|  |  |
| --- | --- |
| Изображение выглядит как эмблема, герб, нашивка, символ  Автоматически созданное описание | **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:***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Анализ данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_ИУ5-64Б\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Белоезров Д.С.\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **\_\_\_\_\_****Ю.Е. Гапанюк\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2024 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 \_\_\_ИУ5\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_В.И. Терехов\_\_

(И.О.Фамилия)

«\_07\_» \_\_\_\_февраля\_\_\_\_\_\_\_ 2024 г.

**ЗАДАНИЕ**

**на выполнение научно-исследовательской работы**

по теме \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_ИУ5-64Б\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Белозеров Дмитрий Сергеевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ИССЛЕДОВАТЕЛЬСКАЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_КАФЕДРА\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения НИР: 25% к \_\_\_\_ нед., 50% к \_\_\_\_ нед., 75% к \_\_\_ нед., 100% к \_\_\_\_ нед.

***Техническое задание*** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на \_41\_\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_07\_» \_\_\_\_февраля\_\_\_\_\_\_\_ 2024 г.

**Руководитель НИР**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_Ю.Е. Гапанюк\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_**Д. С. Белозеров**\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

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

**СОДЕРЖАНИЕ**

[Введение 4](#_TOC_250010)

Основная часть 5

1. [Выбор и подготовка набора данных 6](#_TOC_250009)
2. [Разведочный анализ 9](#_TOC_250008)
3. Корреляционный анализ 15
4. [Метрики для оценки качества моделей 17](#_TOC_250007)
5. [Формирование обучающей и тестовой выборок 19](#_TOC_250006)
6. [Построение базового решения 20](#_TOC_250005)
7. [Подбор гиперпараметров 25](#_TOC_250004)
8. Сравнение качества полученных моделей 29
9. [Выводы о качестве построенных моделей 37](#_TOC_250003)
10. [Вывод 40](#_TOC_250002)

[Заключение 41](#_TOC_250001)

[Литература 42](#_TOC_250000)

## Введение

В данном курсовом проекте 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 обуче- ния 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 визу- ально 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.

Приведенная схема исследования является рекомендуемой. В зависимости от решаемой задачи возможны модификации.

# Выбор и подготовка набора данных

In [74]:

**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\_ **from** sklearn.metrics **import** roc\_curve, roc\_auc\_score, accuracy\_score, confusion\_matr **from** sklearn.model\_selection **import** GridSearchCV, RandomizedSearchCV

**from** scipy.stats **import** randint, uniform

**from** sklearn.ensemble **import** GradientBoostingClassifier

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

In [2]:

train\_data **=** pd.read\_csv('../data/SpotifyFeatures.csv') train\_data.head()

Out[2]:

**genre artist\_name track\_name track\_id popularity acousticness dancea**

**0** Movie

Henri Salvador

C'est beau de faire un

Show

Perdu

0BRjO6ga9RKCKjfDqeFgWV 0 0.611 0

Movie Martin & les

**1**

fées

Movie Joseph Williams

**2**

Movie Henri Salvador

**3**

Movie Fabien Nataf

**4**

d'avance (par Gad Elmaleh)

Don't Let Me Be Lonely Tonight

Dis-moi Monsieur Gordon Cooper

Ouverture

0BjC1NfoEOOusryehmNudP 1 0.246 0

0CoSDzoNIKCRs124s9uTVy 3 0.952 0

|  |  |  |  |
| --- | --- | --- | --- |
| 0Gc6TVm52BwZD07Ki6tIvf | 0 | 0.703 | 0 |
| 0IuslXpMROHdEPvSl1fTQK | 4 | 0.950 | 0 |

Размер набора:

In [3]:

train\_data.shape

Out[3]: (232725, 18)

Удалим лишние столбцы:

In [4]:

train\_data **=** train\_data.filter(['genre', 'artist\_name', 'track\_name', 'energy', 'lou train\_data.head()

Out[4]:

**genre artist\_name track\_name energy loudness speechiness liveness popularity dancea**

**0** Movie

Henri Salvador

C'est beau de faire un

Show

Perdu

0.910 -1.828 0.0525 0.3460 0 0

Movie Martin & les

**1**

fées

Movie Joseph Williams

**2**

Movie Henri Salvador

**3**

Movie Fabien Nataf

**4**

d'avance (par Gad Elmaleh)

Don't Let Me Be Lonely Tonight

Dis-moi Monsieur Gordon Cooper

Ouverture

0.737 -5.559 0.0868 0.1510 1 0

0.131 -13.879 0.0362 0.1030 3 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0.326 | -12.178 | 0.0395 | 0.0985 | 0 | 0 |
| 0.225 | -21.150 | 0.0456 | 0.2020 | 4 | 0 |

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:

In [5]:

train\_data.shape[1]

Out[5]:

12

Переименуем названия столбцов:

In [6]:

train\_data.rename(columns**=**{'genre': 'Genre', 'artist\_name': 'Artist', 'acousticness' train\_data.head()

Out[6]:

Salvador

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Genre Artist Track Energy** | **Loudness** | **Speechiness** | **Liveness** | **Popularity** | **Danceabilit** |
| C'est  **0** Movie Henri beau de 0.910 | -1.828 | 0.0525 | 0.3460 | 0 | 0.38 |

Movie Martin &

**1**

les fées

Movie Joseph Williams

**2**

Movie Henri Salvador

**3**

Movie Fabien Nataf

**4**

faire un Show

Perdu d'avance (par Gad Elmaleh)

Don't Let Me Be Lonely Tonight

Dis-moi Monsieur Gordon Cooper

Ouverture

0.737 -5.559 0.0868 0.1510 1 0.59

0.131 -13.879 0.0362 0.1030 3 0.66

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0.326 | -12.178 | 0.0395 | 0.0985 | 0 | 0.24 |
| 0.225 | -21.150 | 0.0456 | 0.2020 | 4 | 0.33 |

# Разведочный анализ

Проверим пропуски:

In [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:

In [8]:

train\_data['Genre'].nunique()

Out[8]:

27

Количество песен каждого жанра:

In [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):

In [10]:

top\_genres **=** ['Pop', 'Rock']

In [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:

In [12]:

train\_data.shape

Out[12]: (18658, 12)

Подсчитаем количество исполнителей:

In [13]:

train\_data['Artist'].nunique()

Out[13]:

3297

Выведем топ-5 исполнителей каждого жанра:

In [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  Name: Artist, | | 66  dtype: | int64 |
| Rock:  The Beatles | |  | 145 |
| Queen | |  | 97 |
| Led Zeppelin | |  | 76 |
| Panic! At The | | Disco | 74 |

Imagine Dragons 71

Name: Artist, dtype: int64

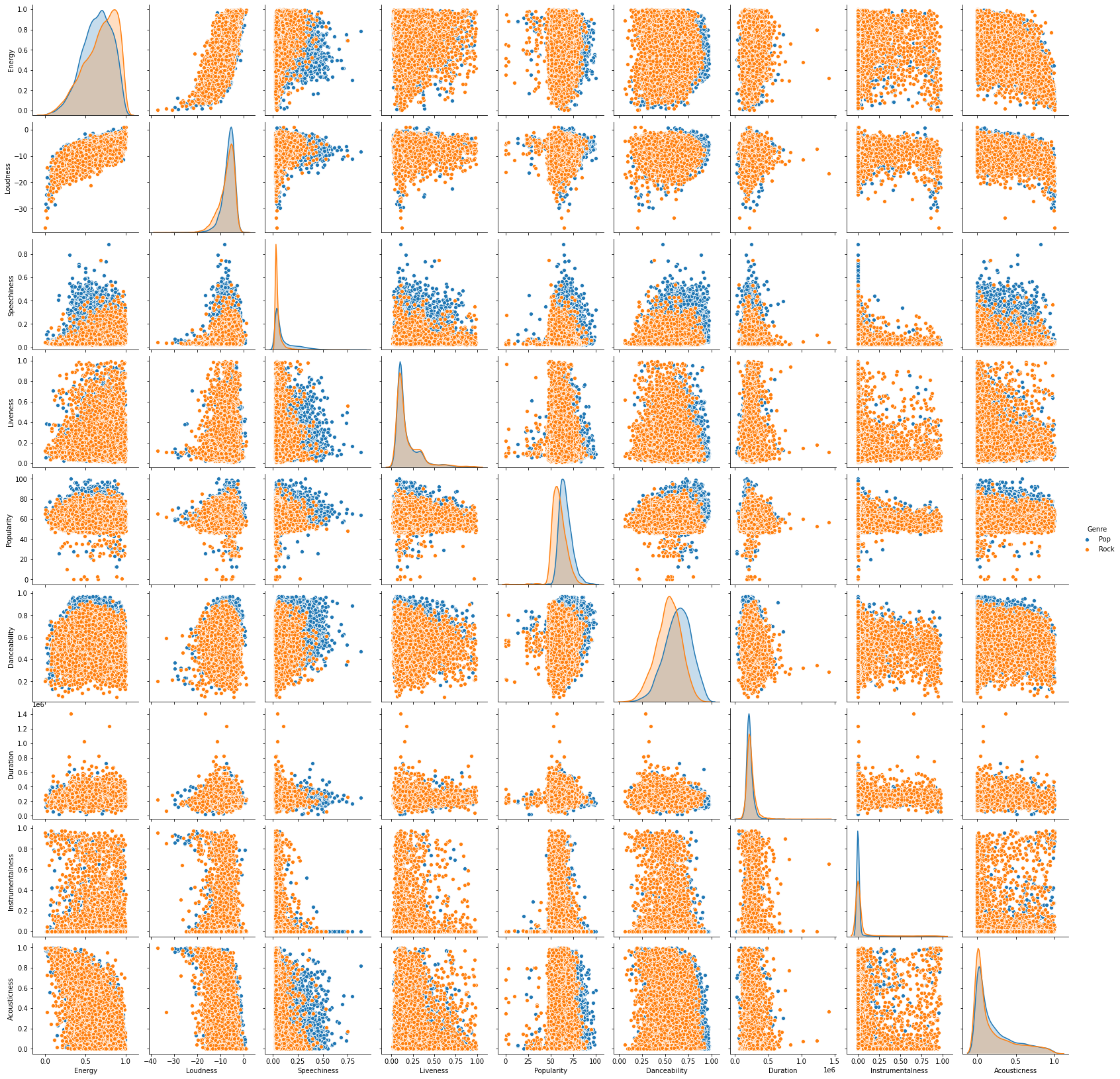
Для понимании структуры набора данных построим графики:

In [15]:

sns.pairplot(train\_data, hue**=**"Genre")

Out[15]:

<seaborn.axisgrid.PairGrid at 0x11774fb70>



Количество уникальных артистов каждого жанра:

In [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()

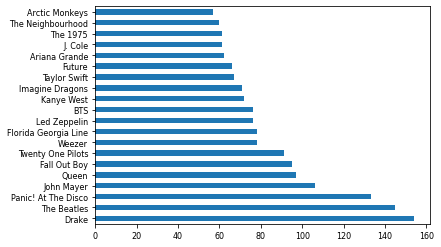
Топ-20 исполнителей по количеству песен:

In [17]:

train\_data['Artist'].value\_counts().head(20).plot(kind**=**'barh', fontsize**=**8)

Out[17]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x11774f2e8>



1. **Корреляционный анализ данных**

In [18]:

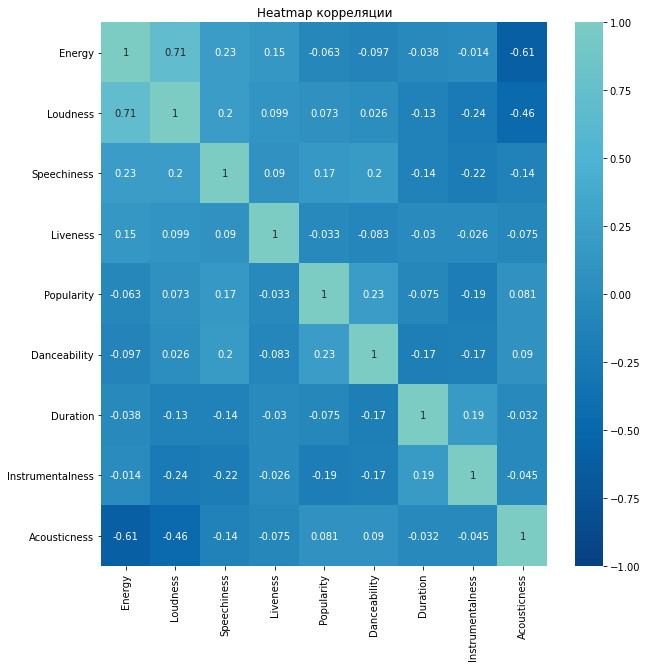
correlation**=**train\_data.corr(method**=**'spearman') plt.figure(figsize**=**(10,10))

plt.title('Heatmap корреляции')

sns.heatmap(correlation,annot**=True**,vmin**=-**1,vmax**=**1,cmap**=**"GnBu\_r",center**=**1)

Out[18]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x11d6c2da0>



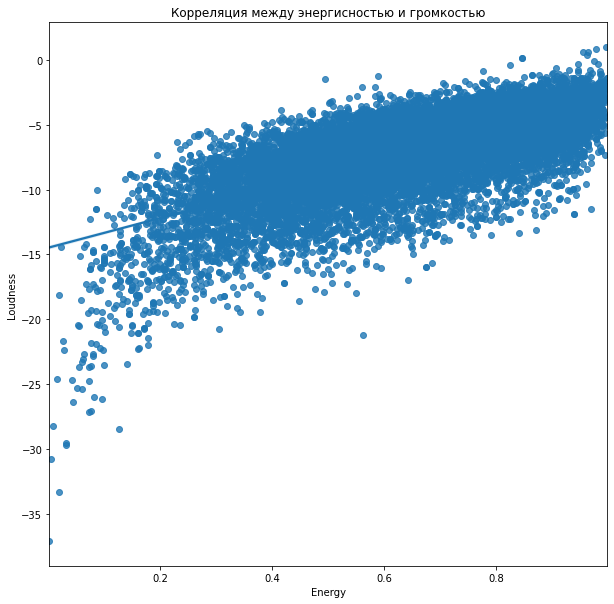
Проверим связь между громкостью и энергичностью:

In [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]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x11d9217b8>



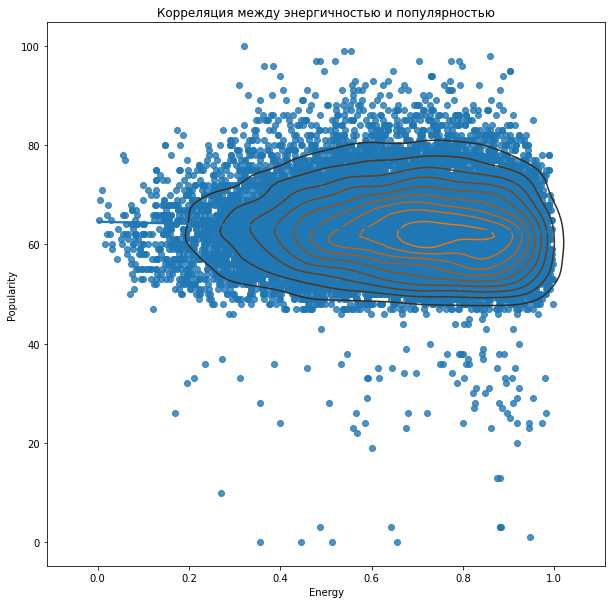
In [20]:

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]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x11d978588>



# Метрики для оценки качества моделей

In [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()

In [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)].inde 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()

In [23]:

metricLogger **=** MetricLogger()

In [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()

# Формирование обучающей и тестовой выборок

In [25]:

features **=** ['Genre', 'Acousticness', 'Instrumentalness', 'Energy', 'Loudness', 'Spee

In [26]:

train\_data\_enc **=** train\_data.filter(features) train\_data\_enc.head()

Out[26]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Genre** | **Acousticness** | **Instrumentalness** | **Energy** | **Loudness** | **Speechiness** | **Liveness Dan** |
| **107802** | Pop | 0.0421 | 0.000000 | 0.554 | -5.290 | 0.0917 | 0.1060 |
| **107803** | Pop | 0.1630 | 0.000002 | 0.539 | -7.399 | 0.1780 | 0.1010 |
| **107804** | Pop | 0.5780 | 0.000000 | 0.321 | -10.744 | 0.3230 | 0.0884 |
| **107805** | Pop | 0.1490 | 0.000000 | 0.364 | -11.713 | 0.2760 | 0.2710 |
| **107806** | Pop | 0.5560 | 0.000000 | 0.479 | -5.574 | 0.0466 | 0.0703 |

Выполним кодирование признака жанра:

In [27]:

le **=** LabelEncoder()

train\_data\_enc['Genre'] **=** le.fit\_transform(train\_data['Genre']);

Разделим выборки:

In [28]:

X **=** train\_data\_enc.drop('Genre', axis**=**1) Y **=** train\_data\_enc['Genre']

In [29]:

X\_train, X\_test, Y\_train, Y\_test **=** train\_test\_split(X, Y, test\_size**=**0.25, random\_sta print('{}, {}'.format(X\_train.shape, X\_test.shape))

print('{}, {}'.format(Y\_train.shape, Y\_test.shape))

(13993, 9), (4665, 9)

(13993,), (4665,)

# Построение базового решения

In [30]:

models **=** {'KNN\_3':KNeighborsClassifier(n\_neighbors**=**3), 'SVC':SVC(),

'Tree':DecisionTreeClassifier(), 'RF':RandomForestClassifier(), 'GB':GradientBoostingClassifier()}

In [31]:

**for** model\_name, model **in** models.items(): test\_model(model\_name, model, metricLogger)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowsk i',

=2,

metric\_params=None, n\_jobs=None, n\_neighbors=3, p

weights='uniform')

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.5060922541340296

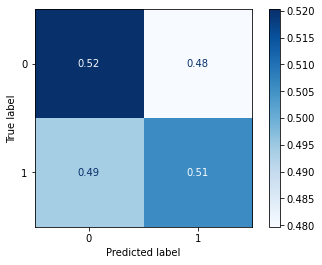
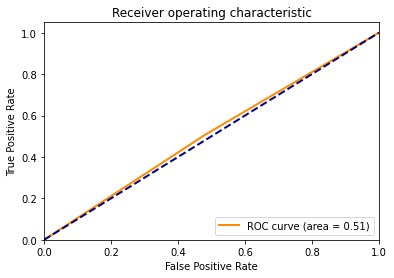
recall: 0.5058721183123097

accuracy: 0.5131832797427652

roc\_auc: 0.5130797616075496

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

SVC(C=1.0, break\_ties=False, cache\_size=200, class\_weight=None, coef0= 0.0,

decision\_function\_shape='ovr', degree=3, gamma='scale', kernel='rb

f',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.6296660117878192

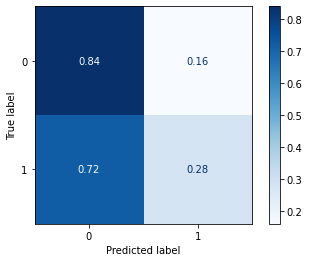
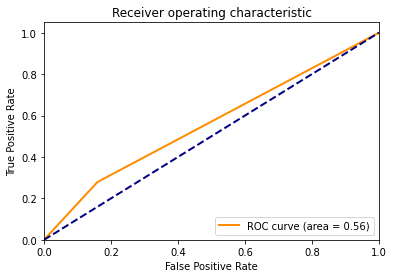
recall: 0.27881687690300133

accuracy: 0.5637727759914255

roc\_auc: 0.559738108781171

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gi ni',

es=None, one,

ed',

max\_depth=None, max\_features=None, max\_leaf\_nod min\_impurity\_decrease=0.0, min\_impurity\_split=N

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, presort='deprecat

random\_state=None, splitter='best')

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.6665111940298507

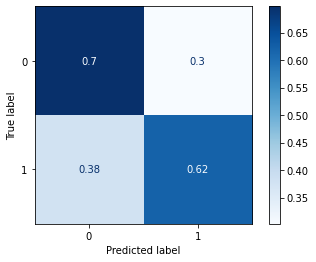
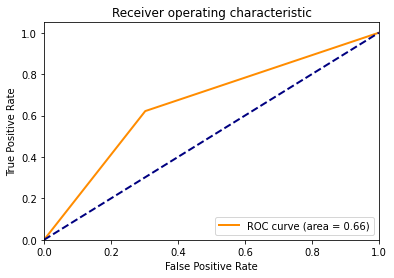
recall: 0.6215745976511526

accuracy: 0.6602357984994641

roc\_auc: 0.6596883977266752

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=Non e,

='auto',

one,

e,

criterion='gini', max\_depth=None, max\_features

max\_leaf\_nodes=None, max\_samples=None, min\_impurity\_decrease=0.0, min\_impurity\_split=N

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=None, oob\_score=False, random\_state=Non

verbose=0, warm\_start=False)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.7151335311572701

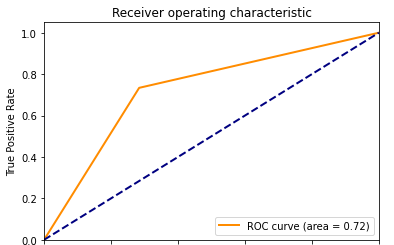
recall: 0.7337973031752936

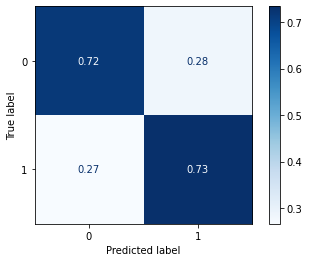
accuracy: 0.7247588424437299

roc\_auc: 0.7248868172681201

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

GradientBoostingClassifier(ccp\_alpha=0.0, criterion='friedman\_mse', in it=None,

th=3,

it=None,

100,

d', 1,

learning\_rate=0.1, loss='deviance', max\_dep

max\_features=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_spl

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=

n\_iter\_no\_change=None, presort='deprecate random\_state=None, subsample=1.0, tol=0.000

validation\_fraction=0.1, verbose=0, warm\_start=False)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.7844234079173839

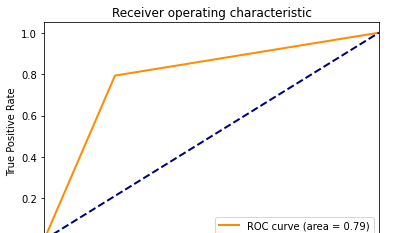
recall: 0.7929534580252283

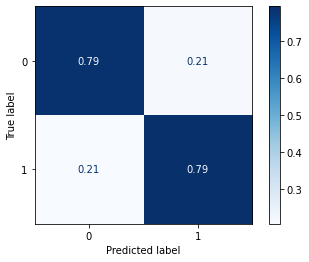
accuracy: 0.7905680600214362

roc\_auc: 0.7906018346761813

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*





# Подбор гиперпараметров

In [32]:

X\_train.shape

Out[32]:

(13993, 9)

Дерево решений

In [33]:

**%%**time

tree\_parameters **=** {'max\_depth': randint(1, 100)}

clf\_gs **=** RandomizedSearchCV(DecisionTreeClassifier(), tree\_parameters, cv**=**5, scoring clf\_gs.fit(X, Y)

CPU times: user 554 ms, sys: 98.4 ms, total: 652 ms Wall time: 6 s

Out[33]:

RandomizedSearchCV(cv=5, error\_score=nan,

estimator=DecisionTreeClassifier(ccp\_alpha=0.0,

class\_weight=None, criterion='gini', max\_depth=None, max\_features=None, max\_leaf\_nodes=Non

e, ase=0.0,

=None, 1,

2,

n\_leaf=0.0, d',

min\_impurity\_decre min\_impurity\_split min\_samples\_leaf= min\_samples\_split= min\_weight\_fractio presort='deprecate

random\_state=None, splitter='best'),

iid='deprecated', n\_iter=10, n\_jobs=-1, param\_distributions={'max\_depth': <scipy.stats.\_dis

tn\_infrastructure.rv\_frozen object at 0x11de52198>},

pre\_dispatch='2\*n\_jobs', random\_state=None, refit=T

rue, se=0)

return\_train\_score=False, scoring='accuracy', verbo

In [34]:

clf\_gs.best\_estimator\_

Out[34]:

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gi ni',

=None, one,

ed',

max\_depth=13, max\_features=None, max\_leaf\_nodes min\_impurity\_decrease=0.0, min\_impurity\_split=N

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, presort='deprecat

random\_state=None, splitter='best')

Лучшее значение параметров:

|  |  |
| --- | --- |
| In [35]: |  |
| clf\_gs.best\_params\_ |  |
| Out[35]: |  |
| {'max\_depth': 13} |  |
| Средний результат: |  |
| In [36]: |  |
| clf\_gs.cv\_results\_['mean\_test\_score'] |  |
| Out[36]: |  |
| array([0.63050352, 0.62975309, 0.63034289, | 0.65966003, 0.63157523, |
| 0.62819865, 0.63586335, 0.62959213, | 0.62996768, 0.62964593]) |

Градиентный бустинг

In [75]:

**%%**time

gb\_parameters **=** {'learning\_rate': [1e-3, 1e-2, 1e-1], 'subsample': uniform(0.5, 1.0) clf\_gs **=** RandomizedSearchCV(GradientBoostingClassifier(), gb\_parameters, cv**=**5, scori clf\_gs.fit(X, Y)

CPU times: user 2.58 s, sys: 94.4 ms, total: 2.68 s Wall time: 34.2 s

Out[75]:

RandomizedSearchCV(cv=5, error\_score=nan,

estimator=GradientBoostingClassifier(ccp\_alpha=0.0,

criterion='fri

edman\_mse',

0.1,

e',

one,

=None, ecrease=0.0, plit=None, af=1,

lit=2, ction\_leaf=0.0, 00,

0.1],

init=None, learning\_rate=

loss='devianc

max\_depth=3, max\_features=N

max\_leaf\_nodes min\_impurity\_d min\_impurity\_s min\_samples\_le min\_samples\_sp min\_weight\_fra n\_estimators=1 n\_ite...

param\_distributions={'learning\_rate': [0.001, 0.01,

'max\_depth': <scipy.stats.\_dis

tn\_infrastructure.rv\_frozen object at 0x11dbc4400>,

'n\_estimators': <scipy.stats.\_ distn\_infrastructure.rv\_frozen object at 0x11dbc46a0>,

'subsample': <scipy.stats.\_dis tn\_infrastructure.rv\_frozen object at 0x11e34b2e8>},

pre\_dispatch='2\*n\_jobs', random\_state=None, refit=T

rue, se=0)

return\_train\_score=False, scoring='accuracy', verbo

In [76]:

clf\_gs.best\_estimator\_

Out[76]:

GradientBoostingClassifier(ccp\_alpha=0.0, criterion='friedman\_mse', in it=None,

th=2,

it=None,

86,

d', 6847, e=0,

learning\_rate=0.1, loss='deviance', max\_dep

max\_features=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_spl

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=

n\_iter\_no\_change=None, presort='deprecate random\_state=None, subsample=0.665363619680 tol=0.0001, validation\_fraction=0.1, verbos warm\_start=False)

Лучшее значение параметров:

In [77]:

clf\_gs.best\_params\_

Out[77]:

{'learning\_rate': 0.1,

'max\_depth': 2,

'n\_estimators': 86,

'subsample': 0.6653636196806847}

Средний результат:

In [78]:

clf\_gs.cv\_results\_['mean\_test\_score']

Out[78]:

|  |  |  |
| --- | --- | --- |
| array([0.72628059, 0.73115791, | nan, | nan, 0.69878671, |
| nan, 0.71915108, | nan, | nan, 0.72478003]) |

1. **Сравнение качества полученных моделей с качеством baseline-моделей**

Дерево решений

In [60]:

test\_model('Tree', DecisionTreeClassifier(), metricLogger)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gi ni',

es=None, one,

ed',

max\_depth=None, max\_features=None, max\_leaf\_nod min\_impurity\_decrease=0.0, min\_impurity\_split=N

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, presort='deprecat

random\_state=None, splitter='best')

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.6702226345083488

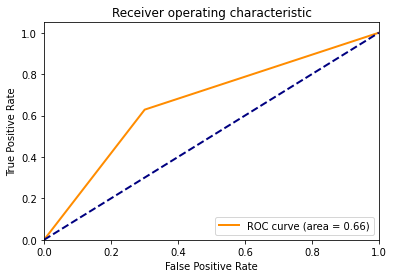
recall: 0.6285341452805567

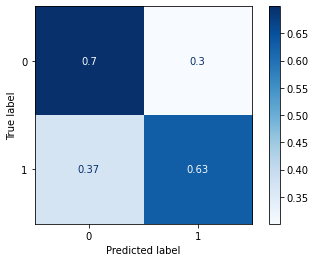
accuracy: 0.6645230439442658

roc\_auc: 0.6640134800789934

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*





In [61]:

test\_model('Tree\_tuned', RandomizedSearchCV(DecisionTreeClassifier(), tree\_parameter

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

RandomizedSearchCV(cv=5, error\_score=nan,

estimator=DecisionTreeClassifier(ccp\_alpha=0.0,

class\_weight=None, criterion='gini', max\_depth=None, max\_features=None, max\_leaf\_nodes=Non

e, ase=0.0,

=None, 1,

2,

n\_leaf=0.0, d',

min\_impurity\_decre min\_impurity\_split min\_samples\_leaf= min\_samples\_split= min\_weight\_fractio presort='deprecate

random\_state=None, splitter='best'),

iid='deprecated', n\_iter=10, n\_jobs=-1, param\_distributions={'max\_depth': <scipy.stats.\_dis

tn\_infrastructure.rv\_frozen object at 0x11de52198>},

pre\_dispatch='2\*n\_jobs', random\_state=None, refit=T

rue,

se=0)

return\_train\_score=False, scoring='accuracy', verbo

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.7199036918138042

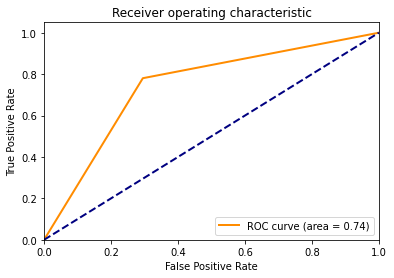
recall: 0.7803392779469335

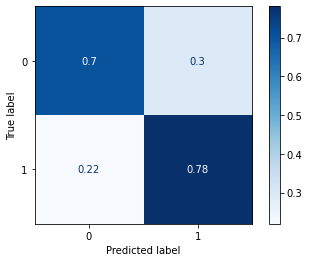
accuracy: 0.7421221864951768

roc\_auc: 0.7426632991594346

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*





Градиентный бустинг

In [79]:

test\_model('GB', GradientBoostingClassifier(), metricLogger)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

GradientBoostingClassifier(ccp\_alpha=0.0, criterion='friedman\_mse', in it=None,

th=3,

it=None,

100,

d', 1,

learning\_rate=0.1, loss='deviance', max\_dep

max\_features=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_spl

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=

n\_iter\_no\_change=None, presort='deprecate random\_state=None, subsample=1.0, tol=0.000

validation\_fraction=0.1, verbose=0, warm\_start=False)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.7844234079173839

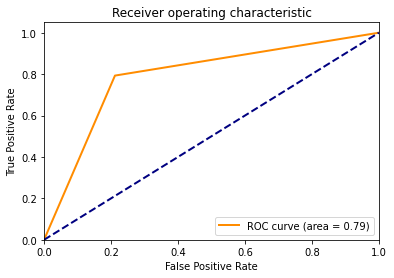
recall: 0.7929534580252283

accuracy: 0.7905680600214362

roc\_auc: 0.7906018346761813

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*





In [80]:

test\_model('GB\_tuned', RandomizedSearchCV(GradientBoostingClassifier(), gb\_parameter

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

RandomizedSearchCV(cv=5, error\_score=nan,

estimator=GradientBoostingClassifier(ccp\_alpha=0.0,

criterion='fri

edman\_mse',

0.1,

e',

one,

=None, ecrease=0.0, plit=None, af=1,

lit=2, ction\_leaf=0.0, 00,

0.1],

init=None, learning\_rate=

loss='devianc

max\_depth=3, max\_features=N

max\_leaf\_nodes min\_impurity\_d min\_impurity\_s min\_samples\_le min\_samples\_sp min\_weight\_fra n\_estimators=1 n\_ite...

param\_distributions={'learning\_rate': [0.001, 0.01,

'max\_depth': <scipy.stats.\_dis

tn\_infrastructure.rv\_frozen object at 0x11dbc4400>,

'n\_estimators': <scipy.stats.\_ distn\_infrastructure.rv\_frozen object at 0x11dbc46a0>,

'subsample': <scipy.stats.\_dis tn\_infrastructure.rv\_frozen object at 0x11e34b2e8>},

pre\_dispatch='2\*n\_jobs', random\_state=None, refit=T

rue,

se=0)

return\_train\_score=False, scoring='accuracy', verbo

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

precision: 0.7754580315296122

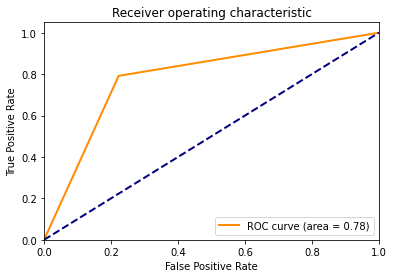
recall: 0.7916485428447151

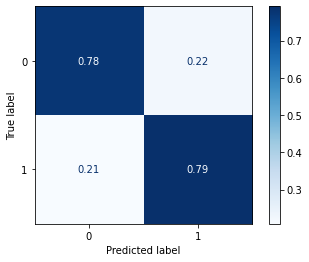
accuracy: 0.7843515541264737

roc\_auc: 0.7844548715914192

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*





# Выводы о качестве построенных моделей

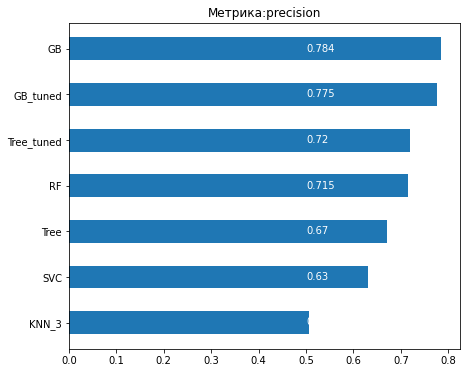
In [81]:

metrics **=** ['precision', 'recall', 'accuracy', 'roc\_auc'] metrics

Out[81]:

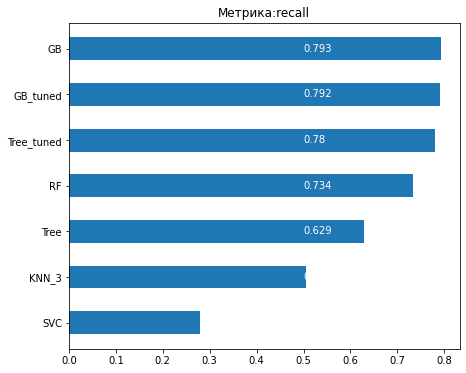
['precision', 'recall', 'accuracy', 'roc\_auc']

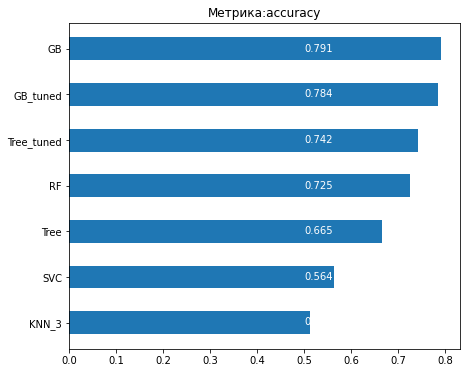
In [82]:

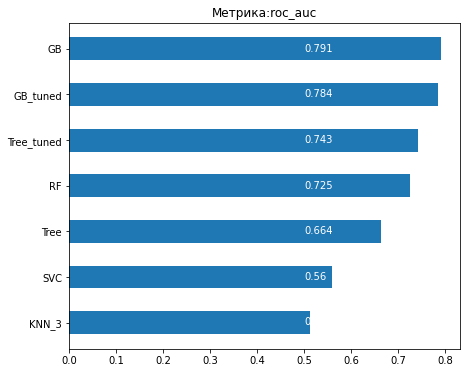


**for** metric **in** metrics:

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 курсовом проекте мы выполнили типовую задачу машинного обуче- ния. Провели анализ данных, преобразовали готовый датасет под наши потреб- ности, подобрали модели, а также подобрали наиболее подходящие гиперпара- метры.

### В нашем случае классификатор на основе градиентного бустинга показал луч- ший 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 совре- менном мире.

## Литература

1. Рукописные лекции за 2024 год по дисциплине «Технологии машинного обучения»
2. https://scikit-learn.org/stable/index.html
3. https:/[/www.ka](http://www.kaggle.com/datasets)g[gle.com/datasets](http://www.kaggle.com/datasets)
4. <http://www.machinelearning.ru/>