Ensemble
```

Для полностью подготовленного датасета без лишнего признака «Заглавие книги» точность Ансамблевой модели (обучилась за 0.29 секунд) составляет 99,8641% при обучении самой AutoML в 36.6 секунд - неплохо!

Если этот признак не удалять, то обучение Ансамблевой модели будет происходить быстрее (0.21 секунды), точность - 99,8228% (разница более 0,04 процента), а обучение самой AutoML в 27.98 секунд - тоже неплохо.

```
In [24]: automl2 = AutoML()
```

```
In [26]: automl2.fit(mm_data, train['Average_rating'])
```

```
Linear algorithm was disabled.
AutoML directory: AutoML_2
The task is regression with evaluation metric rmse
AutoML will use algorithms: ['Baseline', 'Decision Tree', 'Random Forest', 'Xgboost', 'Neural Network']
AutoML will ensemble available models
AutoML steps: ['simple_algorithms', 'default_algorithms', 'ensemble']
* Step simple_algorithms will try to check up to 2 models
1_Baseline rmse 0.200038 trained in 0.42 seconds
2_DecisionTree rmse 0.020906 trained in 3.86 seconds
* Step default_algorithms will try to check up to 3 models
3_Default_Xgboost rmse 0.002333 trained in 5.15 seconds
4_Default_NeuralNetwork rmse 0.002586 trained in 1.11 seconds
5_Default_RandomForest rmse 0.010185 trained in 6.91 seconds
* Step ensemble will try to check up to 1 model
Ensemble rmse 0.001772 trained in 0.21 seconds
AutoML fit time: 27.98 seconds
AutoML best model: Ensemble
```