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»

КАФЕДРА «Системы обработки информации и управления»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Построение моделей машинного обучения

Студент ____ИУ5-62Б___ _____ Еремихин В.С.
(Группа) (Подпись, дата) (И.О.Фамилия)

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

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

2023 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой _____
(Индекс)
_____ _____
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по дисциплине _____Технологии машинного обучения_____

Студент группы ____ИУ5-62Б_____

_____Еремихин Владислав Станиславович_____
(Фамилия, имя, отчество)

Тема курсового проекта __“Построение моделей машинного обучения”_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____
Источник тематики (кафедра, предприятие, НИР) _____

График выполнения проекта: 25% к _ _ нед., 50% к __ нед., 75% к __ нед., 100% к ___ нед.
Задание решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на _29__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания «12» февраля 2023 г.

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

Студент _____ Еремихин В.С.
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

СОДЕРЖАНИЕ

Введение

В данной научно-исследовательской работе предстоит выполнить типовую задачу машинного обучения - провести анализ данных, провести некоторые операции с датасетом, подобрать модели, а также подобрать наиболее подходящие гиперпараметры выбранных моделей. Машинное обучение очень актуально в современном мире,оно используется практически во многих сферах. Программист должен подбирать подходящие технологии машинного обучения для достижения наилучших результатов. Чему мы и научимся в этом курсовом проекте. Попробуем не менее пяти видов различных моделей и подберем наилучшую из них на основе выбранных метрик. Также построим вспомогательные графики, которые помогут нам визуально взглянуть на все необходимые показатели

Задание

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

1. Выбор и подготовка набора данных

Ввод [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score, confusion_matrix, plot_confusion_matrix
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from scipy.stats import randint, uniform
from sklearn.ensemble import GradientBoostingClassifier

# скроем предупреждения о возможных ошибках для лучшей читаемости
import warnings
warnings.filterwarnings('ignore')
```

В качестве набора данных будем использовать набор данных, состоящий из песен с музыкального сервиса Spotify (https://www.kaggle.com/zaheenhamidani/ultimate-spotify-tracks-db)

Ввод [2]:
```
train_data = pd.read_csv('./SpotifyFeatures.csv')
train_data.head()
```

Out[2]:

	genre	artist_name	track_name	track_id	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loud
0	Movie	Henri Salvador	C'est beau de faire un Show	0BRjO6ga9RKCKjfDqeFgWV	0	0.611	0.389	99373	0.910	0.000	C#	0.3460	-
1	Movie	Martin & les fées	Perdu d'avance (par Gad Elmaleh)	0BjC1NfoEOOusryehmNudP	1	0.246	0.590	137373	0.737	0.000	F#	0.1510	-
2	Movie	Joseph Williams	Don't Let Me Be Lonely Tonight	0CoSDzoNIKCRs124s9uTVy	3	0.952	0.663	170267	0.131	0.000	C	0.1030	-1
3	Movie	Henri Salvador	Dis-moi Monsieur Gordon Cooper	0Gc6TVm52BwZD07Ki6tIvf	0	0.703	0.240	152427	0.326	0.000	C#	0.0985	-1
4	Movie	Fabien Nataf	Ouverture	0IusIXpMROHdEPvSl1fTQK	4	0.950	0.331	82625	0.225	0.123	F	0.2020	-2

Размер набора:

Ввод [3]:
```
train_data.shape
```

Out[3]:
```
(232725, 18)
```

Удалим лишние столбцы:

Ввод [4]:
```
train_data = train_data.filter(['genre', 'artist_name', 'track_name', 'energy', 'loudness', 'speechiness', 'liveness',
train_data.head()
```

Out[4]:

	genre	artist_name	track_name	energy	loudness	speechiness	liveness	popularity	danceability	duration_ms	instrumentalness	acousticness
0	Movie	Henri Salvador	C'est beau de faire un Show	0.910	-1.828	0.0525	0.3460	0	0.389	99373	0.000	0.611
1	Movie	Martin & les fées	Perdu d'avance (par Gad Elmaleh)	0.737	-5.559	0.0868	0.1510	1	0.590	137373	0.000	0.246
2	Movie	Joseph Williams	Don't Let Me Be Lonely Tonight	0.131	-13.879	0.0362	0.1030	3	0.663	170267	0.000	0.952
3	Movie	Henri Salvador	Dis-moi Monsieur Gordon Cooper	0.326	-12.178	0.0395	0.0985	0	0.240	152427	0.000	0.703
4	Movie	Fabien Nataf	Ouverture	0.225	-21.150	0.0456	0.2020	4	0.331	82625	0.123	0.950

Итоговый набор содержит следующие колонки:

- genre - жанр песни
- artist_name - исполнитель песни
- track_name - название песни
- energy - энергичность
- loudness - громкость
- speechiness - показатель количества слов в песне
- liveness - показатель того, что песня была записана при аудитории
- popularity - показатель популярности песни по 10-балльной шкале
- danceability - показатель стабильности песни для танца
- duration_ms - длительность песни (в мс)
- instrumentalness - показатель вокала в песне
- acousticness - акустичность (1.0 - песня в акустической версии)

Новое количество колонок:

Ввод [5]:
```
train_data.shape[1]
```

Out[5]:
```
12
```

Переименуем названия столбцов:

Ввод [6]:
```
train_data.rename(columns={'genre': 'Genre', 'artist_name': 'Artist', 'acousticness': 'Acousticness', 'instrumentalness
train_data.head()
```

Out[6]:

	Genre	Artist	Track	Energy	Loudness	Speechiness	Liveness	Popularity	Danceability	Duration	Instrumentalness	Acousticness
0	Movie	Henri Salvador	C'est beau de faire un Show	0.910	-1.828	0.0525	0.3460	0	0.389	99373	0.000	0.611
1	Movie	Martin & les fées	Perdu d'avance (par Gad Elmaleh)	0.737	-5.559	0.0868	0.1510	1	0.590	137373	0.000	0.246
2	Movie	Joseph Williams	Don't Let Me Be Lonely Tonight	0.131	-13.879	0.0362	0.1030	3	0.663	170267	0.000	0.952
3	Movie	Henri Salvador	Dis-moi Monsieur Gordon Cooper	0.326	-12.178	0.0395	0.0985	0	0.240	152427	0.000	0.703
4	Movie	Fabien Nataf	Ouverture	0.225	-21.150	0.0456	0.2020	4	0.331	82625	0.123	0.950

2. Разведочный анализ

Проверим пропуски:

Ввод [7]:
```
train_data.isnull().sum()
```

Out[7]:
```
Genre              0
Artist             0
Track              0
Energy             0
Loudness           0
Speechiness        0
Liveness           0
Popularity         0
Danceability       0
Duration           0
Instrumentalness   0
Acousticness       0
dtype: int64
```

Как видим, пропуски отсутствуют

Количество уникальных музыкальных жанров:

Ввод [8]:
```
train_data['Genre'].nunique()
```

Out[8]:
```
27
```

Количество песен каждого жанра:

Ввод [9]:
```
popular_genre=train_data.groupby('Genre').size().unique
popular_genre
```

Out[9]:
```
<bound method Series.unique of Genre
A Capella            119
Alternative         9263
Anime               8936
Blues               9023
Children's Music    5403
Children's Music    9353
Classical           9256
Comedy              9681
Country             8664
Dance               8701
Electronic          9377
Folk                9299
Hip-Hop             9295
Indie               9543
Jazz                9441
Movie               7806
Opera               8280
Pop                 9386
R&B                 8992
Rap                 9232
Reggae              8771
Reggaeton           8927
Rock                9272
Ska                 8874
Soul                9089
Soundtrack          9646
World               9096
dtype: int64>
```

Для решения задачи классификации выберем два жанра - поп (Pop) и рок (Rock):

Ввод [10]:
```
top_genres = ['Pop', 'Rock']
```

Ввод [11]:
```
train_data = train_data[train_data['Genre'].isin(top_genres)]
train_data['Genre'].unique()
```

Out[11]:
```
array(['Pop', 'Rock'], dtype=object)
```

Проверим размер набора:

Ввод [12]:
```
train_data.shape
```

Out[12]:
```
(18658, 12)
```

Подсчитаем количество исполнителей:

Ввод [13]:
```
train_data['Artist'].nunique()
```

Out[13]:
```
3297
```

Выведем топ-5 исполнителей каждого жанра:

Ввод [14]:
```
for g in top_genres:
    print(g + ':')
    print(train_data[train_data['Genre'] == g]['Artist'].value_counts().head(5))
    print('\n')
```

```
Pop:
Drake           154
BTS              76
Kanye West       72
Taylor Swift     67
Future           66
Name: Artist, dtype: int64


Rock:
The Beatles            145
Queen                   97
Led Zeppelin            76
Panic! At The Disco     74
Imagine Dragons         71
Name: Artist, dtype: int64
```

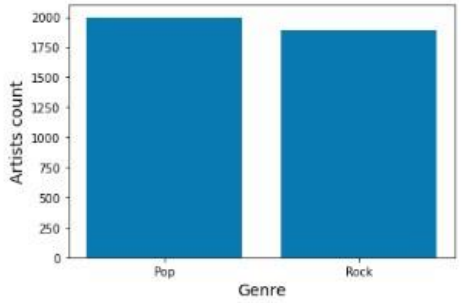
Для понимании структуры набора данных построим графики:

Ввод [15]:
```
sns.pairplot(train_data, hue="Genre")
```

Out[15]:
```
<seaborn.axisgrid.PairGrid at 0x7f7bf1fbdb20>
```

Количество уникальных артистов каждого жанра:

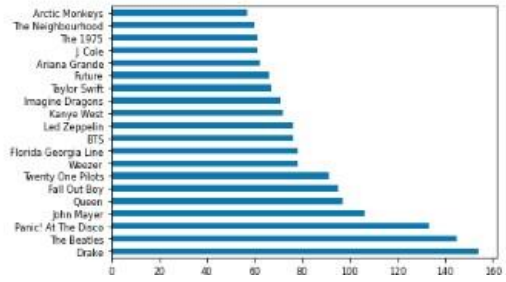
Ввод [16]:
```
x_genres = np.arange(len(top_genres))
y_artists = train_data.groupby('Genre')['Artist'].unique().agg(len)
plt.bar(x_genres, y_artists)
plt.xticks(x_genres, top_genres)
plt.xlabel('Genre', fontsize=14)
plt.ylabel('Artists count', fontsize=14)
plt.show()
```



Топ-20 исполнителей по количеству песен:

Ввод [17]:
```
train_data['Artist'].value_counts().head(20).plot(kind='barh', fontsize=8)
```

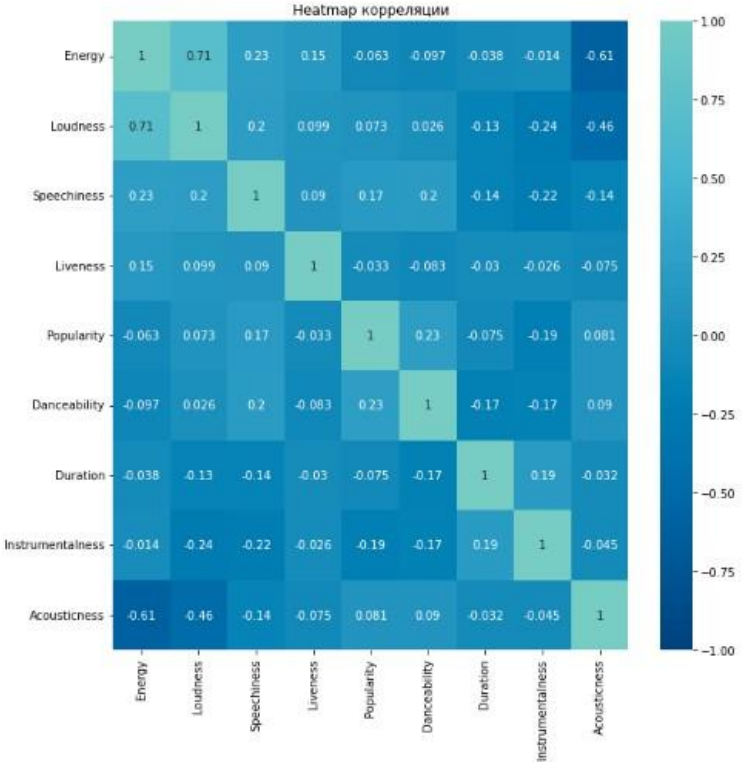
Out[17]: <AxesSubplot:>



3. Корреляционный анализ данных

Ввод [18]:
```
correlation=train_data.corr(method='spearman')
plt.figure(figsize=(10,10))
plt.title('Heatmap корреляции')
sns.heatmap(correlation,annot=True,vmin=-1,vmax=1,cmap="GnBu_r",center=1)
```

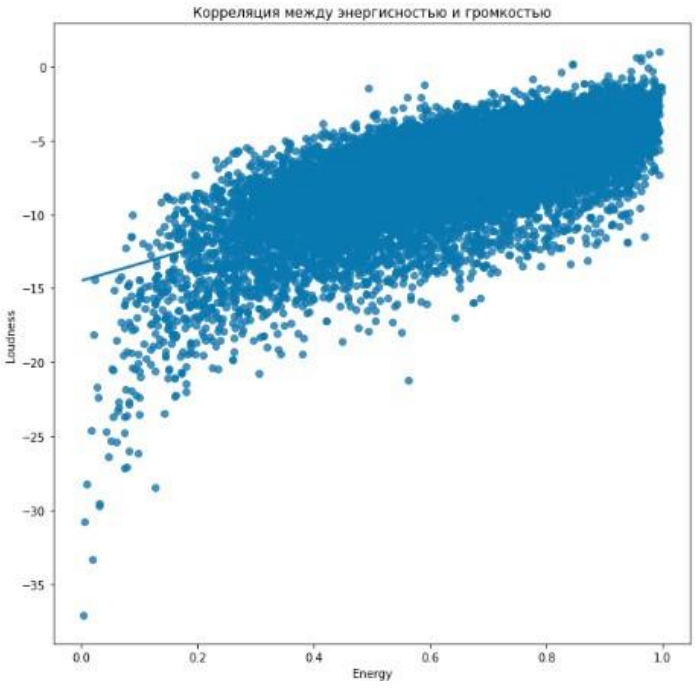
Out[18]: <AxesSubplot:title={'center':'Heatmap корреляции'}>



Проверим связь между громкостью и энергичностью:

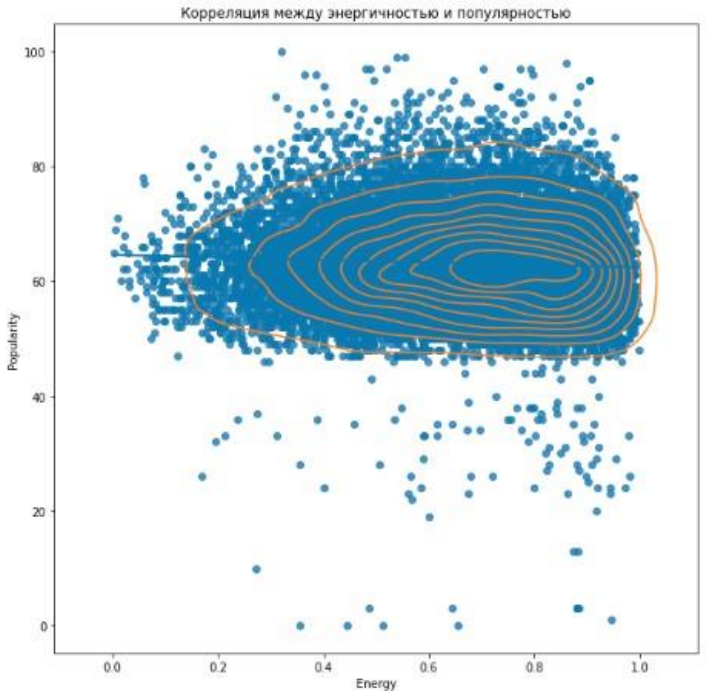
Ввод [19]:
```
fig=plt.subplots(figsize=(10,10))
plt.title('Корреляция между энергисностью и громкостью')
sns.regplot(x='Energy',y='Loudness',data=train_data)
```

Out[19]: <AxesSubplot:title={'center':'Корреляция между энергисностью и громкостью'}, xlabel='Energy', ylabel='Loudness'>



Ввод [20]:
```
fig=plt.subplots(figsize=(10,10))
plt.title('Корреляция между энергичностью и популярностью')
sns.regplot(x='Energy', y='Popularity',
            ci=None, data=train_data)
sns.kdeplot(train_data['Energy'], train_data['Popularity'])
```

Out[20]:
```
<AxesSubplot:title={'center':'Корреляция между энергичностью и популярностью'}, xlabel='Energy', ylabel='Popularity'>
```



4. Метрики для оценки качества моделей

Ввод [21]:
```
def draw_roc_curve(y_true, y_score, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

Ввод [22]:
```
class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace = True)
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

Ввод [23]:
```
metricLogger = MetricLogger()
```

Ввод [24]:
```
def test_model(model_name, model, metricLogger):
    model.fit(X_train, Y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(Y_test, y_pred)
    roc_auc = roc_auc_score(Y_test, y_pred)
    precision = precision_score(Y_test, y_pred)
    recall = recall_score(Y_test, y_pred)

    print('*' * 80)
    print(model)
    print('*' * 80)

    print('precision:', precision)
    print('recall:', recall)
    print('accuracy:', accuracy)
    print('roc_auc:', roc_auc)

    print('*' * 80)

    metricLogger.add('precision', model_name, precision)
    metricLogger.add('recall', model_name, recall)
    metricLogger.add('accuracy', model_name, accuracy)
    metricLogger.add('roc_auc', model_name, roc_auc)

    draw_roc_curve(Y_test, y_pred)

    plot_confusion_matrix(model, X_test, Y_test,
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    plt.show()
```

5. Формирование обучающей и тестовой выборок

Ввод [25]:
```
features = ['Genre', 'Acousticness', 'Instrumentalness', 'Energy', 'Loudness', 'Speechiness', 'Liveness', 'Danceability
```

Ввод [26]:
```
train_data_enc = train_data.filter(features)
train_data_enc.head()
```

Out[26]:

	Genre	Acousticness	Instrumentalness	Energy	Loudness	Speechiness	Liveness	Danceability	Duration	Popularity
107802	Pop	0.0421	0.000000	0.554	-5.290	0.0917	0.1060	0.726	190440	99
107803	Pop	0.1630	0.000002	0.539	-7.399	0.1780	0.1010	0.833	149520	99
107804	Pop	0.5780	0.000000	0.321	-10.744	0.3230	0.0884	0.725	178640	100
107805	Pop	0.1490	0.000000	0.364	-11.713	0.2760	0.2710	0.837	213594	96
107806	Pop	0.5560	0.000000	0.479	-5.574	0.0466	0.0703	0.760	158040	97

Выполним кодирование признака жанра:

Ввод [27]:
```
le = LabelEncoder()
train_data_enc['Genre'] = le.fit_transform(train_data['Genre']);
```

Разделим выборки:

Ввод [28]:
```
X = train_data_enc.drop('Genre', axis=1)
Y = train_data_enc['Genre']
```

Ввод [29]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=1)
print('{}, {}'.format(X_train.shape, X_test.shape))
print('{}, {}'.format(Y_train.shape, Y_test.shape))
```

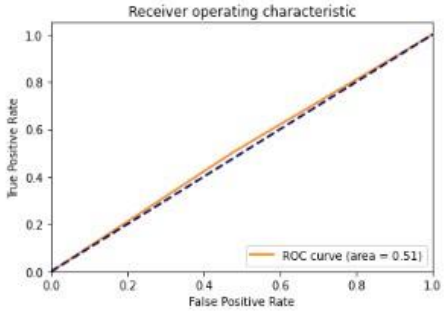
```
(13993, 9), (4665, 9)
(13993,), (4665,)
```

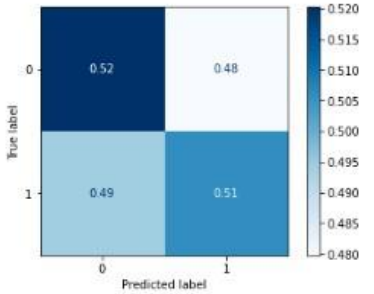
6. Построение базового решения

Ввод [30]:
```
models = {'KNN_3':KNeighborsClassifier(n_neighbors=3),
          'SVC':SVC(),
          'Tree':DecisionTreeClassifier(),
          'RF':RandomForestClassifier(),
          'GB':GradientBoostingClassifier()}
```

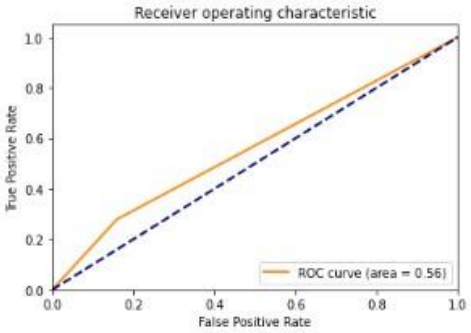
Ввод [31]:
```
for model_name, model in models.items():
    test_model(model_name, model, metricLogger)
```

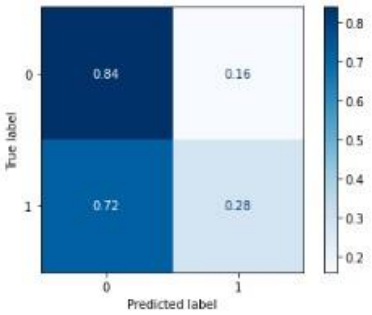
```
********************************************************************************
KNeighborsClassifier(n_neighbors=3)
********************************************************************************
precision: 0.5060922541340296
recall: 0.5058721183123097
accuracy: 0.5131832797427652
roc_auc: 0.5130797616075496
********************************************************************************
```



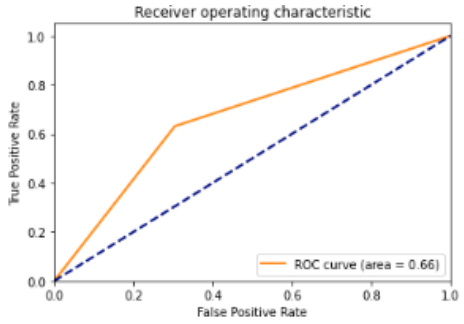


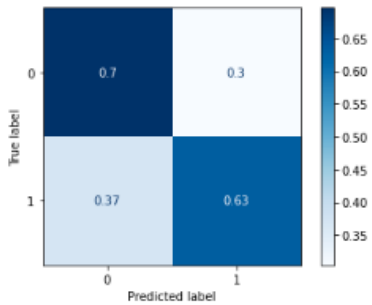

```
********************************************************************************
SVC()
********************************************************************************
precision: 0.6296660117878192
recall: 0.27881687690300133
accuracy: 0.5637727759914255
roc_auc: 0.559738108781171
********************************************************************************
```



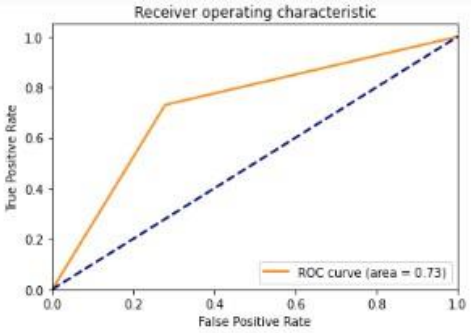


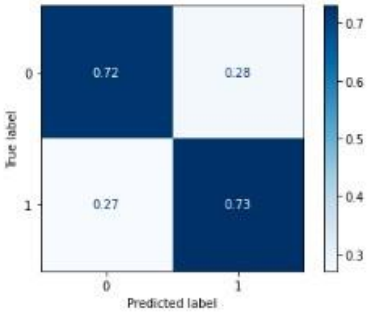

```
**********************************************************************************
DecisionTreeClassifier()
**********************************************************************************
precision: 0.6683555964993091
recall: 0.6311439756415833
accuracy: 0.6638799571275456
roc_auc: 0.6634164510498701
**********************************************************************************
```



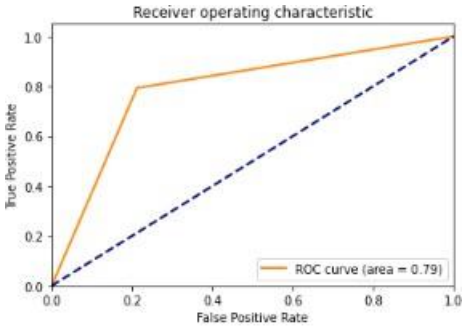


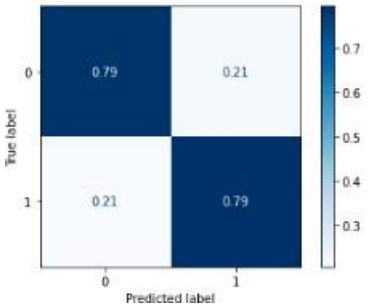

```
**********************************************************************************
RandomForestClassifier()
**********************************************************************************
precision: 0.7180145485665383
recall: 0.7298825576337538
accuracy: 0.7256162915326902
roc_auc: 0.725676697244603
**********************************************************************************
```






```
**********************************************************************************
GradientBoostingClassifier()
**********************************************************************************
precision: 0.7840860215053763
recall: 0.7929534580252283
accuracy: 0.7903536977491962
roc_auc: 0.7903905075417772
**********************************************************************************
```





7. Подбор гиперпараметров

Ввод [32]:
```
X_train.shape
```

Out[32]:
```
(13993, 9)
```

Дерево решений

Ввод [33]:
```
%%time
tree_parameters = {'max_depth': randint(1, 100)}
clf_gs = RandomizedSearchCV(DecisionTreeClassifier(), tree_parameters, cv=5, scoring='accuracy', n_jobs=-1, n_iter=10)
clf_gs.fit(X, Y)
```

```
CPU times: user 256 ms, sys: 211 ms, total: 466 ms
Wall time: 2.96 s
```

Out[33]:
```
RandomizedSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
                   param_distributions={'max_depth': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f7be6b
64fd0>},
                   scoring='accuracy')
```

Ввод [34]:
```
clf_gs.best_estimator_
```

Out[34]:
```
DecisionTreeClassifier(max_depth=8)
```

Лучшее значение параметров:

Ввод [35]:
```
clf_gs.best_params_
```

Out[35]:
```
{'max_depth': 8}
```

Средний результат:

Ввод [36]:
```
clf_gs.cv_results_['mean_test_score']
```

Out[36]:
```
array([0.63259405, 0.69846427, 0.63178994, 0.63173623, 0.63125384,
       0.63077134, 0.63023535, 0.66212556, 0.63259395, 0.63109312])
```

Градиентный бустинг

Ввод [37]:
```
 [1e-3, 1e-2, 1e-1], 'subsample': np.linspace(0.7,0.9,1), 'n_estimators': randint(1, 100), 'max_depth': randint(1, 10)}
entBoostingClassifier(), gb_parameters, cv=5, scoring='accuracy', n_jobs=-1, n_iter=10)
```

```
CPU times: user 4.26 s, sys: 45.4 ms, total: 4.31 s
Wall time: 17.2 s
```

Out[37]:
```
RandomizedSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
                   param_distributions={'learning_rate': [0.001, 0.01, 0.1],
                                        'max_depth': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f7be66
d7490>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f7b
980d8b50>,
                                        'subsample': array([0.7])},
                   scoring='accuracy')
```

Ввод [38]:
```
clf_gs.best_estimator_
```

Out[38]:
```
GradientBoostingClassifier(learning_rate=0.001, max_depth=7, n_estimators=84,
                           subsample=0.7)
```

Лучшее значение параметров:

Ввод [39]:
```
clf_gs.best_params_
```

Out[39]:
```
{'learning_rate': 0.001, 'max_depth': 7, 'n_estimators': 84, 'subsample': 0.7}
```

Средний результат:

Ввод [40]:
```
clf_gs.cv_results_['mean_test_score']
```

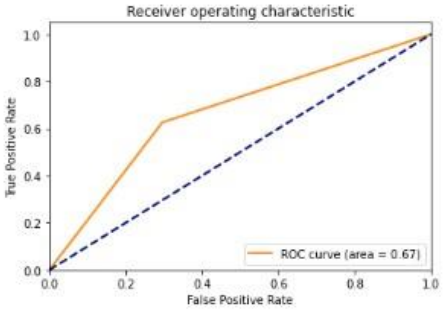
Out[40]:
```
array([0.69647879, 0.67664707, 0.693474  , 0.72467276, 0.72086754,
       0.71904432, 0.69347399, 0.70511156, 0.50305499, 0.69878427])
```

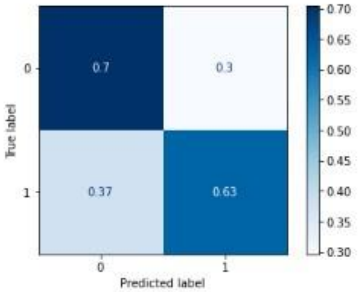
8. Сравнение качества полученных моделей с качеством baseline-моделей

Дерево решений

Ввод [41]:
```
test_model('Tree', DecisionTreeClassifier(), metricLogger)
```

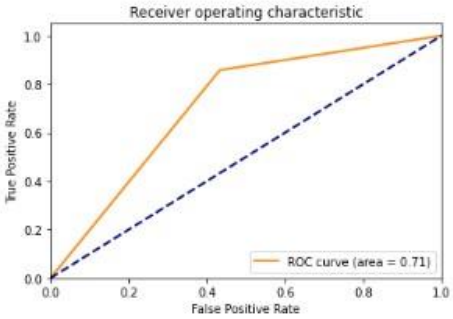
```
********************************************************************************
DecisionTreeClassifier()
********************************************************************************
precision: 0.6727442730247779
recall: 0.6259243149195303
accuracy: 0.6655948553054662
roc_auc: 0.665033163376925
********************************************************************************
```

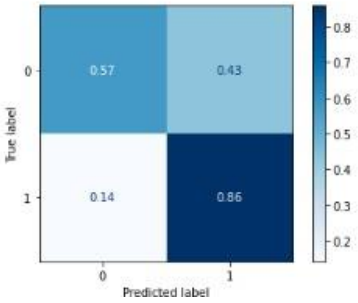




Ввод [42]:
```
test_model('Tree_tuned', RandomizedSearchCV(DecisionTreeClassifier(), tree_parameters, cv=5, scoring='accuracy', n_jobs
```

```
********************************************************************************
RandomizedSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
                   param_distributions={'max_depth': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f7be6b
64fd0>},
                   scoring='accuracy')
********************************************************************************
precision: 0.6582109479305741
recall: 0.8577642453240539
accuracy: 0.7103965702036441
roc_auc: 0.7124831370322721
********************************************************************************
```

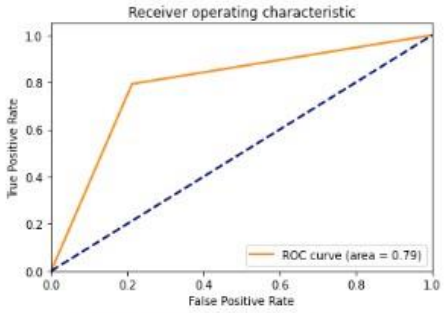

Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.71)
0.0
0.2
0.4
0.6
0.8
1.0

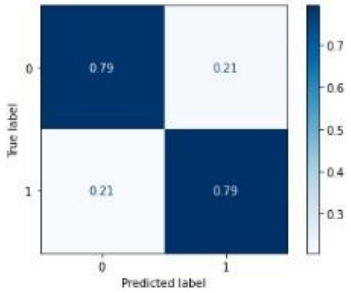

0.57
0.43
0.14
0.86
True label
Predicted label
0
1
0.8
0.7
0.6
0.5
0.4
0.3
0.2

Градиентный бустинг

Ввод [43]:
```
test_model('GB', GradientBoostingClassifier(), metricLogger)
```

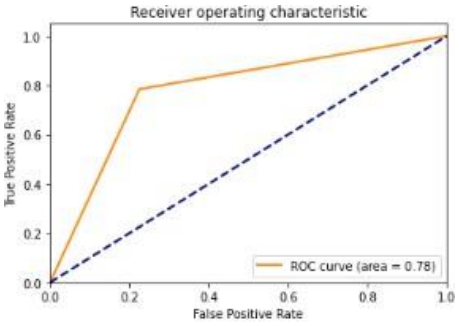
```
********************************************************************************
GradientBoostingClassifier()
********************************************************************************
precision: 0.7840860215053763
recall: 0.7929534580252283
accuracy: 0.7903536977491962
roc_auc: 0.7903905075417772
********************************************************************************
```

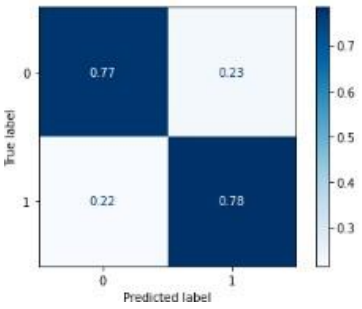




Ввод [44]:
```
test_model('GB_tuned', RandomizedSearchCV(GradientBoostingClassifier(), gb_parameters, cv=5, scoring='accuracy', n_jobs
```

```
********************************************************************************
RandomizedSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
                   param_distributions={'learning_rate': [0.001, 0.01, 0.1],
                                        'max_depth': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f7be66
d7490>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7f7b
980d8b50>,
                                        'subsample': array([0.7])},
                   scoring='accuracy')
********************************************************************************
precision: 0.771832191780822
recall: 0.7842540234884733
accuracy: 0.7794212218649518
roc_auc: 0.779489649106874
********************************************************************************
```


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.78)
0.0
0.2
0.4
0.6
0.8
1.0

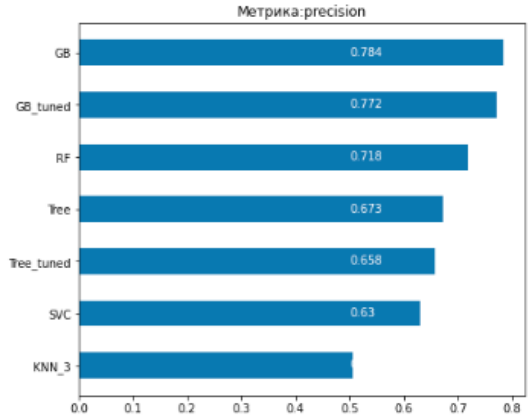

0.77
0.23
0.22
0.78
True label
Predicted label
0
1
0.7
0.6
0.5
0.4
0.3

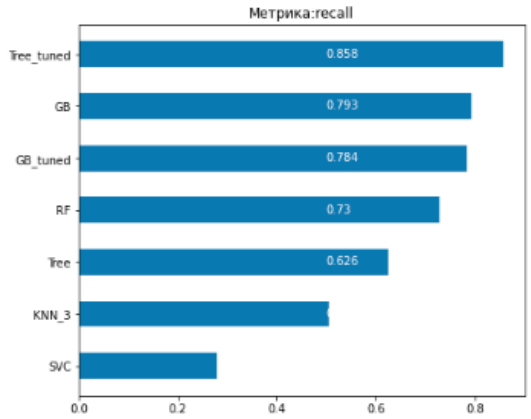
9. Выводы о качестве построенных моделей

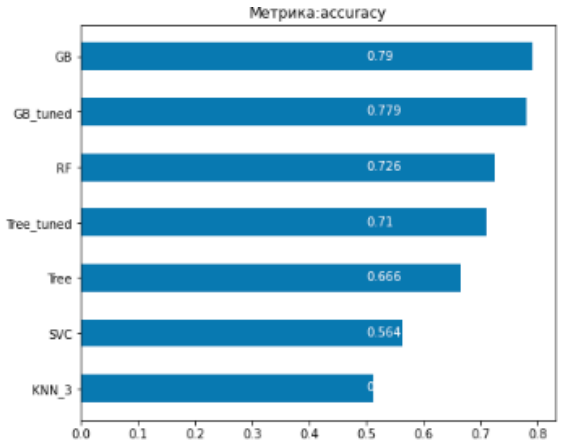
Ввод [45]:
```
metrics = ['precision', 'recall', 'accuracy', 'roc_auc']
metrics
```

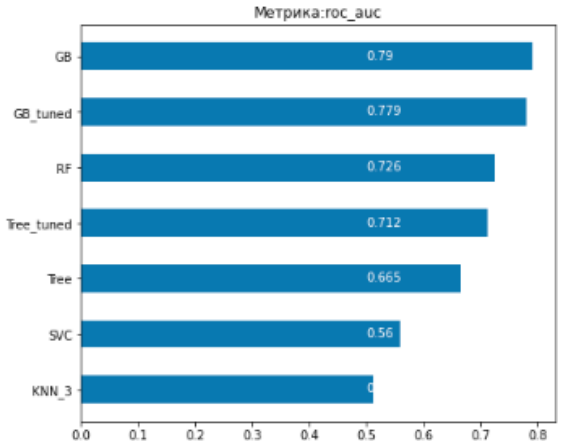
Out[45]:
```
['precision', 'recall', 'accuracy', 'roc_auc']
```

Ввод [46]:
```
for metric in metrics:
    metricLogger.plot('Метрика:' + metric, metric, figsize=(7, 6))
```









Вывод

Было использовано 5 моделей, для двух из них был проведен подбор гиперпараметров - дерево решений и градиентный бустинг. Лучше всего себя показала модель gradient boosting. Худший результат показали модели KNN и SVC с точностью 0,5. В результате подбора гиперпараметров удалось улучшить точность модели decision tree, однако для модели gradient boosting изменение результатов было незначительным.

Заключение

В данном научно-исследовательской работе мы выполнили типовую задачу машинного обучения. Провели анализ данных, преобразовали готовый датасет под наши потребности, подобрали модели, а также подобрали наиболее подходящие гиперпараметры.

В нашем случае классификатор на основе градиентного бустинга показал лучший результат, однако для данной модели изменение результатов в результате подбора гиперпараметров было незначительным.

В данном проекте были закреплены все знания, полученные в курсе лекций и на лабораторных работах. Часть информации была найдена в различных открытых источниках в интернете.

Проделанная работа вызвала интерес к предмету и дальнейшей работе в этой сфере, которая является одной из самых перспективных и актуальных в современном мире.

Литература

1. Лекции за 2023 год по дисциплине «Технологии машинного обучения»
2. https://scikit-learn.org/stable/index.html
3. https://www.kaggle.com/datasets
4. http://www.machinelearning.ru/