# РТ5-61Б, Забурунов Л. В.

# Технологии Машинного Обучения

# Курсовая Работа

## "Решение комплексной задачи машинного обучения для дисбалансированной выборки"

В качестве подопытного выступит набор данных по классификации орбиты небесного тела на основании множества измеренных параметров (числовых признаков). Ссылка на исходный датасет: https://www.kaggle.com/brsdincer/orbitclassification

### 1. Разведочный анализ данных

#### Первый взгляд

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
data = pd.read_csv("ML_Datasets/CW/orbit_source.csv")
```

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1748 entries, 0 to 1747
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   a (AU)      1748 non-null   float64
 1   e           1748 non-null   float64
 2   i (deg)     1748 non-null   float64
 3   w (deg)     1748 non-null   float64
 4   Node (deg)  1748 non-null   float64
 5   M (deg)     1748 non-null   float64
 6   q (AU)      1748 non-null   float64
 7   Q (AU)      1748 non-null   float64
 8   P (yr)      1748 non-null   float64
 9   H (mag)     1748 non-null   float64
 10  MOID (AU)   1748 non-null   float64
 11  class       1748 non-null   object 
dtypes: float64(11), object(1)
memory usage: 164.0+ KB
```

Набор данных имеет следующие колонки:

1. a (AU) - длина большей полуоси орбиты, AU - астрономические единицы;
2. e - эксцентрисцитет орбиты, безразмерная величина;
3. i (deg) - наклон орбиты в эклиптической системе координат, deg - градусы;
4. w (deg) - перигелий;
5. Node (deg) - долгота восходящего узла;
6. M (deg) - средняя аномалия в эпоху;
7. q (AU) - длина орбиты в перигелий;
8. Q (AU) - длина орбиты в афелий;
9. P (yr) - период обращения, yr - год по Юлианскому (человеческому) календарю;
10. H (mag) - абсолютная V-длина, mag - сокращание "magnitude";
11. MOID (AU) - ;
12. class - строковая метка класса небесного тела.

In [4]:

```
data.shape
```

Out[4]:

```
(1748, 12)
```

Видим, что в наборе данных отсутствуют пропуски и категориальные признаки за исключением целевого. Закодируем метки численными значениями:

In [5]:

```
from sklearn.preprocessing import LabelEncoder
```

In [6]:

```
target_column = LabelEncoder().fit_transform(data["class"])
```

In [7]:

```
data["class"] = target_column
```

### Выбор признаков для обучения

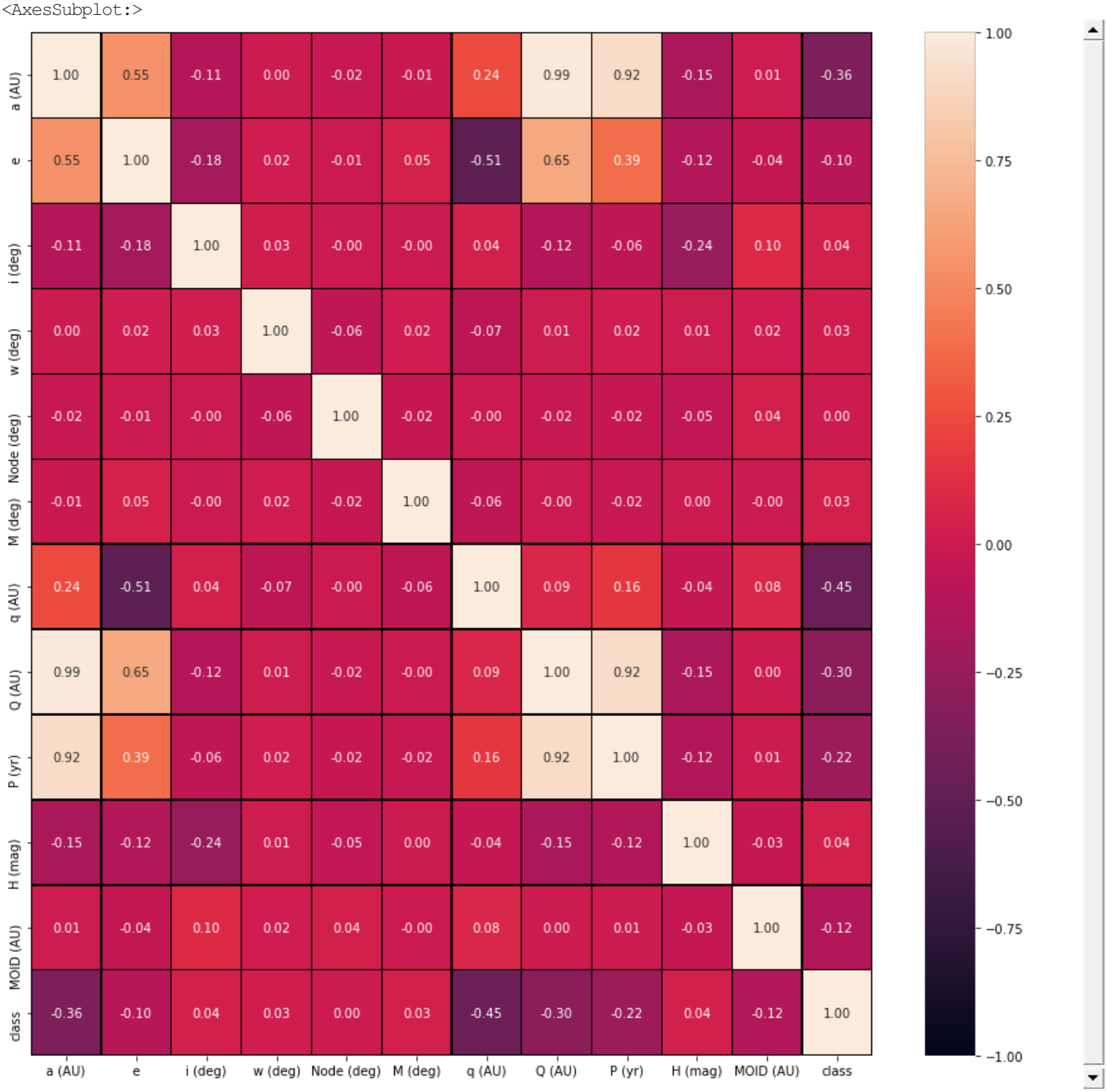
Теперь можно посмотреть на взаимосвязь признаков:

In [8]:

```
fig, ax = plt.subplots(figsize=(15, 15))
sns.heatmap(data.corr(), ax=ax, annot=True, fmt=".2f", linewidths=0.3, linecolor="black", vmin = -1, vmax = 1)
```

Out[8]:

```
<AxesSubplot:>
```




Структура данных является сложной, поскольку видим полное отсутствие сильных линейных зависимостей с целевым признаком. При этом, как говорится, есть надежда...

Имеется треугольник из сверхсильных линейных зависимостей между **a**, **Q** и **P**. Чтобы избавиться от нежелательного влияния таких строгих линейных зависимостей на качество модели, удалим признаки **Q** и **P**, поскольку они зависят от целевого меньше, нежели **a**.

In [9]:

```
columns = data.columns.to_list()
```

In [10]:

```
columns.remove("Q (AU)")
columns.remove("P (yr)")
```

In [11]:

```
data = data[columns]
```

In [12]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1748 entries, 0 to 1747
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   a (AU)      1748 non-null   float64
 1   e           1748 non-null   float64
 2   i (deg)     1748 non-null   float64
 3   w (deg)     1748 non-null   float64
 4   Node (deg)  1748 non-null   float64
 5   M (deg)     1748 non-null   float64
 6   q (AU)      1748 non-null   float64
 7   H (mag)     1748 non-null   float64
 8   MOID (AU)   1748 non-null   float64
 9   class       1748 non-null   int32  
dtypes: float64(9), int32(1)
memory usage: 129.9 KB
```

Нежелательные признаки успешно удалены.

**Масштабирование признаков**

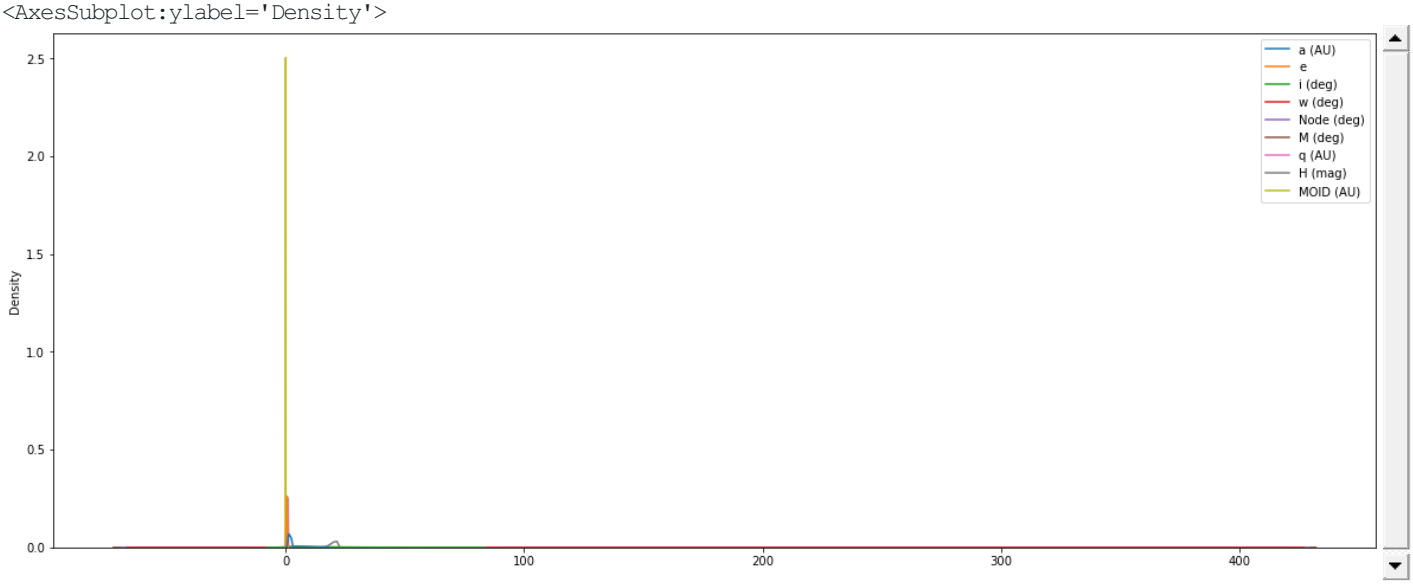
Теперь посмотрим на распределение признаков и определимся с необходимостью масштабирования:

In [13]:

```
plt.figure(figsize = (20, 8))
sns.kdeplot(data = data[columns[:-1]])
```

Out[13]:

```
<AxesSubplot:ylabel='Density'>
```




Масштабирование однозначно понадобится, поскольку мы видим, что кривая распределения многих признаков "схлопывается" до вертикальной прямой.

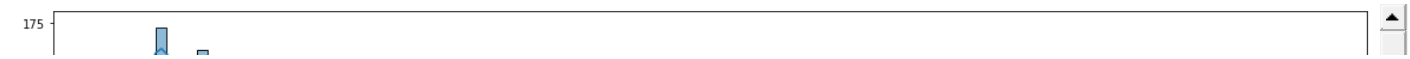
Для начала посмотрим на распределение каждого признака по отдельности, чтобы определиться со стратегией масштабирования:

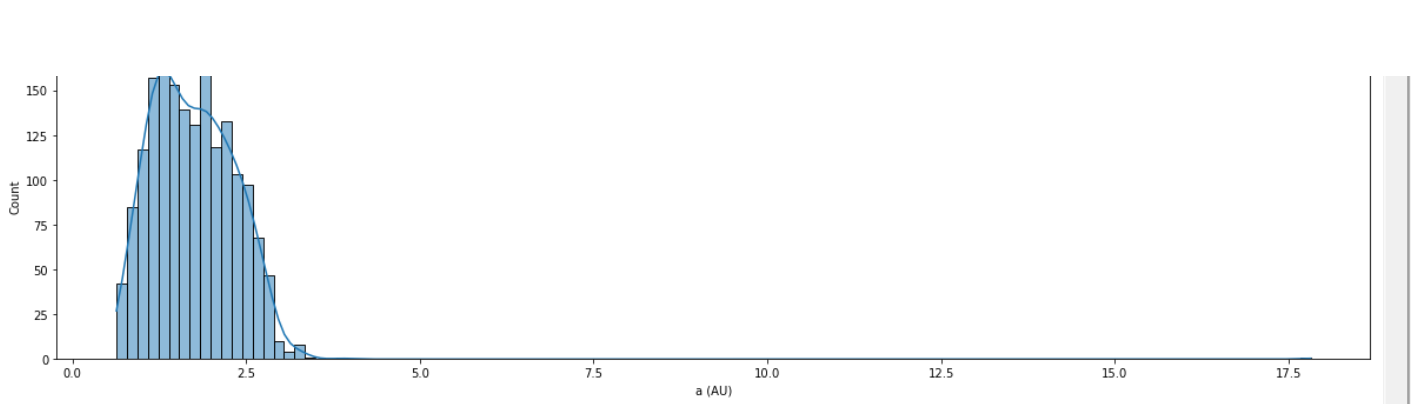
In [14]:

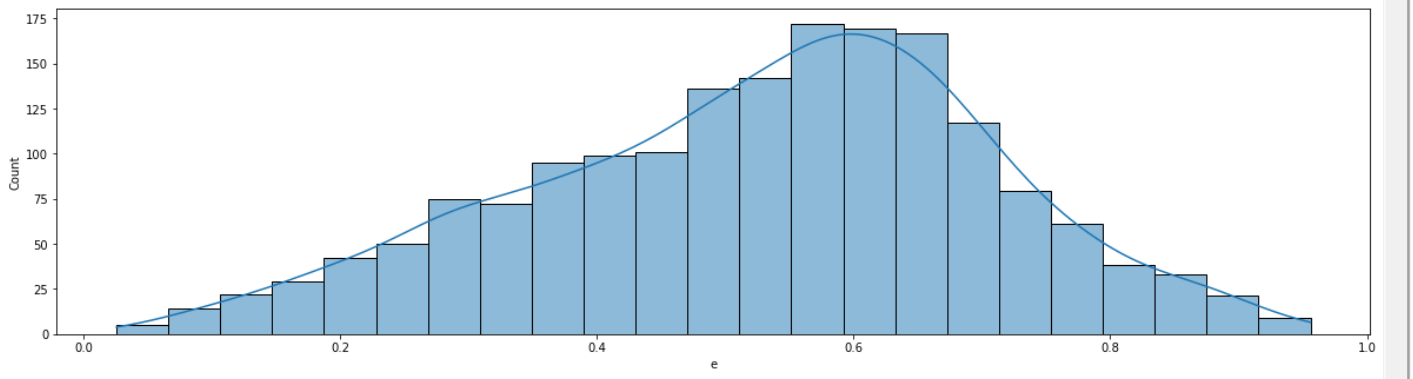
```
def ShowColumnsDistribution(data):
    columns = data.columns.to_list()
    fig, ax = plt.subplots(len(columns), 1, figsize=(20,6 * len(columns)))
    for i in range(len(columns)):
        sns.histplot(data[columns[i]], ax=ax[i], kde = True)

    return
```

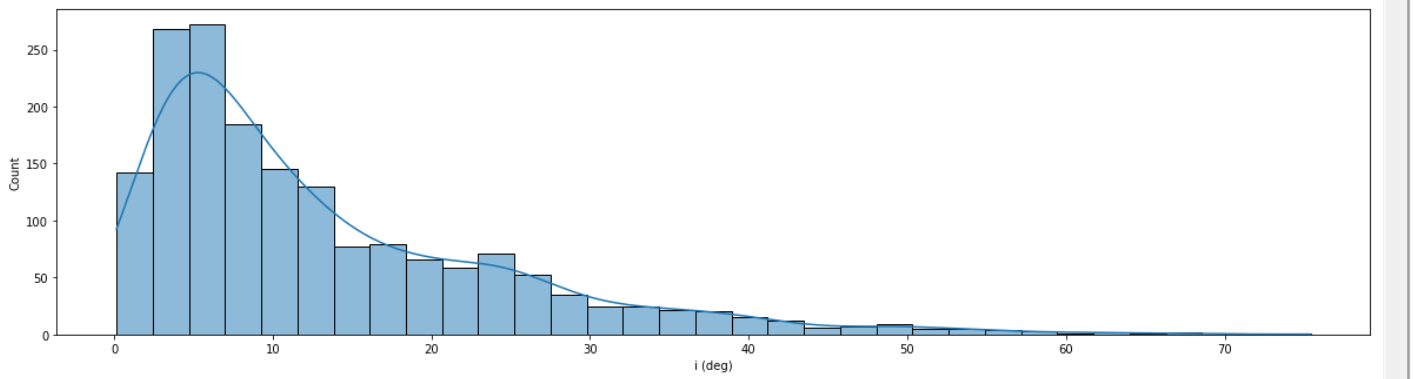
In [15]:

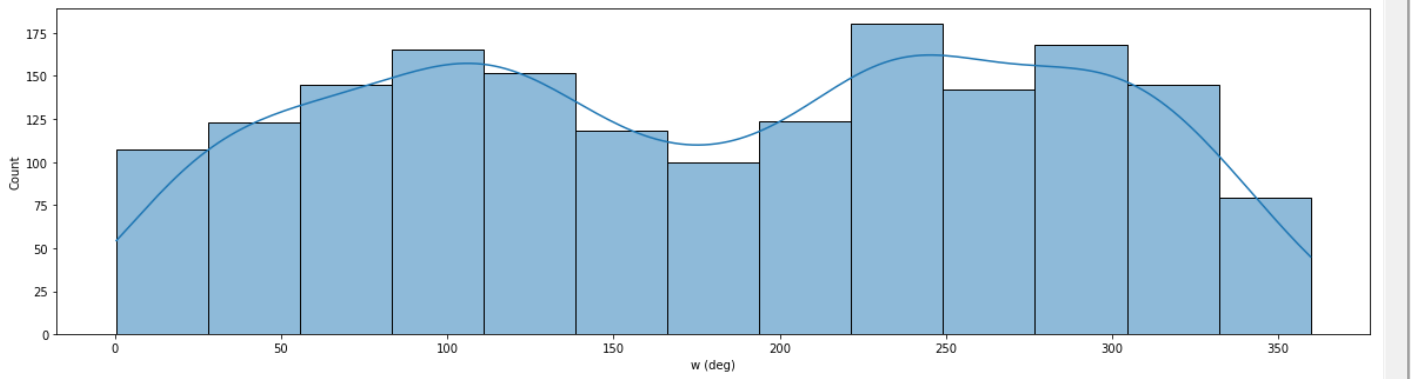
```
ShowColumnsDistribution(data[columns[:-1]])
```

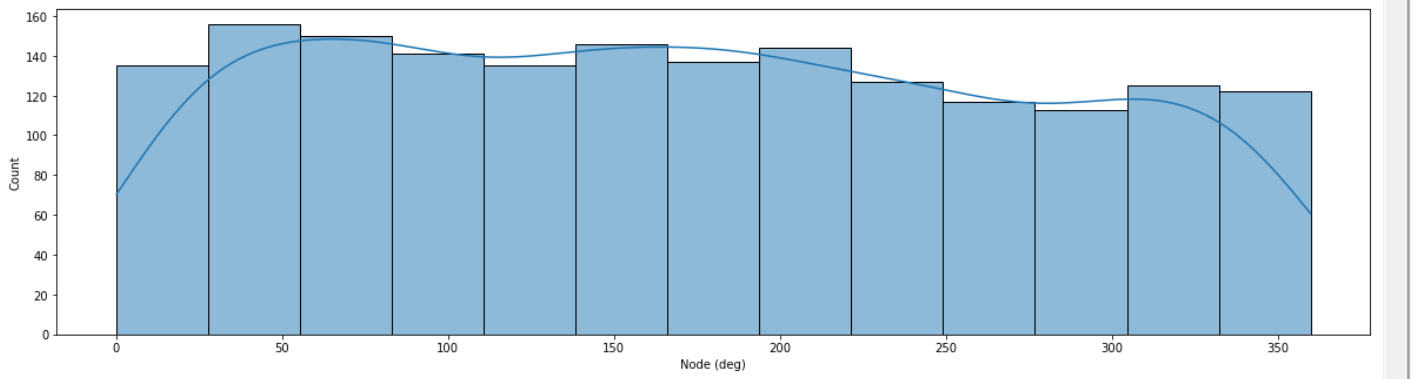


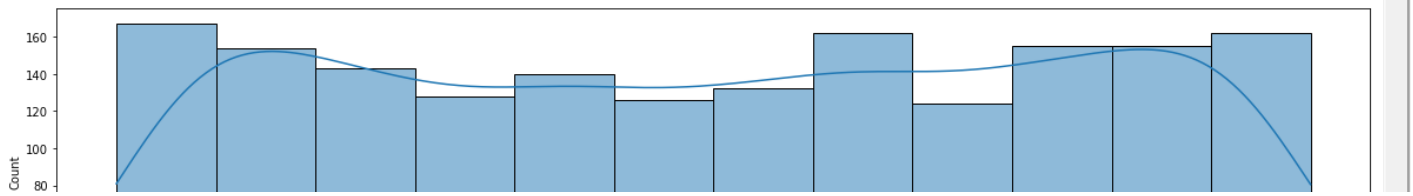

Count
a (AU)

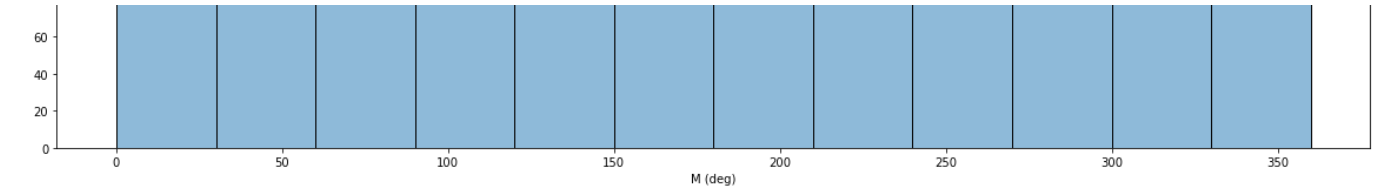

Count
e

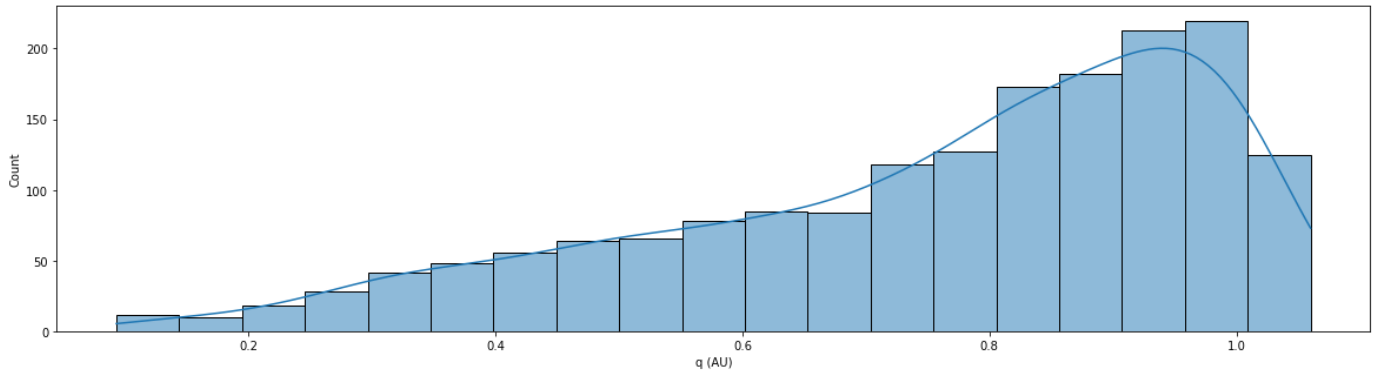

Count
i (deg)

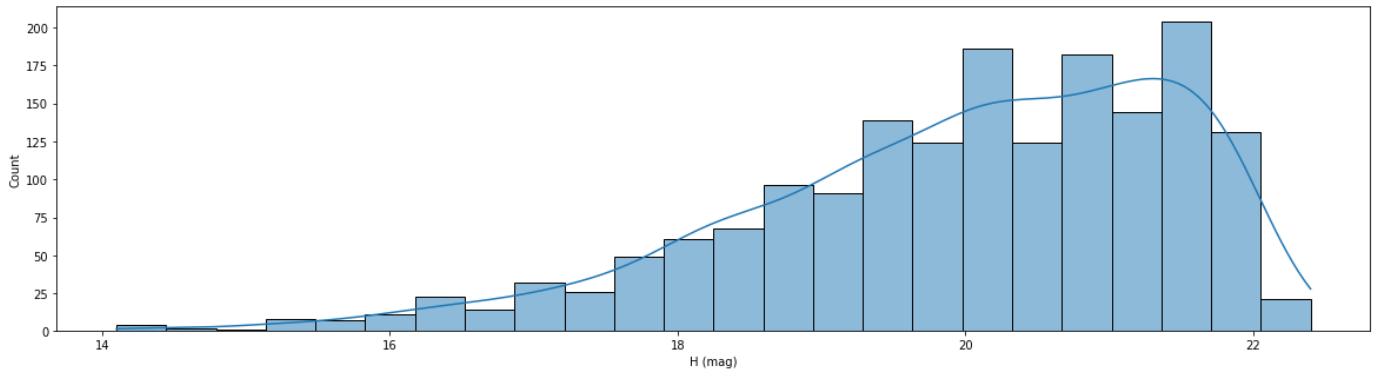

Count
w (deg)

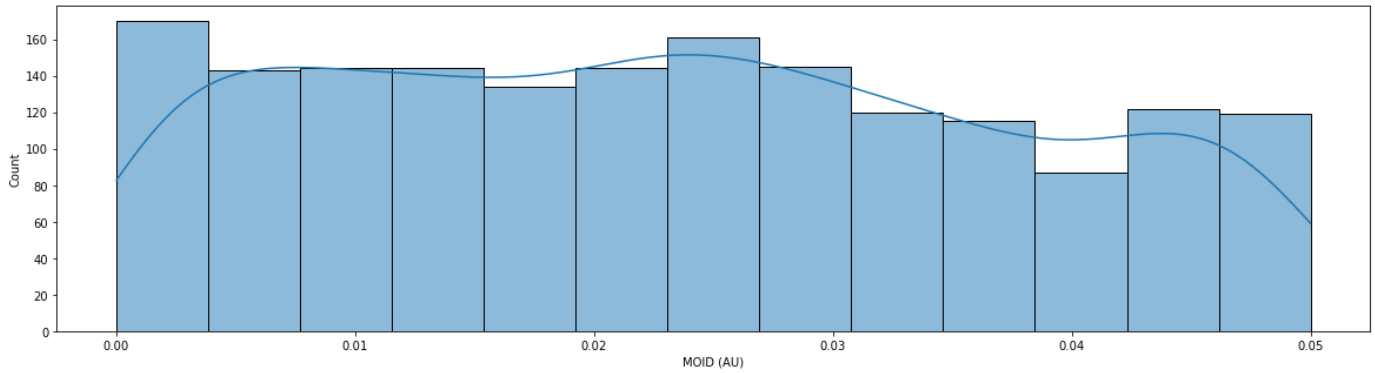

Count
Node (deg)


Count



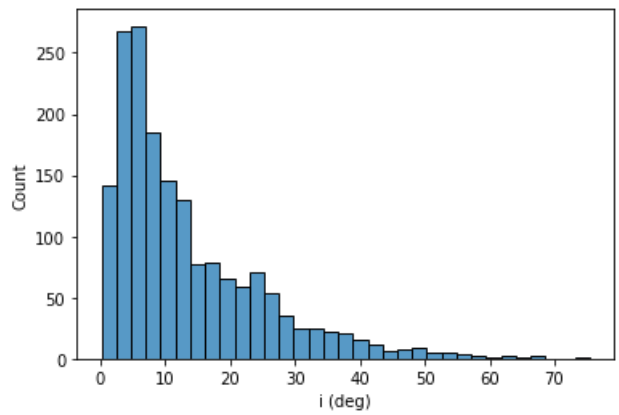




In [16]:

```
sns.histplot(data = data["i (deg)"])
```

Out[16]:

```
<AxesSubplot:xlabel='i (deg)', ylabel='Count'>
```

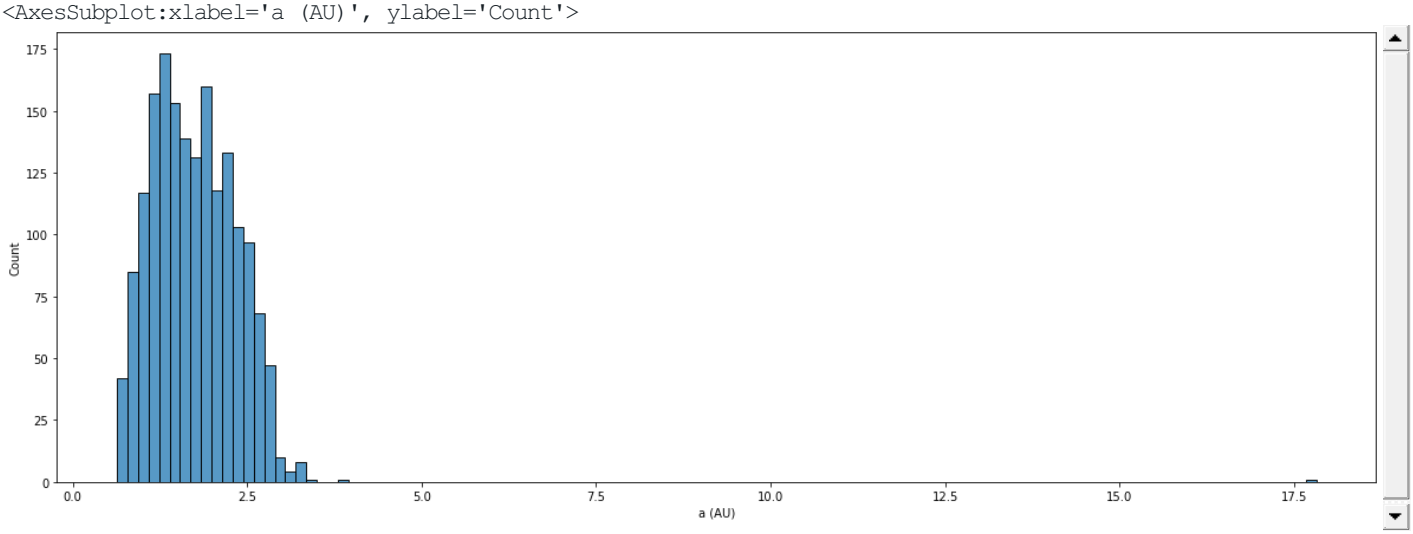



In [17]:

```
fig, ax = plt.subplots(figsize = (20, 7))
sns.histplot(data = data["a (AU)"], ax = ax)
```

Out[17]:

```
<AxesSubplot:xlabel='a (AU)', ylabel='Count'>
```




Проблемы с признаком **a**: видим одиночный выброс с аномальным значением.

In [18]:

```
data[data["a (AU)"] > 5]
```

Out[18]:

| | a (AU) | e | i (deg) | w (deg) | Node (deg) | M (deg) | q (AU) | H (mag) | MOID (AU) | class |
|---|---|---|---|---|---|---|---|---|---|---|
| **654** | 17.818679 | 0.946101 | 19.370544 | 332.891363 | 49.304199 | 83.055105 | 0.9604 | 17.2 | 0.020649 | 2 |

Заметим, что данная аномалия принадлежит к доминирующему классу, поэтому можно удалить данную запись.

In [19]:

```
data.shape
```

Out[19]:

```
(1748, 10)
```

In [20]:

```
data = data.drop(data[data["a (AU)"] > 5].index[0])
```

In [21]:

```
data.shape
```

Out[21]:

```
(1747, 10)
```

In [22]:

```
data[data["a (AU)"] > 5]
```

Out[22]:

| a (AU) | e | i (deg) | w (deg) | Node (deg) | M (deg) | q (AU) | H (mag) | MOID (AU) | class |
|---|---|---|---|---|---|---|---|---|---|

Итак, с точки зрения масштабирования видим следующее:

1. Признак **e** уже распределён в интервале [0; 1] и не требует масштабирования, что не является сюрпризом, поскольку эксцентриситет эллиптического сечения не может принимать другие значения;
2. Признак **i** имеет нормальное распределение с длинным хвостом. Применим для него RobustScaler(), чтобы слишком большое количество примеров не "смялось" в близких к нулю значениях и в то же время интервал распределения как можно меньше отличался от остальных признаков.

Можно заметить, что **i** измеряется в градусах, и с точки зрения человеческого понимания правильное решение - это, возможно, масштабировать вместе с остальными признаками, имеющими значения в градусах. Однако модели машинного обучения человеческое понимание не слишком интересно, поэтому не будем обращать внимание на единицы измерения;

1. Остальные признаки, выраженные в градусах - **w**, **Node** и **M**, - заключены в интервале [0; 360], поэтому для них применим мин-макс масштабирование;
2. Признаки **a**, **q**, **H** и **MOID** тоже заключены в строгом интервале, поэтому тоже используем мин-макс масштабирование.

In [23]:

```
from sklearn.preprocessing import RobustScaler, MinMaxScaler
```

In [25]:

```
for column in ["i (deg)"]:
    data[column] = RobustScaler().fit_transform(data[[column]])

for column in ["a (AU)", "w (deg)", "Node (deg)", "M (deg)", "q (AU)", "H (mag)", "MOID (AU)"]:
    data[column] = MinMaxScaler().fit_transform(data[[column]])
```

In [26]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1747 entries, 0 to 1747
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   a (AU)      1747 non-null   float64
 1   e           1747 non-null   float64
 2   i (deg)     1747 non-null   float64
 3   w (deg)     1747 non-null   float64
 4   Node (deg)  1747 non-null   float64
 5   M (deg)     1747 non-null   float64
 6   q (AU)      1747 non-null   float64
 7   H (mag)     1747 non-null   float64
 8   MOID (AU)   1747 non-null   float64
 9   class       1747 non-null   int32  
dtypes: float64(9), int32(1)
memory usage: 143.3 KB
```

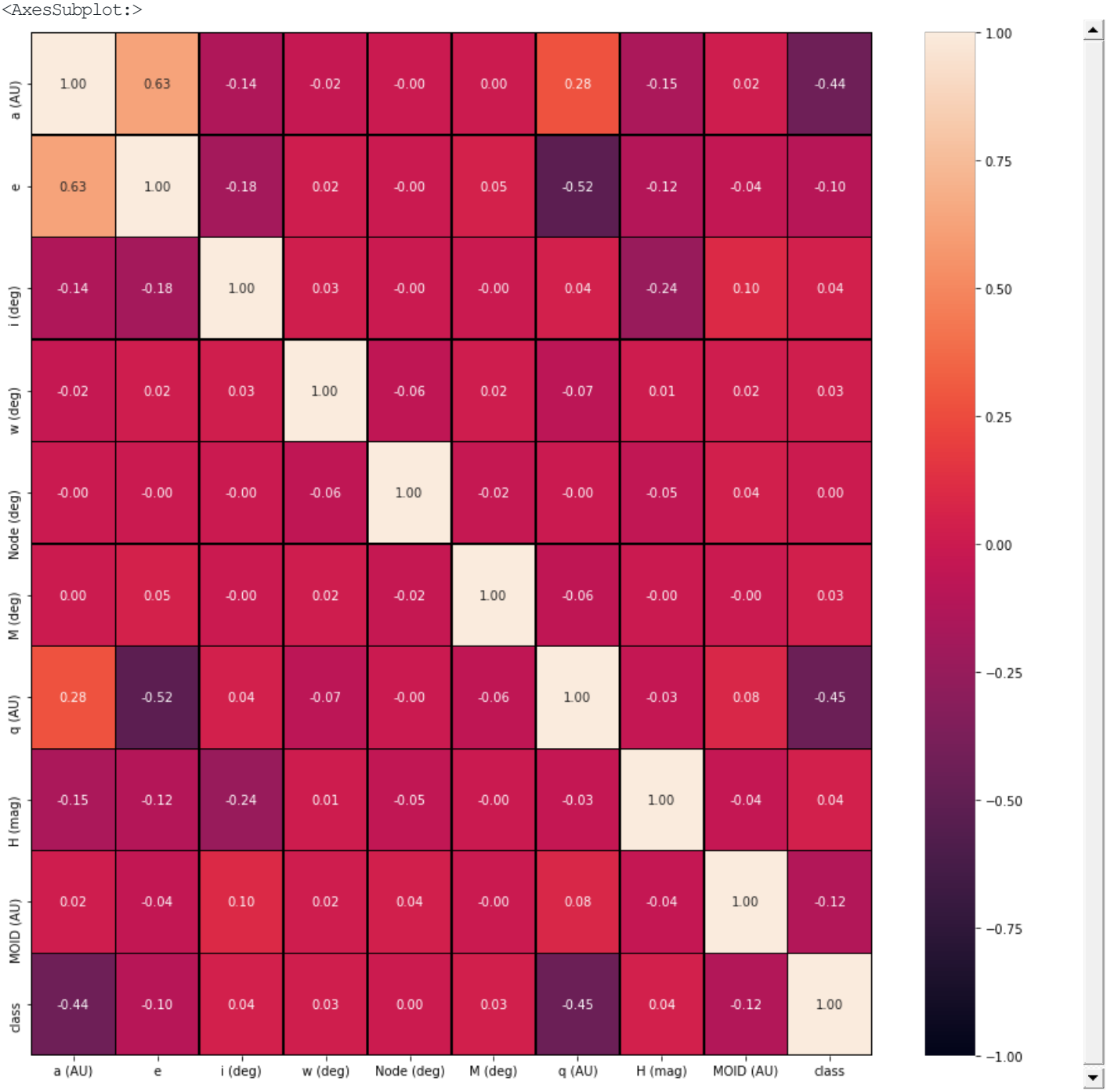
Подготовка данных завершена, итоговый вид:

In [27]:

```
fig, ax = plt.subplots(figsize=(15, 15))
sns.heatmap(data.corr(), ax=ax, annot=True, fmt=".2f", linewidths=0.3, linecolor="black", vmin = -1, vmax = 1)
```

Out[27]:

<AxesSubplot:>

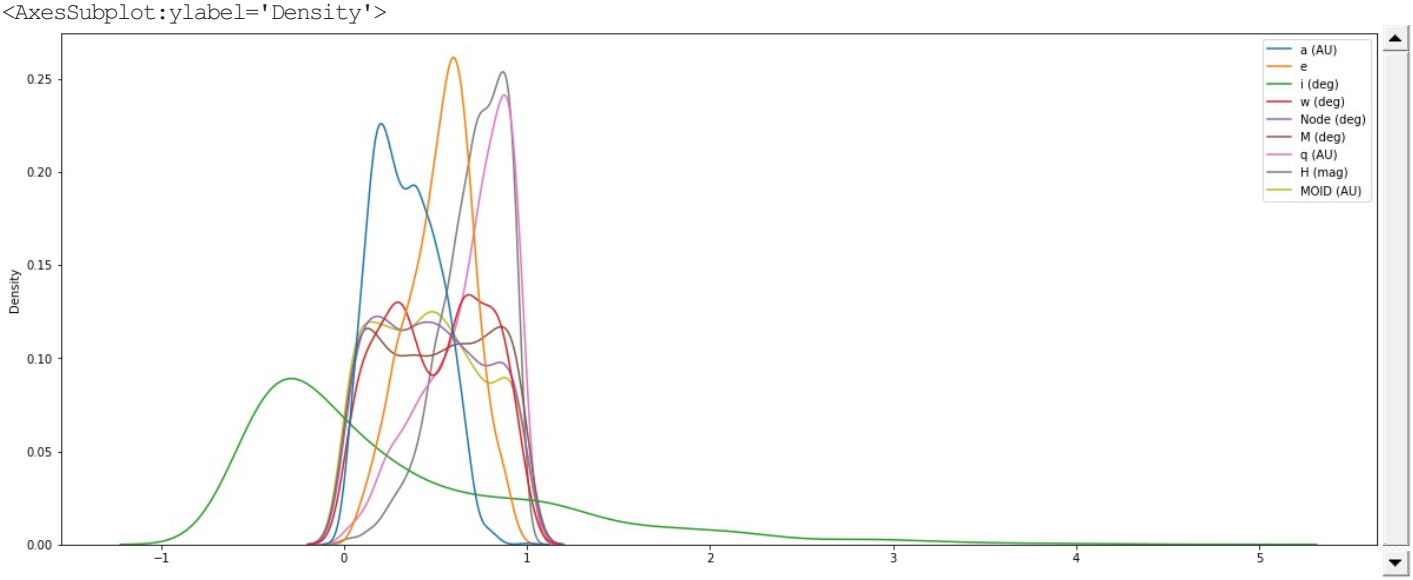



In [28]:

```
plt.figure(figsize = (20, 8))
sns.kdeplot(data = data[columns[:-1]])
```

Out[28]:

```
<AxesSubplot:ylabel='Density'>
```




In [29]:

```
data.to_csv("ML_Datasets/CW/orbit_prepared.csv")
```

### 2. Восстановление баланса выборки

In [30]:

```
data["class"].value_counts()
```

Out[30]:

```
2    1476
4     149
0      96
1      14
3       7
5       5
Name: class, dtype: int64
```

In [31]:

```
def get_labels_probability(data, feature_name):
    sum = 0
    temp_dict = {}
    for i in data[feature_name].unique():
        cnt = data[data[feature_name] == i].count()[-1]
        sum = sum + cnt
        temp_dict.update([(i, cnt)])

    for key in temp_dict:
        temp_dict[key] /= float(sum)

    #print(temp_dict)
    return temp_dict
```

In [32]:

```
get_labels_probability(data, "class")
```

Out[32]:

```
{2: 0.8448769318832284,
 4: 0.08528906697195192,
 0: 0.05495134516313681,
 1: 0.008013737836290785,
 5: 0.0028620492272467086,
 3: 0.004006868918145392}
```

Наша выборка сильно дисбалансирована. Примерно 80-90 процентов данных принадлежат одному из классов, а их здесь целых 6!

В связи с этим попробуем воспользоваться несколькими методами восстановления баланса выборки. Будем действовать следующим образом:

1. Сформируем несколько дополнительных DataFrame-ов, в которых будут содержаться более сбалансированные данные;
2. При обучении каждой модели подбор гиперпараметров будет происходить на основном наборе;
3. Для сравнения будем смотреть на результаты лучшей модели со всеми наборами данных.

Для восстановления баланса выборки существует две основных группы методов: undersampling (исключение записей, относящихся к доминирующему классу) и oversampling (искусственное внесение данных, относящихся к тем классам, для которых не хватает данных). Методы обеих групп могут оказаться полезными, однако больший прирост качества ожидается для второй группы потому, что 5, 7 и 14 образцов - это катастрофически малое значение.

Воспользуемся следующими методами восстановления баланса:

1. Random Undersampling (удаление случайных элементов до тех пор, пока не будет достигнуто желаемое соотношение);
2. Edited Nearest Neighbours (снижение числа объектов доминирующего класса на разделяющей полосе засчёт работы KNN);
3. Instance Hardness Threshold (исключение объектов, предсказание в отношении которых - самое неуверенное);
4. SMOTE (Synthetic Minority Over-Sampling Technique; интерполяция ближайших соседей для малопредставленных классов);
5. ADASYN (Adaptive Synthetic Algorithm, расширение SMOTE).

In [33]:

```
from imblearn.under_sampling import RandomUnderSampler, EditedNearestNeighbours, InstanceHardnessThreshold
from imblearn.over_sampling import SMOTE, ADASYN
```

**Undersampling**

In [34]:

```
rus = RandomUnderSampler(sampling_strategy = {2: 500}, random_state = 16)

rus_x, rus_y = rus.fit_resample(data[columns[:-1]], data[columns[-1]].astype("int32"))
data_rus = rus_x.join(rus_y)
data_rus.to_csv("ML_Datasets/CW/orbit_rus.csv")
```

In [35]:

```
data_rus["class"].value_counts()
```

Out[35]:

```
2    500
4    149
0     96
1     14
3      7
5      5
Name: class, dtype: int64
```

In [36]:

```
enn = EditedNearestNeighbours(sampling_strategy = [2, ], n_jobs = 4)

enn_x, enn_y = enn.fit_resample(data[columns[:-1]], data[columns[-1]].astype("int32"))
data_enn = enn_x.join(enn_y)
data_enn.to_csv("ML_Datasets/CW/orbit_enn.csv")
```

In [37]:

```
data_enn["class"].value_counts()
```

Out[37]:

```
2    1172
4     149
0      96
1      14
3       7
5       5
Name: class, dtype: int64
```

In [38]:

```
iht = InstanceHardnessThreshold(sampling_strategy = "majority", cv = 5, n_jobs = 8, random_state = 16)

iht_x, iht_y = iht.fit_resample(data[columns[:-1]], data[columns[-1]].astype("int32"))
data_iht = iht_x.join(iht_y)
data_iht.to_csv("ML_Datasets/CW/orbit_iht.csv")
```

In [39]:

```
data_iht["class"].value_counts()
```

Out[39]:

```
2    952
4    149
0     96
1     14
3      7
5      5
Name: class, dtype: int64
```

**Oversampling**

Чтобы не перегружать датасет искусственными данными, зададим желаемое число образцов. Сделаем его всё ещё сильно меньшим в сравнении с major-классом, но предположительно достаточным для прироста качества.

In [40]:
```
smote = SMOTE(sampling_strategy = {0 : 250, 1 : 200, 3 : 200, 4 : 300,  5 : 200}, random_state = 16, n_jobs =

smote_x, smote_y = smote.fit_resample(data[columns[:-1]], data[columns[-1]].astype("int32"))
data_smote = smote_x.join(smote_y)
data_smote.to_csv("ML_Datasets/CW/orbit_smote.csv")
```

In [41]:
```
data_smote["class"].value_counts()
```

Out[41]:
```
2    1476
4     300
0     250
5     200
3     200
1     200
Name: class, dtype: int64
```

Оставлю здесь то, что обнаружил в процессе экспериментов: ни одна другая вариация **SMOTE** не пожелала корректно работать с этим датасетом! Borderline-SMOTE не смог сгенерировать данные для minor-классов, а SVMSMOTE и KMeansSMOTE выдавали исключения.

In [42]:
```
adasyn = ADASYN(sampling_strategy = {0 : 250, 1 : 200, 3 : 200, 4 : 300,  5 : 200}, random_state = 16, n_neigh

adasyn_x, adasyn_y = adasyn.fit_resample(data[columns[:-1]], data[columns[-1]].astype("int32"))
data_adasyn = adasyn_x.join(adasyn_y)
data_adasyn.to_csv("ML_Datasets/CW/orbit_adasyn.csv")
```

In [43]:
```
data_adasyn["class"].value_counts()
```

Out[43]:
```
2    1476
4     341
0     235
1     205
3     203
5     200
Name: class, dtype: int64
```

### 3. Подготовка к процессу обучения моделей

Разобьём все выборки на обучающие и тестовые, подготовим "пайплайн" для обучения, подбора гиперпараметров и оценки качества, а также выберем метрики.

In [44]:
```
from sklearn.model_selection import train_test_split
```

In [45]:
```
x_train, x_test, y_train, y_test = train_test_split(data[columns[:-1]],
                                                    data[columns[-1]],
                                                    train_size = 0.75,
                                                    random_state = 12)
rus_x_train, rus_x_test, rus_y_train, rus_y_test = train_test_split(data_rus[columns[:-1]],
                                                                    data_rus[columns[-1]],
                                                                    train_size = 0.75,
                                                                    random_state = 12)
enn_x_train, enn_x_test, enn_y_train, enn_y_test = train_test_split(data_enn[columns[:-1]],
                                                                    data_enn[columns[-1]],
                                                                    train_size = 0.75,
                                                                    random_state = 12)
iht_x_train, iht_x_test, iht_y_train, iht_y_test = train_test_split(data_iht[columns[:-1]],
                                                                    data_iht[columns[-1]],
                                                                    train_size = 0.75,
                                                                    random_state = 12)
smote_x_train, smote_x_test, smote_y_train, smote_y_test = train_test_split(data_smote[columns[:-1]],
                                                                            data_smote[columns[-1]],
                                                                            train_size = 0.75,
                                                                            random_state = 12)
```

```
adasyn_x_train, adasyn_x_test, adasyn_y_train, adasyn_y_test = train_test_split(data_adasyn[columns[:-1]],
                                                                                 data_adasyn[columns[-1]],
                                                                                 train_size = 0.75,
                                                                                 random_state = 12)
```

Какие метрики будем использовать для качества?

1. Никак не обойтись без классической *accuracy*;
2. Будем смотреть на взвешенные и усреднённые показатели *precision* и *recall* (выборка сложная, поэтому отказываемся от F1 ввиду того, что за средним гармоническим можно не уследить за тем, что происходит с каждой из метрик);
3. Также воспользуемся ROC AUC по стратегии One-versus-Rest.

Для удобства также будем выводить матрицу ошибок.

In [46]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score, plot_confusion_matr
```

In [47]:

```
# Прячем сюда "рутинную" работу по обучению моделей
def FitPredictClassifier(model, x_train, x_test, y_train, y_test):
    solver = model
    solver.fit(x_train, y_train)
    prediction = solver.predict(x_test)
    return solver, prediction


# Прячем сюда "рутинную" работу по выводу результатов обучения
def PrintClassificationMetrics(y_test, y_predicted):
    # Результат accuracy не требует никакой обработки
    accuracy = accuracy_score(y_test, y_predicted)
    print("1. Общая точность (accuracy) =", accuracy)
    # А вот результаты по precision и recall требуют обработки, поскольку функция вернёт список значений по к
    scores_list = precision_score(y_test, y_predicted, zero_division = 0, average = None)
    print("2. Меткость (precision) по классам:", FormatMulticlassMetricsScore(scores_list, "precision"))

    scores_list = recall_score(y_test, y_predicted, average = None)
    print("3. Полнота (recall) по классам:", FormatMulticlassMetricsScore(scores_list, "recall"))

    return


# Упаковка оценки качества обучения на многоклассовой классификации в строковый формат для чтения
def FormatMulticlassMetricsScore(scores_list, metric_label):
    result_string = str()
    i = 0
    sum = 0
    for score in scores_list:
        result_string += "\nКласс {0}: {1}_score = {2}".format(i, metric_label, score)
        i = i + 1
        sum = sum + score
    result_string += "\nСреднее арифм-е: {0}_score = {1}".format(metric_label, sum / i)

    weights = np.array([data[columns[-1]].value_counts(sort = False)], dtype = float)
    for i in range(weights.shape[1]):
        weights[0, i] /= data[columns[-1]].shape[0]

    result_string += "\nСредневзвешенное: {0}_score = {1}".format(metric_label, (np.dot(weights, scores_list)
    return result_string


# Небольшая обёртка над выводом матрицы ошибок, чтобы не создавать каждый раз новую фигуру явно
def ShowConfusionMatrix(fitted_model, x_test, y_test):
    fig, ax = plt.subplots(figsize = (8, 8))
    plot_confusion_matrix(fitted_model, x_test, y_test, ax = ax)

    return
```

In [48]:

```
from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize
```

In [49]:

```
# Отрисовка ROC-кривых для каждого класса по схеме One-versus-Rest
# Основной код взят отсюда: https://stackoverflow.com/questions/45332410/roc-for-multiclass-classification
def PlotMulticlassRocAuc(base_model, x_train, x_test, y_train, y_test):
```

```
    # Предварительно превращаем метки классов в своего рода One-Hot-Encoded таблицу
    y_train = label_binarize(y_train, classes = sorted(y_train.unique()))
    y_test = label_binarize(y_test, classes = sorted(y_test.unique()))

    # Обучаем классификатор
    ovrc = OneVsRestClassifier(base_model)
    y_score = ovrc.fit(x_train, y_train).decision_function(x_test)
    n_classes = y_score.shape[1]

    # Вычисление ROC и AUC-ROC
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    # Строим кривую и считаем площадь под ней для каждого класса
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Настраиваем область отрисовки
    fig, ax = plt.subplots(figsize = (10, 10))
    ax.set_title("ROC AUC ({})".format(base_model))
    ax.set_xlim(0.0, 1.0)
    ax.set_ylim(0.0, 1.01)
    ax.set_xlabel("FPR")
    ax.set_ylabel("TPR")

    # Рисуем базовую кривую, для которой AUC-ROC = 0.5
    ax.plot([0, 1], [0, 1], 'k--')

    # Отрисовываем кривую для каждого класса
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve for class {0} (area = {1})'.format(i, roc_auc[i]))
        ax.legend(loc="lower right")

    return
```

## 4. Обучение моделей

Воспользуемся следующими моделями:

1. Логистическая регрессия (используется стратегия One-versus-Rest);
2. LinearSVC (OvR);
3. SVC (OvO);
4. Дерево решений;
5. Случайный лес;
6. Градиентный бустинг.

In [50]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from catboost import CatBoostClassifier
```

### Логистическая регрессия

На этой модели не ожидаем никаких достойных результатов, поскольку тяжело получить таковые от линейной модели на 10-мерном пространстве без выраженных линейных зависимостей.

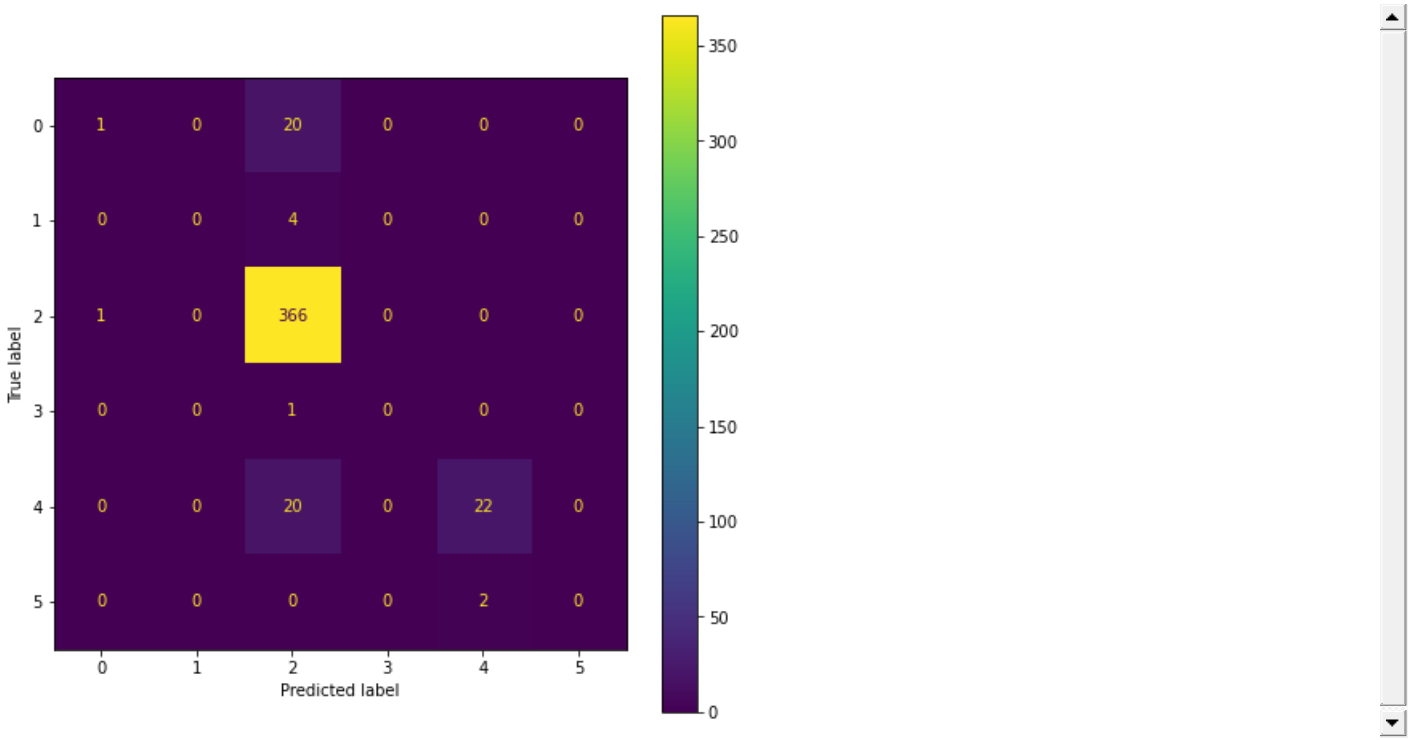
In [51]:

```
logreg, logreg_prediction = FitPredictClassifier(LogisticRegression(),
                                                 x_train,
                                                 x_test,
                                                 y_train,
                                                 y_test)
```

```
D:\Anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to conve
rge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [52]:

```
ShowConfusionMatrix(logreg, x_test, y_test)
```




In [53]:

```
PrintClassificationMetrics(y_test, logreg_prediction)
```

```
1. Общая точность (accuracy) = 0.8901601830663616
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.5
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.8905109489051095
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.9166666666666666
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.384529602595296
Средневзвешенное: precision_score = 0.8580294756252289
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.047619047619047616
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.997275204359673
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.5238095238095238
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.26145062929804075
Средневзвешенное: recall_score = 0.889866771181411
```

Попробуем исправить эту печальную ситуацию подбором гиперпараметров:

In [54]:

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from scipy.stats import expon, uniform, randint
```

In [55]:

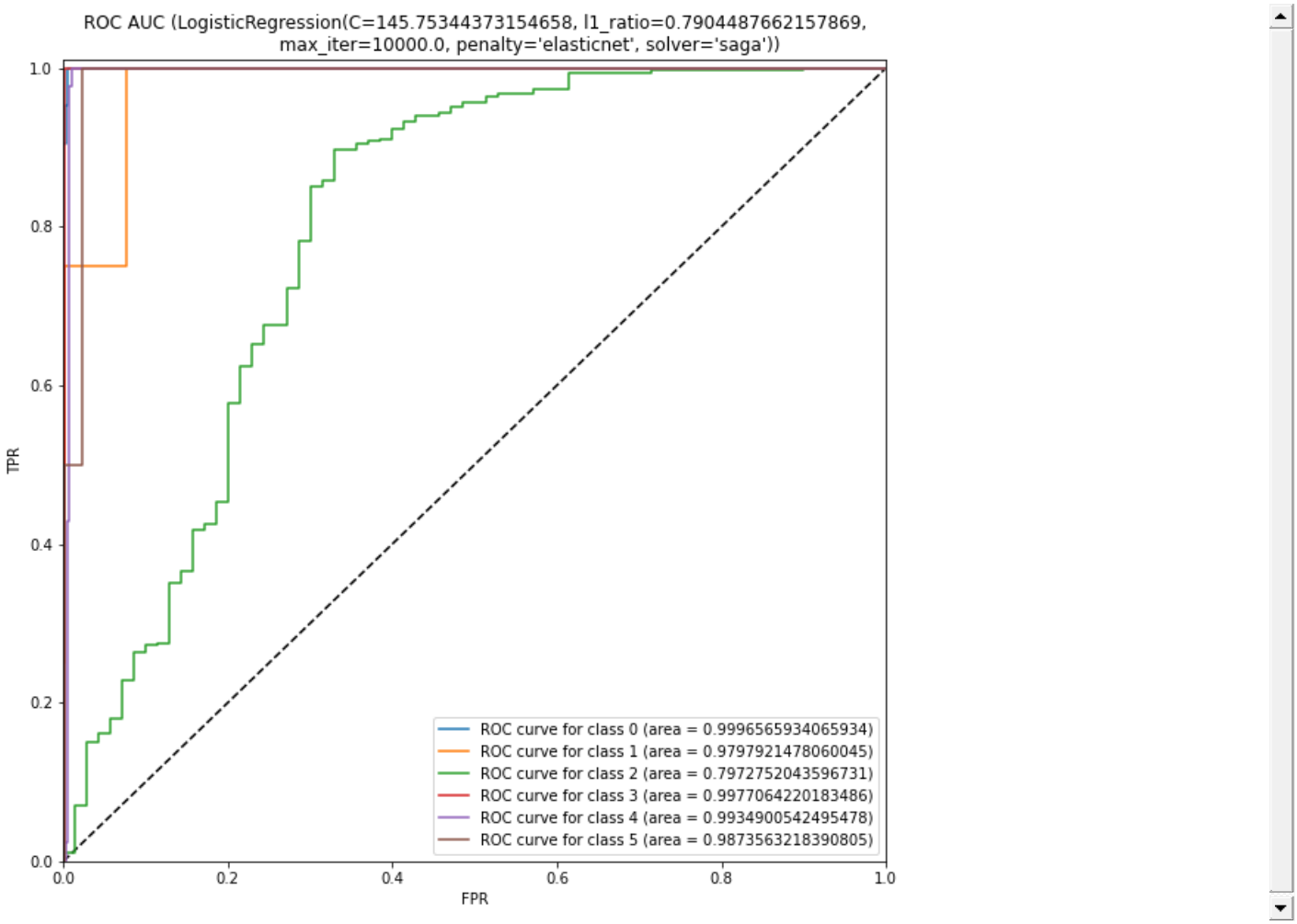
```
logreg_grid = RandomizedSearchCV(estimator = LogisticRegression(penalty = "elasticnet", solver = "saga", max_
                         param_distributions = {'C': expon(scale=100), 'l1_ratio' : uniform()},
                         n_iter = 20,
                         scoring = "precision_macro",
                         n_jobs = 8,
                         refit = True,
                         random_state = 16,
                         error_score = 0)
```

In [56]:

```
logreg_best = logreg_grid.fit(data[columns[:-1]], data[columns[-1]]).best_estimator_
```

In [57]:

```
PlotMulticlassRocAuc(logreg_best, x_train, x_test, y_train, y_test)
```




(для исходной выборки крайнее мало смысла строить кривую, поскольку слишком мало образцов для некоторых классов; кривая походит разве что на кусок бумаги с вырезом от ножниц)

In [58]:

```
logreg_best_prediction = logreg_best.predict(x_test)
PrintClassificationMetrics(y_test, logreg_best_prediction)
```

```
1. Общая точность (accuracy) = 0.9794050343249427
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.9812834224598931
Класс 3: precision_score = 1.0
Класс 4: precision_score = 0.9534883720930233
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.6557952990921527
Средневзвешенное: precision_score = 0.9693440749814899
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.9047619047619048
Класс 1: recall_score = 0.0
Класс 2: recall_score = 1.0
Класс 3: recall_score = 1.0
Класс 4: recall_score = 0.9761904761904762
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.6468253968253969
Средневзвешенное: recall_score = 0.9818600594215935
```

Результат удивительно хорош! Подбор гиперпараметров дал хорошие результаты даже для линейной модели, а катастрофически низкое значение качество по некоторым классам связано только с единичными экземплярами.

Теперь опробуем различные методы восстановления баланса выборки

#### Random Undersampling

In [59]:

```
from sklearn.base import clone
```

In [60]:

```
logreg_rus, logreg_rus_prediction = FitPredictClassifier(clone(logreg_best),
                                                         rus_x_train,
                                                         rus_x_test,
                                                         rus_y_train,
                                                         rus_y_test)
PrintClassificationMetrics(rus_y_test, logreg_rus_prediction)
```

```
1. Общая точность (accuracy) = 0.9378238341968912
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.9629629629629629
Класс 1: precision_score = 1.0
Класс 2: precision_score = 0.9523809523809523
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.8717948717948718
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.6311897978564646
Средневзвешенное: precision_score = 0.9399291162313487
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.9285714285714286
Класс 1: recall_score = 0.2
Класс 2: recall_score = 0.9836065573770492
Класс 3: recall_score = 0.0
Класс 4: recall_score = 1.0
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.5186963309914129
Средневзвешенное: recall_score = 0.9689445539962117
```

#### Edited Nearest Neighbours

In [61]:

```
logreg_enn, logreg_enn_prediction = FitPredictClassifier(clone(logreg_best),
                                                         enn_x_train,
                                                         enn_x_test,
                                                         enn_y_train,
                                                         enn_y_test)
PrintClassificationMetrics(enn_y_test, logreg_enn_prediction)
```

```
1. Общая точность (accuracy) = 0.961218836565097
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 1.0
Класс 2: precision_score = 0.9735973597359736
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.8604651162790697
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.6390104126691739
Средневзвешенное: precision_score = 0.9589233001121228
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.7777777777777778
Класс 1: recall_score = 0.25
Класс 2: recall_score = 0.9932659932659933
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.9736842105263158
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.4991213302616811
Средневзвешенное: recall_score = 0.9669755123615879
```

#### Instance Hardness Threshold

In [62]:

```
logreg_iht, logreg_iht_prediction = FitPredictClassifier(clone(logreg_best),
                                                         iht_x_train,
                                                         iht_x_test,
                                                         iht_y_train,
                                                         iht_y_test)
PrintClassificationMetrics(iht_y_test, logreg_iht_prediction)
```

```
1. Общая точность (accuracy) = 0.9444444444444444
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.8
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.9790794979079498
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.875
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.4423465829846583
Средневзвешенное: precision_score = 0.9457906919932079
3. Полнота (recall) по классам:
Класс 0: recall_score = 1.0
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.9831932773109243
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.9459459459459459
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.48818987054281177
Средневзвешенное: recall_score = 0.9663075118814369
```

**SMOTE**

In [63]:

```
logreg_smote, logreg_smote_prediction = FitPredictClassifier(clone(logreg_best),
                                                             smote_x_train,
                                                             smote_x_test,
                                                             smote_y_train,
                                                             smote_y_test)
PrintClassificationMetrics(smote_y_test, logreg_smote_prediction)
```

```
1. Общая точность (accuracy) = 0.9802130898021308
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.927536231884058
Класс 1: precision_score = 0.9642857142857143
Класс 2: precision_score = 0.9912023460410557
Класс 3: precision_score = 0.9830508474576272
Класс 4: precision_score = 1.0
Класс 5: precision_score = 0.9672131147540983
Среднее арифм-е: precision_score = 0.9722147090704256
Средневзвешенное: precision_score = 0.9881371279470188
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.9552238805970149
Класс 1: recall_score = 1.0
Класс 2: recall_score = 0.9797101449275363
Класс 3: recall_score = 1.0
Класс 4: recall_score = 0.9594594594594594
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 0.9823989141640018
Средневзвешенное: recall_score = 0.976939396628401
```

**ADASYN**

In [64]:

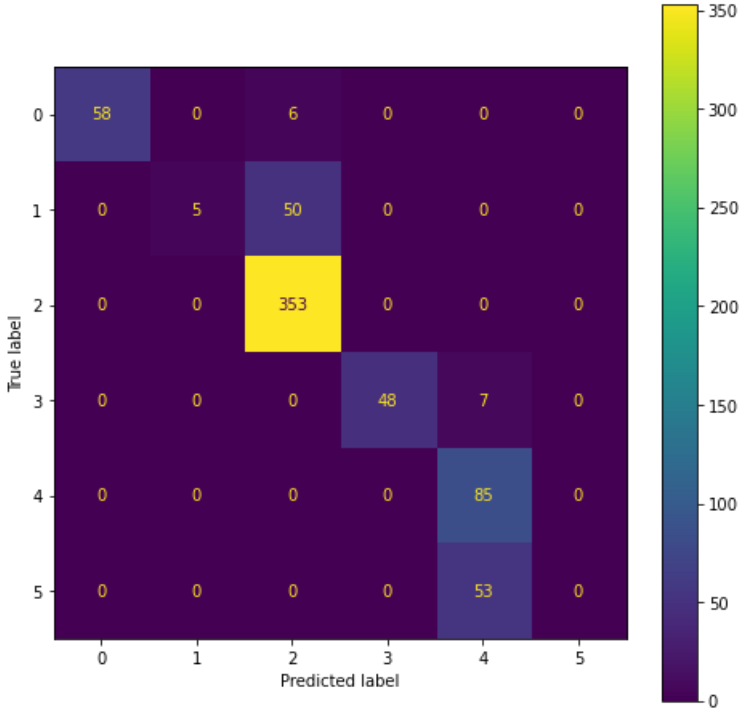
```
logreg_adasyn, logreg_adasyn_prediction = FitPredictClassifier(clone(logreg_best),
                                                               adasyn_x_train,
                                                               adasyn_x_test,
                                                               adasyn_y_train,
                                                               adasyn_y_test)
PrintClassificationMetrics(adasyn_y_test, logreg_adasyn_prediction)
```

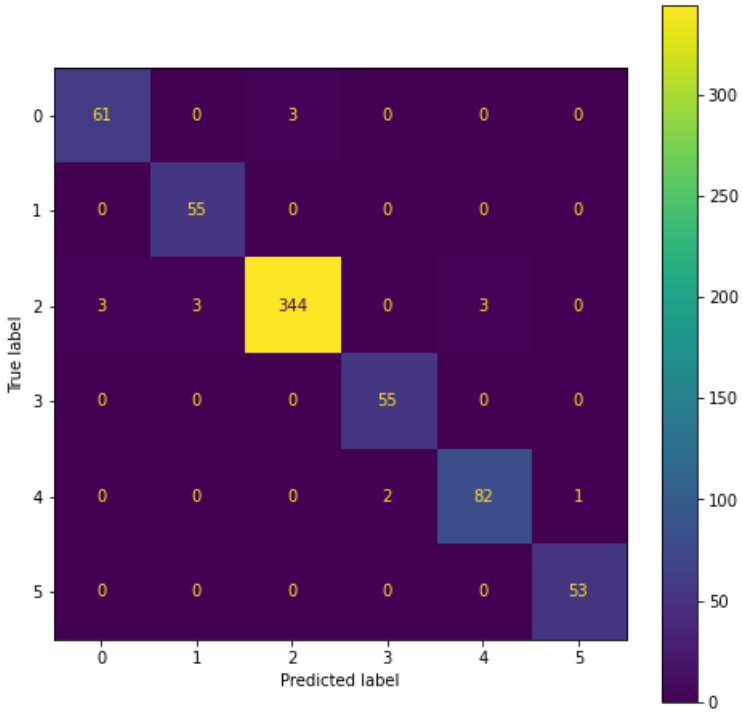
```
1. Общая точность (accuracy) = 0.9774436090225563
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.953125
Класс 1: precision_score = 0.9482758620689655
Класс 2: precision_score = 0.9913544668587896
Класс 3: precision_score = 0.9649122807017544
Класс 4: precision_score = 0.9647058823529412
Класс 5: precision_score = 0.9814814814814815
Среднее арифм-е: precision_score = 0.9673091622439887
Средневзвешенное: precision_score = 0.9865014453322535
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.953125
Класс 1: recall_score = 1.0
Класс 2: recall_score = 0.9745042492917847
Класс 3: recall_score = 1.0
Класс 4: recall_score = 0.9647058823529412
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 0.9820558552741209
Средневзвешенное: recall_score = 0.9728731816973455
```

Сравним матрицы ошибок на исходной и сбалансированной выборках:

In [65]:

```
ShowConfusionMatrix(logreg_best, adasyn_x_test, adasyn_y_test)
ShowConfusionMatrix(logreg_adasyn, adasyn_x_test, adasyn_y_test)
```



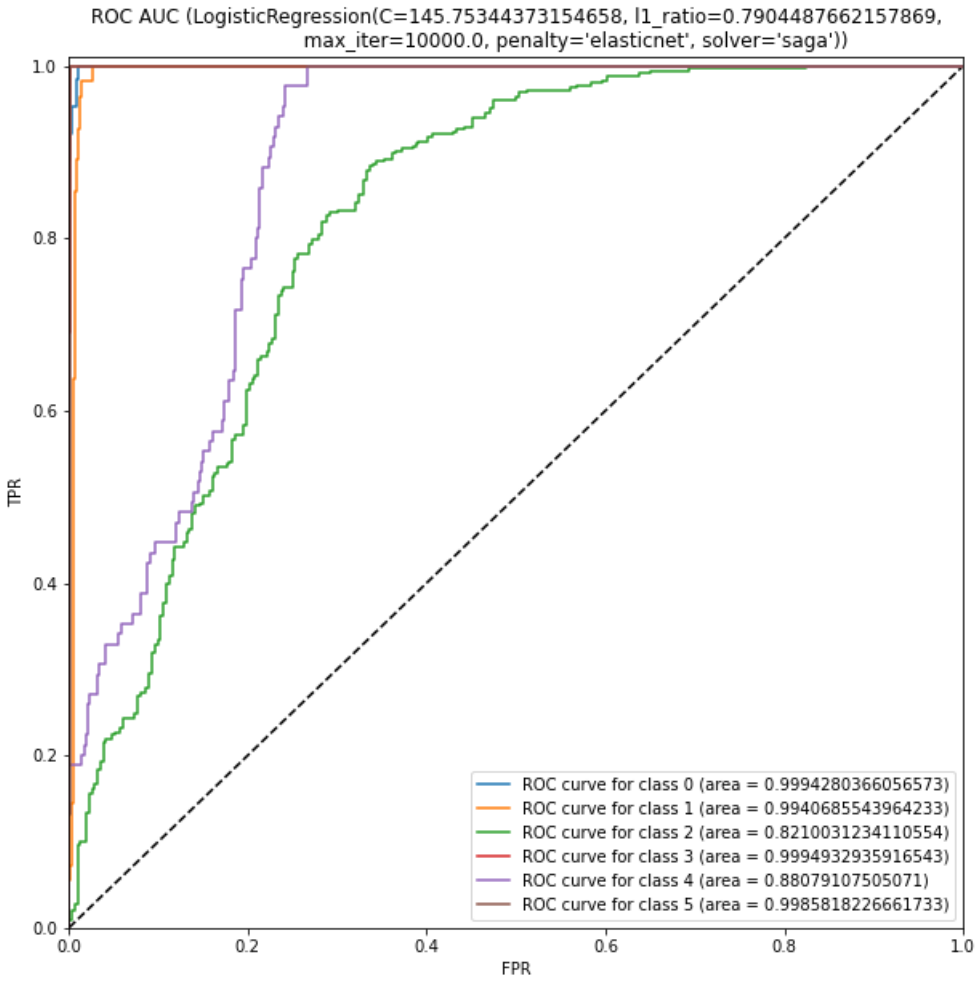

На данной матрице ошибок (здесь оптимальное решение обучено на дисбалансированной выборке, а валидация проводится на сбалансированной) можно найти подтверждение проблемам дисбаланса: многие образцы ошибочно предсказываются как major-классы.

Соответственно, после обучения модели на сбалансированных данных вышеотмеченная проблема исчезает. Результаты от oversampling-а превзошли все ожидания: датасет оказался весьма посильным.

Во время выполнения курсовой работы был проведён тест метода ADASYN без контроля количества искусственных данных (число образцов по каждому классу стало равно исходному числу образцов major-класса). В целом качество становилось лучше (хотя это можно оценить только по числу девяток после запятой), однако страдал recall у major-класса (например, для эксперимента выше recall был 0.95). Без погружения в предметную область сказать, что является оптимальным компромиссом, сложно.

In [66]:

```
PlotMulticlassRocAuc(logreg_adasyn, adasyn_x_train, adasyn_x_test, adasyn_y_train, adasyn_y_test)
```




**Машина опорных векторов**

Опробуем в деле модель LinearSVC

In [67]:

```
lsvc, lsvc_prediction = FitPredictClassifier(LinearSVC(),
                                             x_train,
                                             x_test,
                                             y_train,
                                             y_test)
PrintClassificationMetrics(y_test, lsvc_prediction)
```

```
1. Общая точность (accuracy) = 0.8924485125858124
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.5
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.8926829268292683
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.92
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.3854471544715447
Средневзвешенное: precision_score = 0.8601488265598168
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.047619047619047616
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.997275204359673
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.5476190476190477
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.2654188832662947
Средневзвешенное: recall_score = 0.8918974632521717
```

Подбор гиперпараметров:

In [68]:

```
lsvc_grid = RandomizedSearchCV(estimator = LinearSVC(max_iter = 1e+05, random_state = 16),
                               param_distributions = {'C': expon(scale=50)},
                               n_iter = 20,
                               scoring = "recall_macro",
                               n_jobs = 8,
```

```
                        refit = True,
                        random_state = 16,
                        error_score = 0)
```

In [69]:

```
lsvc_best = lsvc_grid.fit(data[columns[:-1]], data[columns[-1]]).best_estimator_
```

In [70]:

```
lsvc_best_prediction = lsvc_best.predict(x_test)
PrintClassificationMetrics(y_test, lsvc_best_prediction)
```

```
1. Общая точность (accuracy) = 0.9336384439359268
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.9338422391857506
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.9142857142857143
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.4746879922452441
Средневзвешенное: precision_score = 0.921911686586571
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.42857142857142855
Класс 1: recall_score = 0.0
Класс 2: recall_score = 1.0
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.7619047619047619
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.3650793650793651
Средневзвешенное: recall_score = 0.93340965464606
```

Результат уже не такой однозначный, как при использовании логистической регрессии. Попробуем использовать сбалансированные выборки:

#### Random Undersampling

In [71]:

```
lsvc_rus, lsvc_rus_prediction = FitPredictClassifier(clone(lsvc_best),
                                                     rus_x_train,
                                                     rus_x_test,
                                                     rus_y_train,
                                                     rus_y_test)
PrintClassificationMetrics(rus_y_test, lsvc_rus_prediction)
```

```
1. Общая точность (accuracy) = 0.8808290155440415
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.9523809523809523
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.8872180451127819
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.8205128205128205
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.4433519696677591
Средневзвешенное: precision_score = 0.8719054472074687
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.7142857142857143
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.9672131147540983
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.9411764705882353
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.43711254993800797
Средневзвешенное: recall_score = 0.9366990727339009
```

#### Edited Nearest Neighbours

In [72]:

```
lsvc_enn, lsvc_enn_prediction = FitPredictClassifier(clone(lsvc_best),
                                                     enn_x_train,
                                                     enn_x_test,
                                                     enn_y_train,
                                                     enn_y_test)
PrintClassificationMetrics(enn_y_test, lsvc_enn_prediction)
```

```
1. Общая точность (accuracy) = 0.9362880886426593
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 1.0
Класс 2: precision_score = 0.9426751592356688
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.8648648648648649
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.6345900040167556
Средневзвешенное: precision_score = 0.9331730966781409
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.5
Класс 1: recall_score = 0.25
Класс 2: recall_score = 0.9966329966329966
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.8421052631578947
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.4314563766318152
Средневзвешенное: recall_score = 0.9433337076364221
```

#### Instance Hardness Threshold

In [73]:

```
lsvc_iht, lsvc_iht_prediction = FitPredictClassifier(clone(lsvc_best),
                                                     iht_x_train,
                                                     iht_x_test,
                                                     iht_y_train,
                                                     iht_y_test)
PrintClassificationMetrics(iht_y_test, lsvc_iht_prediction)
```

```
1. Общая точность (accuracy) = 0.9411764705882353
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.9
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.9556451612903226
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.868421052631579
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.45401103565365025
Средневзвешенное: precision_score = 0.9309255838045916
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.9
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.9957983193277311
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.8918918918918919
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.46461503520327047
Средневзвешенное: recall_score = 0.9668518667542205
```

#### SMOTE

In [74]:

```
lsvc_smote, lsvc_smote_prediction = FitPredictClassifier(clone(lsvc_best),
                                                         smote_x_train,
                                                         smote_x_test,
                                                         smote_y_train,
                                                         smote_y_test)
PrintClassificationMetrics(smote_y_test, lsvc_smote_prediction)
```

```
1. Общая точность (accuracy) = 0.9147640791476408
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.9814814814814815
Класс 1: precision_score = 0.9152542372881356
Класс 2: precision_score = 0.8766066838046273
Класс 3: precision_score = 0.9830508474576272
Класс 4: precision_score = 1.0
Класс 5: precision_score = 0.9833333333333333
Среднее арифм-е: precision_score = 0.9566210972275342
Средневзвешенное: precision_score = 0.893935471916861
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.7910447761194029
Класс 1: recall_score = 1.0
Класс 2: recall_score = 0.9884057971014493
Класс 3: recall_score = 1.0
Класс 4: recall_score = 0.4864864864864865
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 0.8776561766178897
Средневзвешенное: recall_score = 0.9349248663512814
```

**ADASYN**

In [75]:

```
lsvc_adasyn, lsvc_adasyn_prediction = FitPredictClassifier(clone(lsvc_best),
                                                           adasyn_x_train,
                                                           adasyn_x_test,
                                                           adasyn_y_train,
                                                           adasyn_y_test)
PrintClassificationMetrics(adasyn_y_test, lsvc_adasyn_prediction)
```

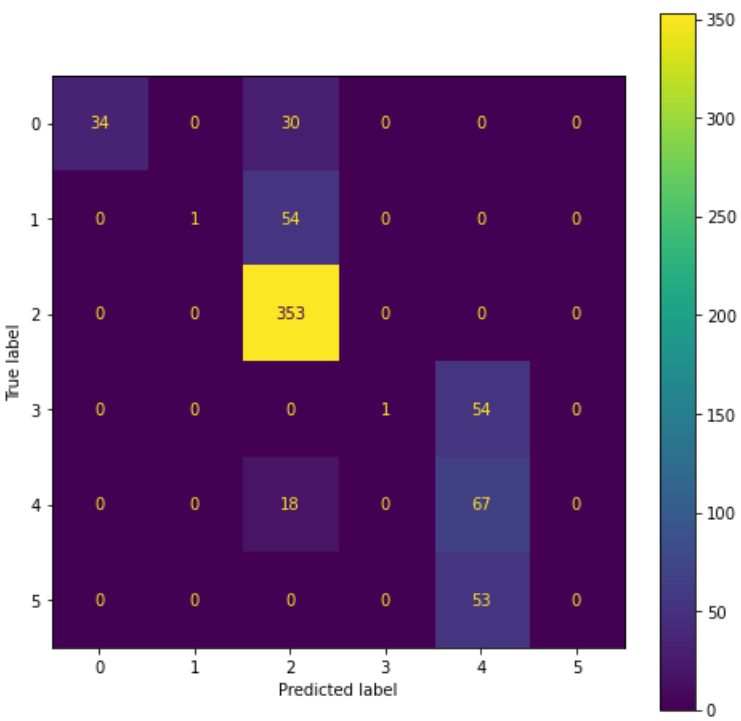
```
1. Общая точность (accuracy) = 0.9172932330827067
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 0.9137931034482759
Класс 2: precision_score = 0.8920308483290489
Класс 3: precision_score = 0.9649122807017544
Класс 4: precision_score = 0.9482758620689655
Класс 5: precision_score = 0.9464285714285714
Среднее арифм-е: precision_score = 0.9442401109961026
Средневзвешенное: precision_score = 0.9033830955078896
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.734375
Класс 1: recall_score = 0.9636363636363636
Класс 2: recall_score = 0.9830028328611898
Класс 3: recall_score = 1.0
Класс 4: recall_score = 0.6470588235294118
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 0.8880121700044943
Средневзвешенное: recall_score = 0.9406496022323455
```

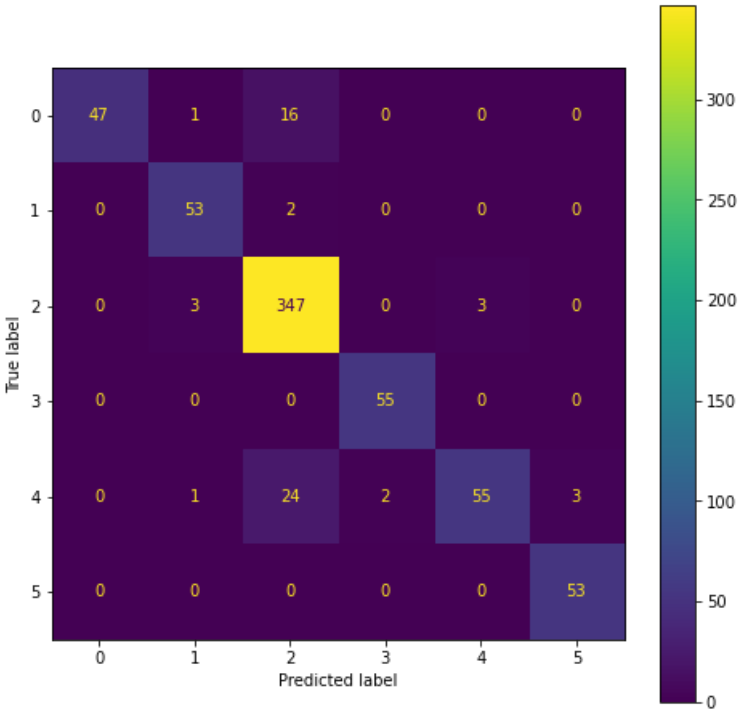
Показатели качества повторяют ситуацию с логистической регрессией: методы undersampling-а не имеют однозначного влияния на метрики (где-то станет лучше, где-то хуже - и всё это незначительно с точки зрения статистики), в то время как oversampling даёт понять, что модель "успешно" справляется со всеми классами. Условность успеха объясняется следующим:

1. Так или иначе, мы добавляем данные недостающих классов исключительно на основе уже имеющихся знаний, поэтому мы просто даём больше образцов для разгадки ("закрепляем успех"). С другой стороны, если знать что-то о предметно области, то это необязательно плохо;
2. Вдобавок к искусственному характеру данных прибавляются катастрофически малые количества образцов некоторых классов. Ни одна статистика не способна дать что-то устойчивое на выборке объёмом 5 или 7 (да и даже 14), поэтому скорее всего новые данные формируются слишком осторожно и в целом они далеки от истины;
3. Можно также заметить одну мелкую деталь: показатели качества снижаются для major-класса.

In [76]:

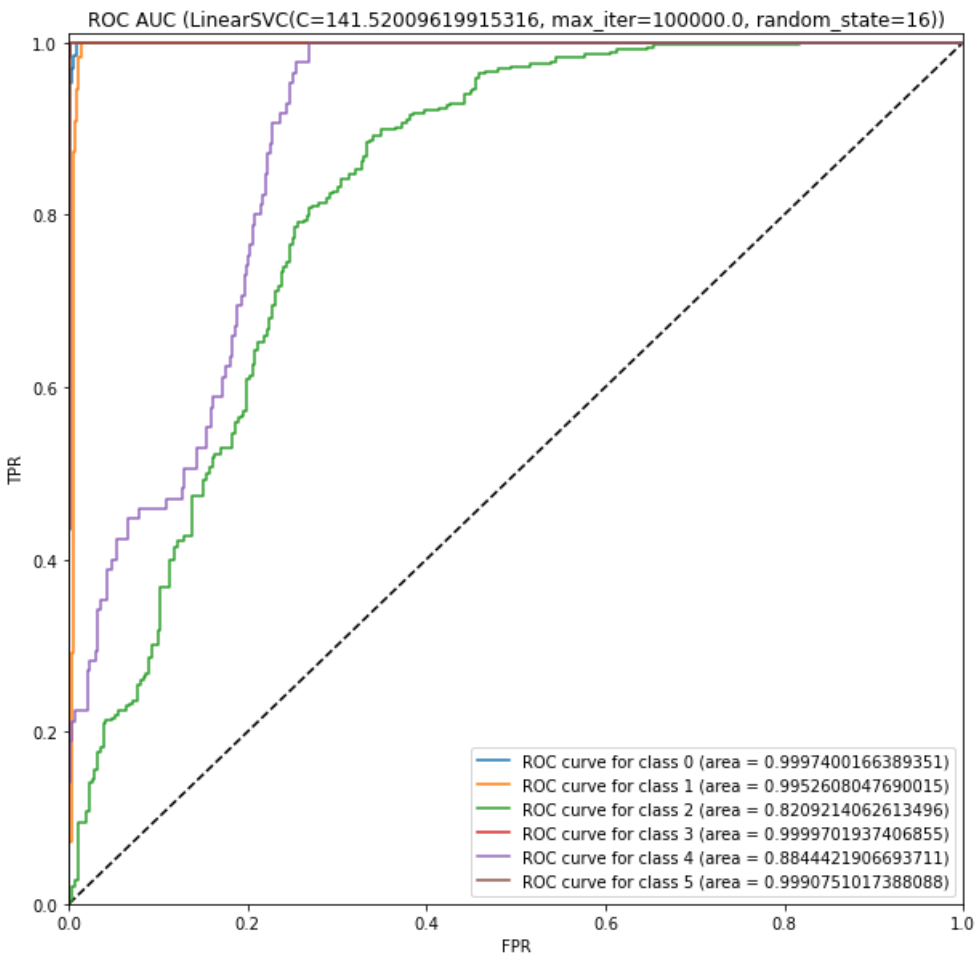
```
ShowConfusionMatrix(lsvc_best, adasyn_x_test, adasyn_y_test)
ShowConfusionMatrix(lsvc_adasyn, adasyn_x_test, adasyn_y_test)
```




In [77]:

```
PlotMulticlassRocAuc(lsvc_adasyn, adasyn_x_train, adasyn_x_test, adasyn_y_train, adasyn_y_test)
```




Теперь можно перейти к модели SVC. Возьмём радиально-базисное ядро и проделаем аналогичные операции:

In [78]:

```
svc_rbf, svc_rbf_prediction = FitPredictClassifier(SVC(random_state = 16),
                                                   x_train,
                                                   x_test,
                                                   y_train,
                                                   y_test)
PrintClassificationMetrics(y_test, svc_rbf_prediction)
```

```
1. Общая точность (accuracy) = 0.8901601830663616
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.0
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.8924205378973105
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.8571428571428571
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.29159389917336126
Средневзвешенное: precision_score = 0.8270904405556474
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.0
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.9945504087193461
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.5714285714285714
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.2609964966913196
Средневзвешенное: recall_score = 0.8890093076202702
```

Подбор гиперпараметров:

In [79]:

```
svc_rbf_grid = RandomizedSearchCV(estimator = SVC(random_state = 16),
                          param_distributions = {'C': expon(scale=50), 'gamma' : uniform(1e-04, 2)},
                          n_iter = 20,
                          scoring = "recall_macro",
                          n_jobs = 8,
                          refit = True,
                          random_state = 16,
                          error_score = 0)
```

In [80]:

```
svc_rbf_best = svc_rbf_grid.fit(data[columns[:-1]], data[columns[-1]]).best_estimator_
```

In [81]:

```
svc_rbf_best_prediction = svc_rbf_best.predict(x_test)
PrintClassificationMetrics(y_test, svc_rbf_best_prediction)
```

```
1. Общая точность (accuracy) = 0.9954233409610984
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.9523809523809523
Класс 1: precision_score = 1.0
Класс 2: precision_score = 0.997275204359673
Класс 3: precision_score = 1.0
Класс 4: precision_score = 1.0
Класс 5: precision_score = 1.0
Среднее арифм-е: precision_score = 0.9916093594567709
Средневзвешенное: precision_score = 0.9950811522973376
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.9523809523809523
Класс 1: recall_score = 1.0
Класс 2: recall_score = 0.997275204359673
Класс 3: recall_score = 1.0
Класс 4: recall_score = 1.0
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 0.9916093594567709
Средневзвешенное: recall_score = 0.9950811522973376
```

Что ж, здесь никакой сэмплинг не требуется...Но всё же проверим работу на ADASYN (для получения более устойчивого результата):

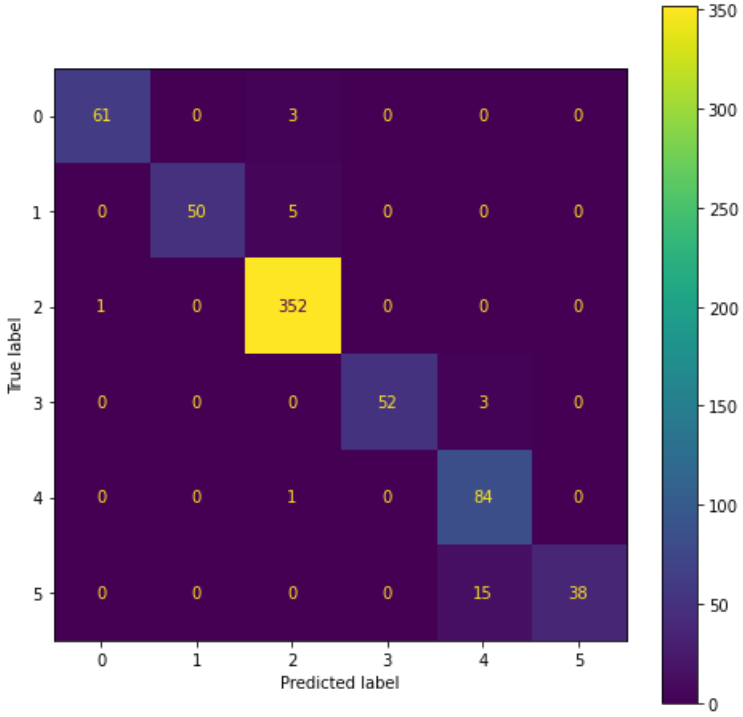
In [82]:

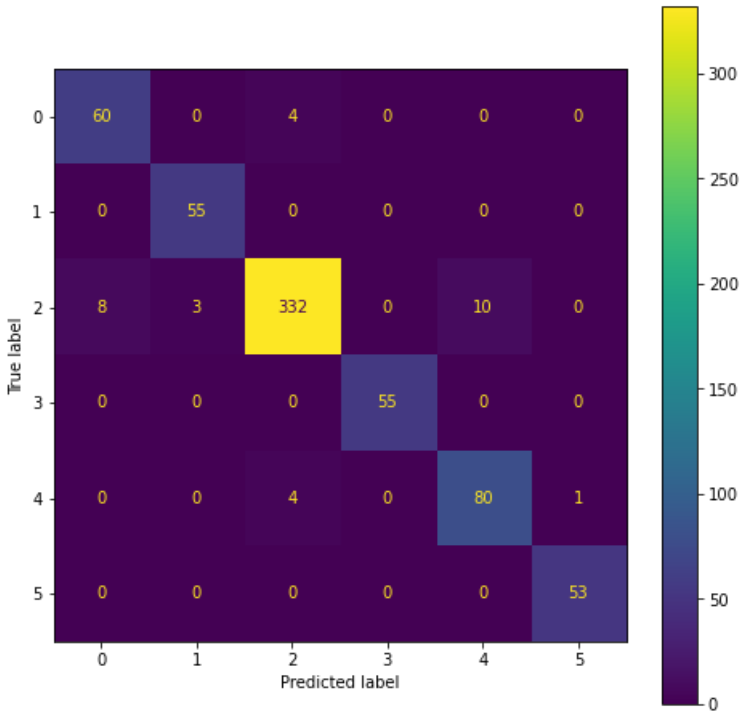
```
svc_rbf_adasyn, svc_rbf_adasyn_prediction = FitPredictClassifier(clone(svc_rbf_best),
                                                                 adasyn_x_train,
                                                                 adasyn_x_test,
                                                                 adasyn_y_train,
                                                                 adasyn_y_test)
PrintClassificationMetrics(adasyn_y_test, svc_rbf_adasyn_prediction)
```

```
1. Общая точность (accuracy) = 0.9548872180451128
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.8823529411764706
Класс 1: precision_score = 0.9482758620689655
Класс 2: precision_score = 0.9764705882352941
Класс 3: precision_score = 1.0
Класс 4: precision_score = 0.8888888888888888
Класс 5: precision_score = 0.9814814814814815
Среднее арифм-е: precision_score = 0.9462449603085168
Средневзвешенное: precision_score = 0.9637116110526918
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.9375
Класс 1: recall_score = 1.0
Класс 2: recall_score = 0.9405099150141643
Класс 3: recall_score = 1.0
Класс 4: recall_score = 0.9411764705882353
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 0.9698643976004
Средневзвешенное: recall_score = 0.9412867365074721
```

In [83]:

```
ShowConfusionMatrix(svc_rbf_best, adasyn_x_test, adasyn_y_test)
ShowConfusionMatrix(svc_rbf_adasyn, adasyn_x_test, adasyn_y_test)
```




Поэкспериментируем с машиной опорных векторов ещё. Заменим ядро на сигмоидальное:

In [84]:

```
svc_sigm, svc_sigm_prediction = FitPredictClassifier(SVC(kernel = "sigmoid", random_state = 16),
                                                     x_train,
                                                     x_test,
                                                     y_train,
                                                     y_test)
PrintClassificationMetrics(y_test, svc_sigm_prediction)
```

```
1. Общая точность (accuracy) = 0.7597254004576659
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.0
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.8341708542713567
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.0
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.13902847571189278
Средневзвешенное: precision_score = 0.7047717120231956
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.0
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.9046321525885559
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.0
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.15077202543142598
Средневзвешенное: recall_score = 0.7643028375619396
```

Подбор гиперпараметров:

In [85]:

```
svc_sigm_grid = RandomizedSearchCV(estimator = SVC(kernel = "sigmoid", random_state = 16),
                           param_distributions = {'C': expon(scale=50), 'gamma' : uniform(1e-04, 2), 'coef0' :
                           n_iter = 20,
                           scoring = "recall_macro",
                           n_jobs = 8,
                           refit = True,
                           random_state = 16,
                           error_score = 0)
```

In [86]:

```
svc_sigm_best = svc_sigm_grid.fit(data[columns[:-1]], data[columns[-1]]).best_estimator_
```

In [87]:

```
svc_sigm_best_prediction = svc_sigm_best.predict(x_test)
PrintClassificationMetrics(y_test, svc_sigm_best_prediction)
```

```
1. Общая точность (accuracy) = 0.8123569794050344
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.0625
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.8733850129198967
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.5517241379310345
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.2479348584751552
Средневзвешенное: precision_score = 0.7883933460912946
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.047619047619047616
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.9209809264305178
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.38095238095238093
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.2249253925003244
Средневзвешенное: recall_score = 0.8132233432998155
```

In [88]:

```
svc_sigm_adasyn, svc_sigm_adasyn_prediction = FitPredictClassifier(clone(svc_sigm_best),
                                                    adasyn_x_train,
                                                    adasyn_x_test,
                                                    adasyn_y_train,
                                                    adasyn_y_test)
PrintClassificationMetrics(adasyn_y_test, svc_sigm_adasyn_prediction)
```

```
1. Общая точность (accuracy) = 0.49473684210526314
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.4444444444444444
Класс 1: precision_score = 0.15625
Класс 2: precision_score = 0.6169154228855721
Класс 3: precision_score = 0.45454545454545453
Класс 4: precision_score = 0.3055555555555556
Класс 5: precision_score = 0.5
Среднее арифм-е: precision_score = 0.4129518129051711
Средневзвешенное: precision_score = 0.5762054532371879
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.1875
Класс 1: recall_score = 0.2727272727272727
Класс 2: recall_score = 0.7025495750708215
Класс 3: recall_score = 0.36363636363636365
Класс 4: recall_score = 0.25882352941176473
Класс 5: recall_score = 0.22641509433962265
Среднее арифм-е: recall_score = 0.3352753058643075
Средневзвешенное: recall_score = 0.6302367432869033
```

Проделаем то же самое для полиномиального ядра:

In [89]:

```
svc_poly, svc_poly_prediction = FitPredictClassifier(SVC(kernel = "poly", random_state = 16),
                                                     x_train,
                                                     x_test,
                                                     y_train,
                                                     y_test)
PrintClassificationMetrics(y_test, svc_poly_prediction)
```

```
1. Общая точность (accuracy) = 0.9382151029748284
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.7
Класс 1: precision_score = 0.0
Класс 2: precision_score = 0.9622641509433962
Класс 3: precision_score = 0.0
Класс 4: precision_score = 0.8666666666666667
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.4214884696016772
Средневзвешенное: precision_score = 0.9253779165001639
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.6666666666666666
Класс 1: recall_score = 0.0
Класс 2: recall_score = 0.9727520435967303
Класс 3: recall_score = 0.0
Класс 4: recall_score = 0.9285714285714286
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.4279983564724709
Средневзвешенное: recall_score = 0.9376869829455735
```

Подбор гиперпараметров:

In [90]:

```
svc_poly_grid = RandomizedSearchCV(estimator = SVC(kernel = "poly", random_state = 16),
                          param_distributions = {'C': expon(scale=50), 'degree' : randint(2, 8), 'gamma' : un
                          n_iter = 20,
                          scoring = "recall_macro",
                          n_jobs = 8,
                          refit = True,
                          random_state = 16,
                          error_score = 0)
```

In [91]:

```
svc_poly_best = svc_poly_grid.fit(data[columns[:-1]], data[columns[-1]]).best_estimator_
```

In [92]:

```
svc_poly_best_prediction = svc_poly_best.predict(x_test)
PrintClassificationMetrics(y_test, svc_poly_best_prediction)
```

```
1. Общая точность (accuracy) = 1.0
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 1.0
Класс 2: precision_score = 1.0
Класс 3: precision_score = 1.0
Класс 4: precision_score = 1.0
Класс 5: precision_score = 1.0
Среднее арифм-е: precision_score = 1.0
Средневзвешенное: precision_score = 1.0
3. Полнота (recall) по классам:
Класс 0: recall_score = 1.0
Класс 1: recall_score = 1.0
Класс 2: recall_score = 1.0
Класс 3: recall_score = 1.0
Класс 4: recall_score = 1.0
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 1.0
Средневзвешенное: recall_score = 1.0
```

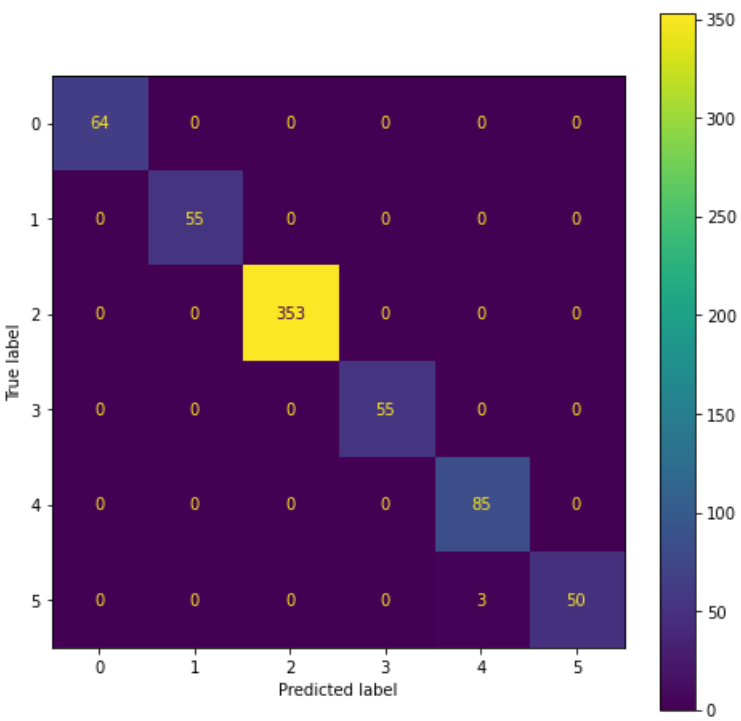
In [93]:

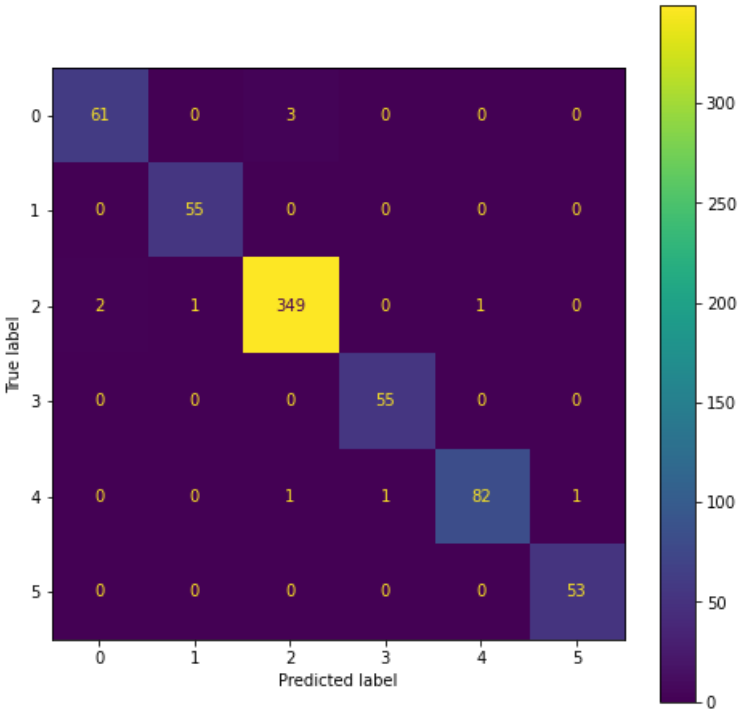
```
svc_poly_adasyn, svc_poly_adasyn_prediction = FitPredictClassifier(clone(svc_poly_best),
                                                    adasyn_x_train,
                                                    adasyn_x_test,
                                                    adasyn_y_train,
                                                    adasyn_y_test)
PrintClassificationMetrics(adasyn_y_test, svc_poly_adasyn_prediction)
```

```
1. Общая точность (accuracy) = 0.9849624060150376
2. Меткость (precision) по классам:
Класс 0: precision_score = 0.9682539682539683
Класс 1: precision_score = 0.9821428571428571
Класс 2: precision_score = 0.9886685552407932
Класс 3: precision_score = 0.9821428571428571
Класс 4: precision_score = 0.9879518072289156
Класс 5: precision_score = 0.9814814814814815
Среднее арифм-е: precision_score = 0.9817735877484789
Средневзвешенное: precision_score = 0.9873866028461978
3. Полнота (recall) по классам:
Класс 0: recall_score = 0.953125
Класс 1: recall_score = 1.0
Класс 2: recall_score = 0.9886685552407932
Класс 3: recall_score = 1.0
Класс 4: recall_score = 0.9647058823529412
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 0.9844165729322891
Средневзвешенное: recall_score = 0.9848402770497992
```

In [94]:

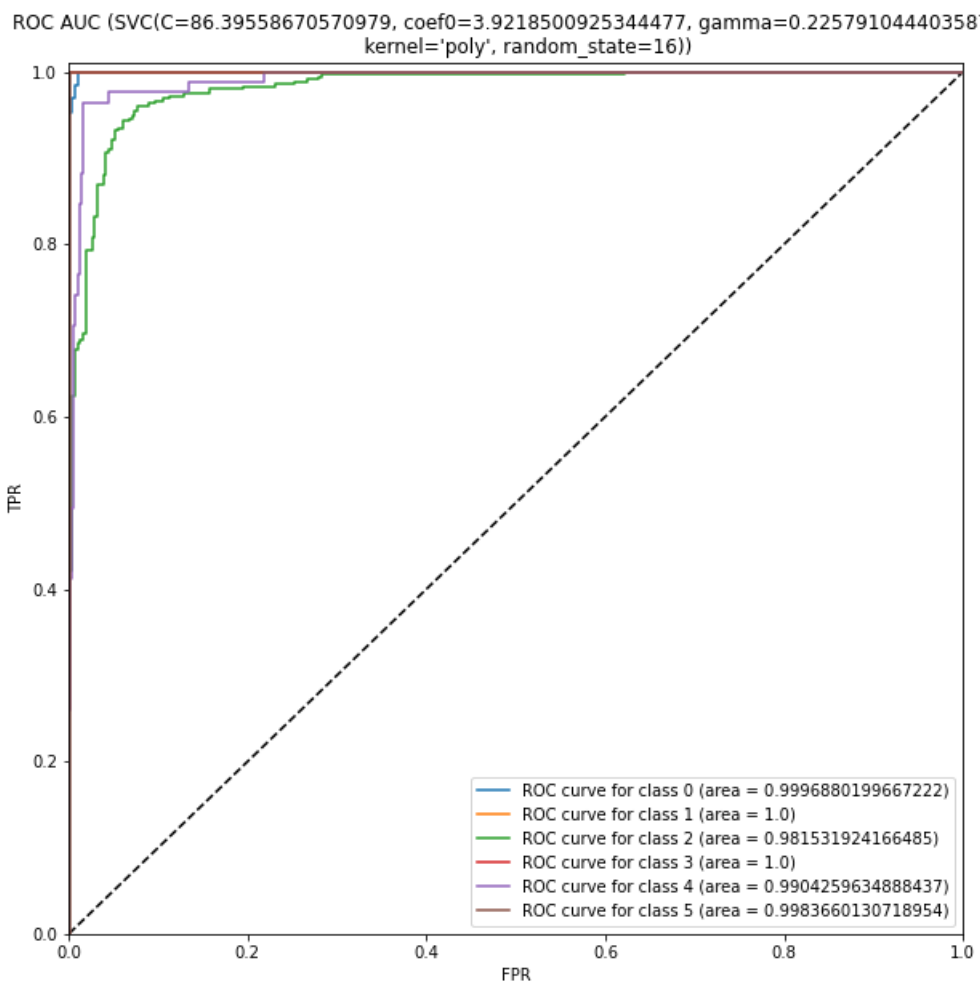
```
ShowConfusionMatrix(svc_poly_best, adasyn_x_test, adasyn_y_test)
ShowConfusionMatrix(svc_poly_adasyn, adasyn_x_test, adasyn_y_test)
```




In [95]:

```
PlotMulticlassRocAuc(svc_poly_adasyn, adasyn_x_train, adasyn_x_test, adasyn_y_train, adasyn_y_test)
```




Что ж, победило полиномиальное ядро!

**Дерево решений**

Дать какое-либо предсказание по результатам работы дерева сложно. Можно разве что сказать, что дерево, возможно, окажется менее чувствительным к нехватке данных из-за того, что при небольшой глубине все имеющиеся образцы будут определены как один класс (при этом дерево даже не будет "знать", что это верное решение).

Смотрим на базовое решение:

In [96]:

```
tree, tree_prediction = FitPredictClassifier(DecisionTreeClassifier(random_state = 16),
                                             x_train,
                                             x_test,
                                             y_train,
                                             y_test)
PrintClassificationMetrics(y_test, tree_prediction)
```

```
1. Общая точность (accuracy) = 0.9954233409610984
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 1.0
Класс 2: precision_score = 1.0
Класс 3: precision_score = 1.0
Класс 4: precision_score = 0.9545454545454546
Класс 5: precision_score = 0.0
Среднее арифм-е: precision_score = 0.8257575757575758
Средневзвешенное: precision_score = 0.993261175001301
3. Полнота (recall) по классам:
Класс 0: recall_score = 1.0
Класс 1: recall_score = 1.0
Класс 2: recall_score = 1.0
Класс 3: recall_score = 1.0
Класс 4: recall_score = 1.0
Класс 5: recall_score = 0.0
Среднее арифм-е: recall_score = 0.8333333333333334
Средневзвешенное: recall_score = 0.9971379507727534
```

Посмотрим, насколько базовое решение далеко от оптимального:

In [97]:

```
tree_grid = RandomizedSearchCV(estimator = DecisionTreeClassifier(random_state = 16),
                               param_distributions = {"max_depth" : randint(2, 15)},
                               n_iter = 10,
                               scoring = "precision_macro",
                               n_jobs = 8,
                               refit = True,
                               random_state = 16,
                               error_score = 0)
```

In [98]:

```
tree_best = tree_grid.fit(data[columns[:-1]], data[columns[-1]]).best_estimator_
```

In [99]:

```
tree_best_prediction = tree_best.predict(x_test)
PrintClassificationMetrics(y_test, tree_best_prediction)
```

```
1. Общая точность (accuracy) = 1.0
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 1.0
Класс 2: precision_score = 1.0
Класс 3: precision_score = 1.0
Класс 4: precision_score = 1.0
Класс 5: precision_score = 1.0
Среднее арифм-е: precision_score = 1.0
Средневзвешенное: precision_score = 1.0
3. Полнота (recall) по классам:
Класс 0: recall_score = 1.0
Класс 1: recall_score = 1.0
Класс 2: recall_score = 1.0
Класс 3: recall_score = 1.0
Класс 4: recall_score = 1.0
Класс 5: recall_score = 1.0
Среднее арифм-е: recall_score = 1.0
Средневзвешенное: recall_score = 1.0
```

Спасибо дереву решений за то, что оно существует! Но нельзя не проверить такой чудесный результат на большей выборке.

In [100]:

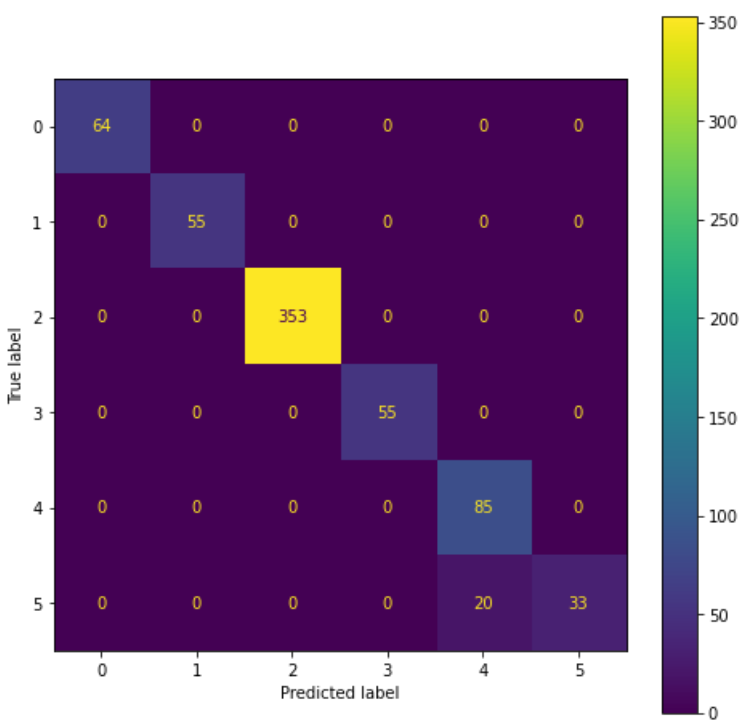
```
tree_adasyn, tree_adasyn_prediction = FitPredictClassifier(clone(tree_best),
                                                           adasyn_x_train,
                                                           adasyn_x_test,
                                                           adasyn_y_train,
                                                           adasyn_y_test)
PrintClassificationMetrics(adasyn_y_test, tree_adasyn_prediction)
```

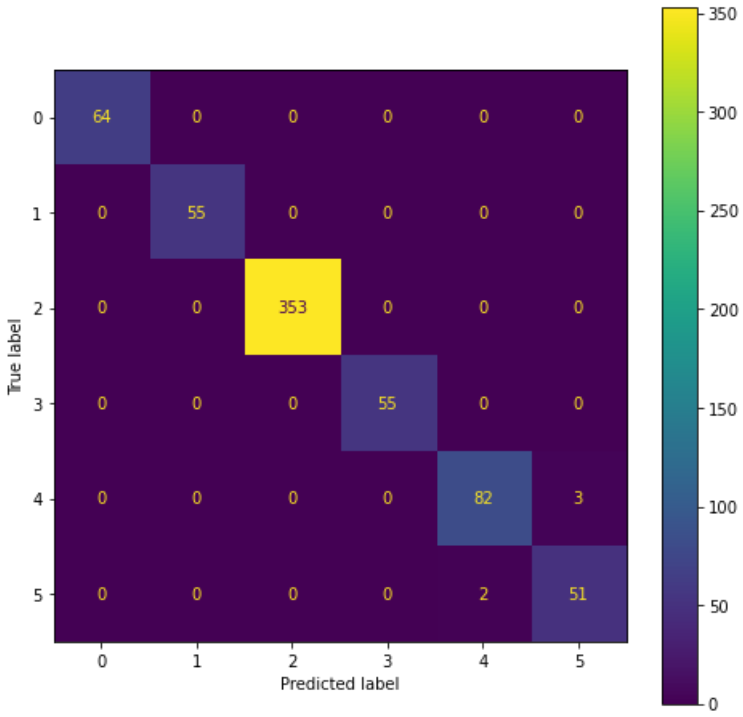
```
1. Общая точность (accuracy) = 0.9924812030075187
2. Меткость (precision) по классам:
Класс 0: precision_score = 1.0
Класс 1: precision_score = 1.0
Класс 2: precision_score = 1.0
Класс 3: precision_score = 1.0
Класс 4: precision_score = 0.9761904761904762
Класс 5: precision_score = 0.9444444444444444
Среднее арифм-е: precision_score = 0.9867724867724869
Средневзвешенное: precision_score = 0.9978103051943923
3. Полнота (recall) по классам:
Класс 0: recall_score = 1.0
Класс 1: recall_score = 1.0
Класс 2: recall_score = 1.0
Класс 3: recall_score = 1.0
Класс 4: recall_score = 0.9647058823529412
Класс 5: recall_score = 0.9622641509433962
Среднее арифм-е: recall_score = 0.9878283388827228
Средневзвешенное: recall_score = 0.9968817957786522
```

Уже не так красиво...

In [101]:

```
ShowConfusionMatrix(tree_best, adasyn_x_test, adasyn_y_test)
ShowConfusionMatrix(tree_adasyn, adasyn_x_test, adasyn_y_test)
```




Не имеет никакого смысла переходить к ансамблевым моделям, поскольку результат уже хороший и никакое переобучение здесь не светит...

### Выводы

Проблемы набора данных успешно побеждены: даже без искусственного занесения в выборку получили единичные (или близкие) по значению показатели качества для нескольких моделей.

Плохо себя показала только машина опорных векторов с сигмоидальным ядром, и ещё небезупречен линейный SVM-классификатор. Можно также отметить, что для обучения логистической регрессии жизненно важна искусственно сбалансированная выборка.

Тяжело сказать, какой эффект oversampling произвёл бы на модель при реальном применении, однако на текущий момент данных достаточно для того, чтобы обучиться и не переобучиться.

### Бонус! Пульсары по запросу Юрия Евгеньевича

Продолжим работу с датасетом из четвёртой и пятой лабораторных работ. Здесь выборка тоже дисбалансированная, однако проблема совершенно иная: 11.5 тысяч меток против тысячи. В связи с этим были использованы только методы Undersampling-а, точнее - только InstanceHardnessThreshold. Расширим данное исследование, опробовав на разных моделях ещё и ADASYN. Соответственно, здесь же задействуем и ансамблевые модели.

Предобработка данных уже проводилась, поэтому просто загружаем датасеты и делим выборку.

In [102]:

```
pulsar_data = pd.read_csv("ML_Datasets/Prepared/pulsar_filled_n_scaled.csv")
pulsar_iht_data = pd.read_csv("ML_Datasets/Prepared/pulsar_undersampg_iht.csv")
pulsar_adasyn_data = pd.read_csv("ML_Datasets/Prepared/pulsar_oversampg_adasyn.csv")
pulsar_data = pulsar_data[pulsar_data.columns.to_list()[1:]]
pulsar_iht_data = pulsar_iht_data[pulsar_iht_data.columns.to_list()[1:]]
pulsar_adasyn_data = pulsar_adasyn_data[pulsar_adasyn_data.columns.to_list()[1:]]
```

In [103]:

```
pulsar_x_train, pulsar_x_test, pulsar_y_train, pulsar_y_test = train_test_split(pulsar_data[pulsar_data.columı
                                                                                pulsar_data[pulsar_data.columı
                                                                                train_size = 0.85,
                                                                                random_state = 16)

pulsar_iht_x_train, pulsar_iht_x_test, pulsar_iht_y_train, pulsar_iht_y_test = train_test_split(pulsar_iht_daı
                                                                                              pulsar_iht_daı
                                                                                              train_size = (
                                                                                              random_state =

pulsar_adasyn_x_train, pulsar_adasyn_x_test, pulsar_adasyn_y_train, pulsar_adasyn_y_test = train_test_split(pu
                                                                                                          pu
                                                                                                          tr
                                                                                                          ra
```

In [104]:

```
from sklearn.metrics import balanced_accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
from sklearn.metrics import plot_confusion_matrix


def PrintBinaryClassificationMetrics(y_test, y_predicted):
    print("-Погрешность (accuracy, balanced) = {0};\
          \n-Меткость (precision, класс 1) = {1};\
          \n-Полнота (recall, класс 1) = {2};\
          \n-F1 (класс 1) = {3};\
          \n-ROC AUC = {4}.".format(balanced_accuracy_score(y_test, y_predicted),
                                   precision_score(y_test, y_predicted, average = "binary"),
                                   recall_score(y_test, y_predicted, average = "binary"),
                                   f1_score(y_test, y_predicted, average = "binary"),
                                   roc_auc_score(y_test, y_predicted)))
```

In [105]:

```
# Отрисовка ROC-кривых для бинарной классификации (модификация функции выше)
def PlotBinaryRocAuc(base_model, y_test, y_prediction):
    # Строим кривую и считаем площадь под ней
    fpr, tpr, _ = roc_curve(y_test, y_prediction)
    roc_auc = auc(fpr, tpr)

    # Настраиваем область отрисовки
    fig, ax = plt.subplots(figsize = (6, 6))
    ax.set_title("ROC AUC ({})".format(base_model))
    ax.set_xlim(0.0, 1.0)
    ax.set_ylim(0.0, 1.01)
    ax.set_xlabel("FPR")
    ax.set_ylabel("TPR")

    # Рисуем базовую кривую, для которой AUC-ROC = 0.5
    ax.plot([0, 1], [0, 1], 'k--')

    # Отрисовываем кривую для каждого класса
    ax.plot(fpr, tpr, label='ROC curve (area = {0})'.format(roc_auc))
    ax.legend(loc="lower right")

    return
```

Опробуем модели логистической регрессии, машины опорных векторов на радиально-базисном ядре, случайный лес и градиентный бустинг.

**Логистическая регрессия**

В лабораторных работах датасет больше всего страдал из-за сравнительно низкого recall-а, поэтому будем использовать подбор гиперпараметров с опорой на эту метрику.

In [106]:

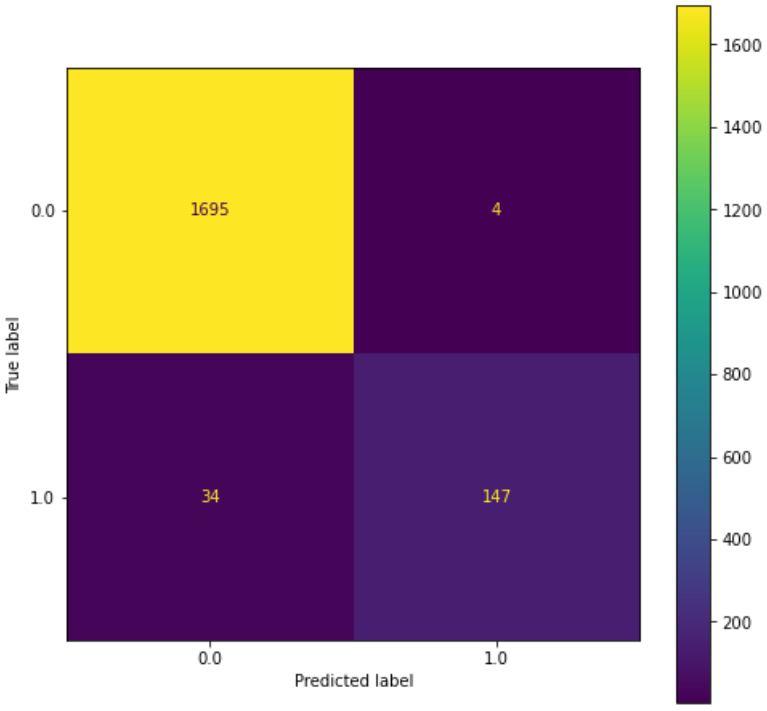
```
logreg_best = logreg_grid.fit(pulsar_data[pulsar_data.columns.to_list()[:-1]],
```

```
                                    pulsar_data[pulsar_data.columns.to_list()[-1]]).best_estimator_
```

In [129]:

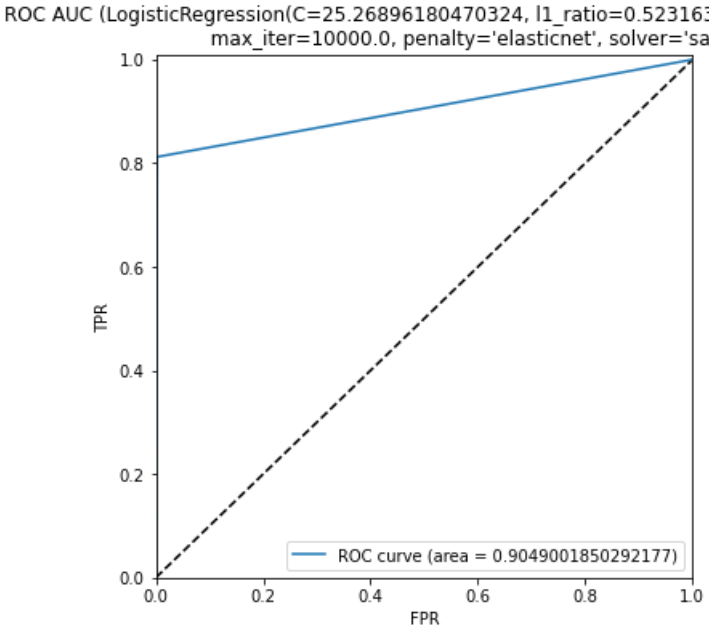
```
logreg_best_prediction = logreg_best.predict(pulsar_x_test)
PrintBinaryClassificationMetrics(pulsar_y_test, logreg_best_prediction)
ShowConfusionMatrix(logreg_best, pulsar_x_test, pulsar_y_test)
```

-Погрешность (accuracy, balanced) = 0.9049001850292178;
-Меткость (precision, класс 1) = 0.9735099337748344;
-Полнота (recall, класс 1) = 0.8121546961325967;
-F1 (класс 1) = 0.8855421686746988;
-ROC AUC = 0.9049001850292177.




In [108]:

```
PlotBinaryRocAuc(logreg_best, pulsar_y_test, logreg_best_prediction)
```




Теперь с восстановлением баланса...

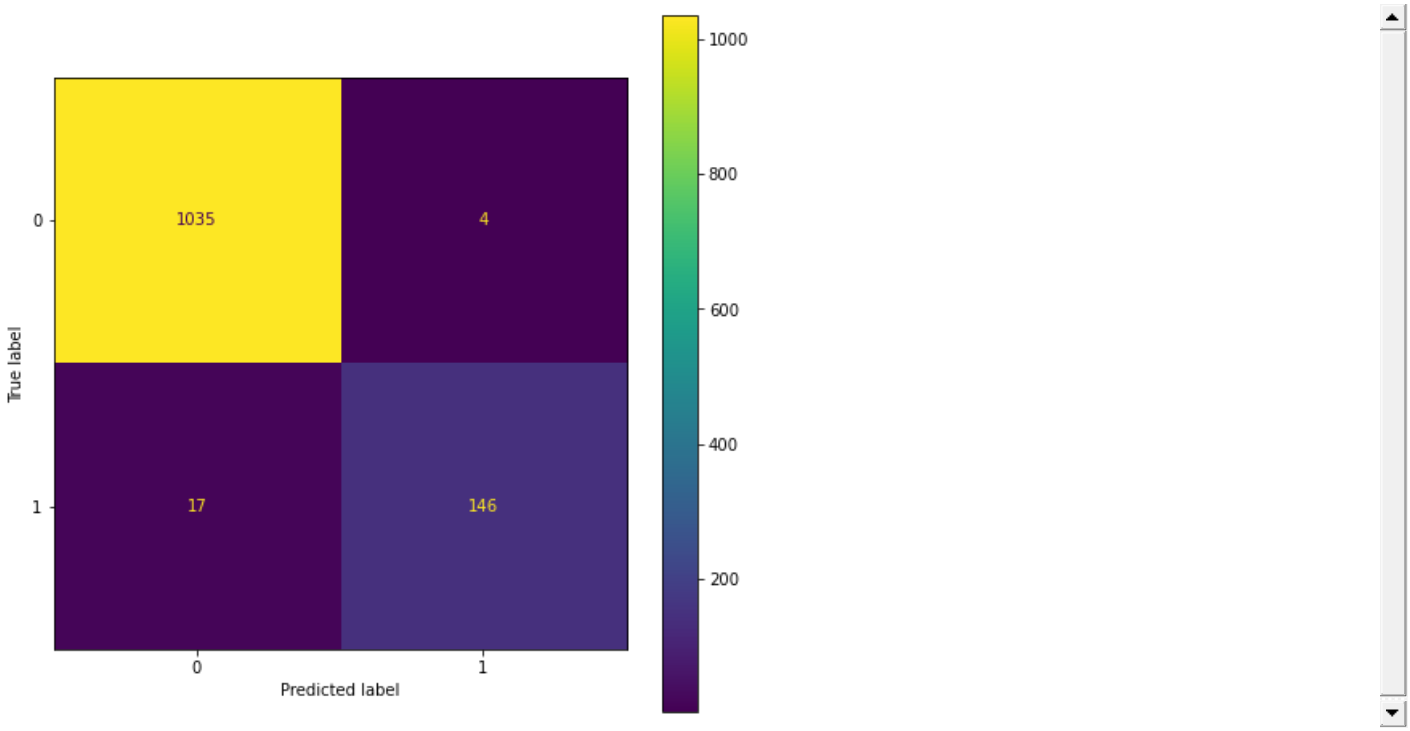
In [109]:

```
logreg_iht, logreg_iht_prediction = FitPredictClassifier(clone(logreg_best),
                                                         pulsar_iht_x_train,
                                                         pulsar_iht_x_test,
                                                         pulsar_iht_y_train,
                                                         pulsar_iht_y_test)
PrintBinaryClassificationMetrics(pulsar_iht_y_test, logreg_iht_prediction)
```

-Погрешность (accuracy, balanced) = 0.9459278329209895;
-Меткость (precision, класс 1) = 0.9733333333333334;
-Полнота (recall, класс 1) = 0.8957055214723927;
-F1 (класс 1) = 0.9329073482428116;
-ROC AUC = 0.9459278329209895.

In [131]:

```
ShowConfusionMatrix(logreg_iht, pulsar_iht_x_test, pulsar_iht_y_test)
```




Видим очень хороший результат: не потеряли в качестве ни по одной метрике, при этом сильный прирост в recall.

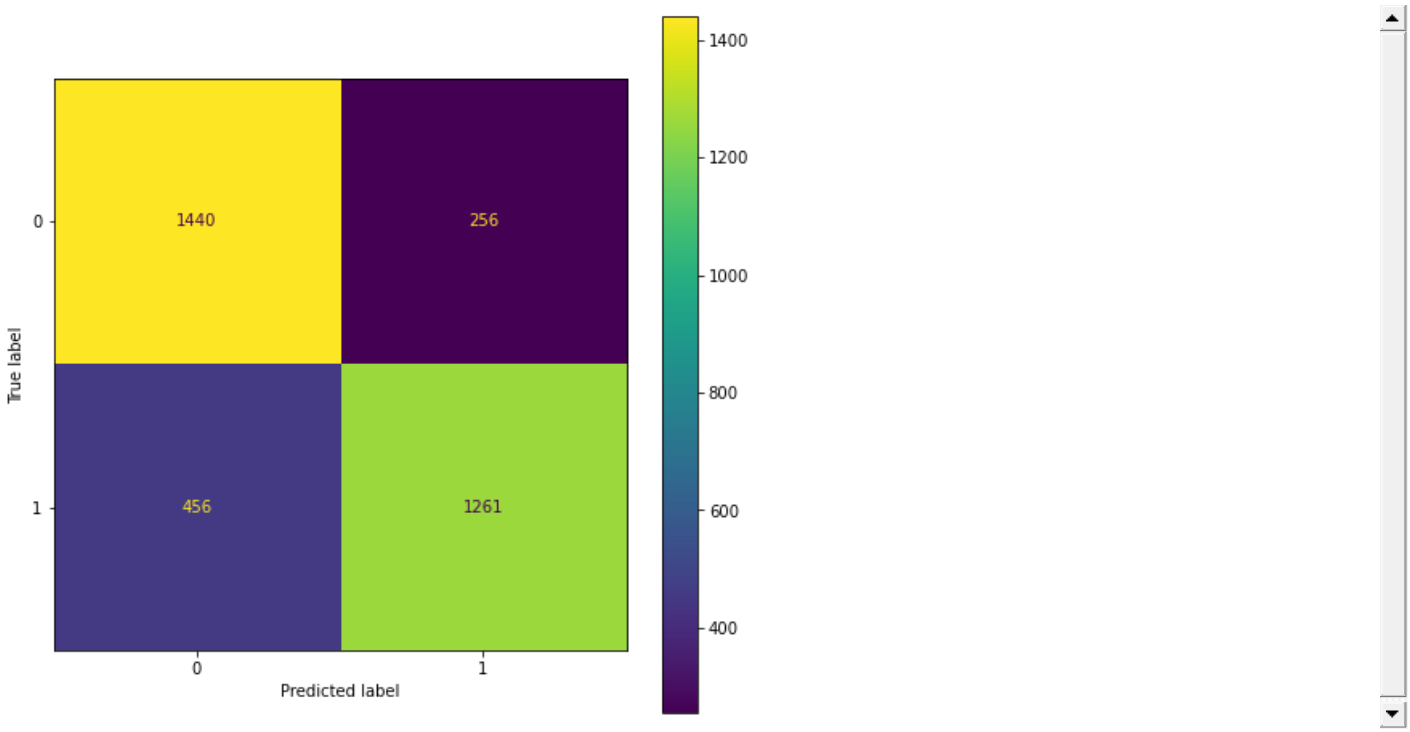
In [111]:

```
logreg_adasyn, logreg_adasyn_prediction = FitPredictClassifier(clone(logreg_best),
                                                               pulsar_adasyn_x_train,
                                                               pulsar_adasyn_x_test,
                                                               pulsar_adasyn_y_train,
                                                               pulsar_adasyn_y_test)
PrintBinaryClassificationMetrics(pulsar_adasyn_y_test, logreg_adasyn_prediction)
```

-Погрешность (accuracy, balanced) = 0.7917385523236009;
-Меткость (precision, класс 1) = 0.8312458800263678;
-Полнота (recall, класс 1) = 0.7344205008736168;
-F1 (класс 1) = 0.7798392084106369;
-ROC AUC = 0.7917385523236009.

In [112]:

```
ShowConfusionMatrix(logreg_adasyn, pulsar_adasyn_x_test, pulsar_adasyn_y_test)
```




Метод oversampling-а ожидаемо не дал никакого положительного результата, поскольку данных в minor-классе было изначально достаточно. В итоге модель, скорее всего, просто запуталась.

**Машина опорных векторов**

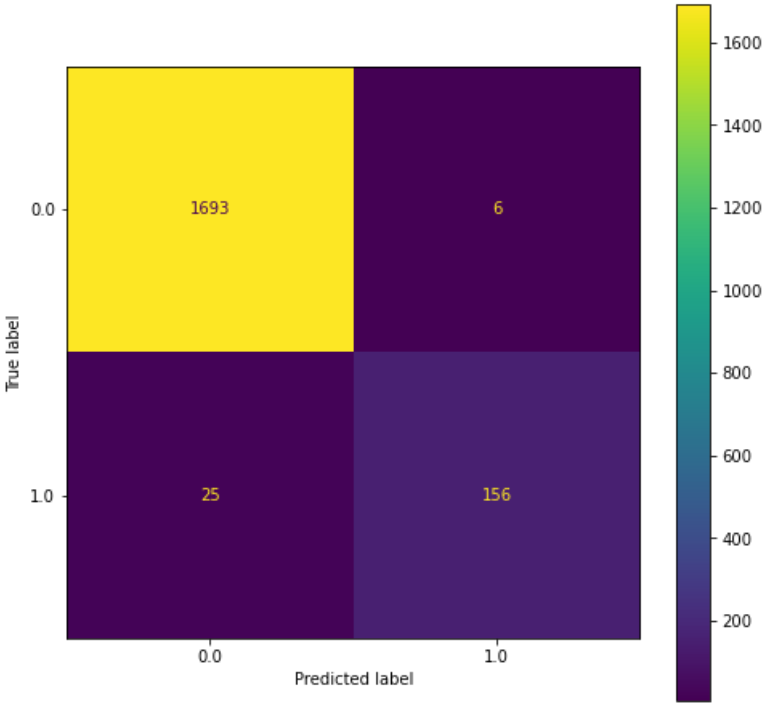
In [113]:

```
svc_rbf_grid = RandomizedSearchCV(estimator = SVC(random_state = 16),
                          param_distributions = {'C': expon(scale=50), 'gamma' : uniform(1e-04, 2)},
                          n_iter = 20,
                          scoring = "recall_macro",
                          n_jobs = 8,
                          refit = True,
                          random_state = 16,
                          error_score = 0)
svc_rbf_best = svc_rbf_grid.fit(pulsar_data[pulsar_data.columns.to_list()[:-1]],
                        pulsar_data[pulsar_data.columns.to_list()[-1]]).best_estimator_
```

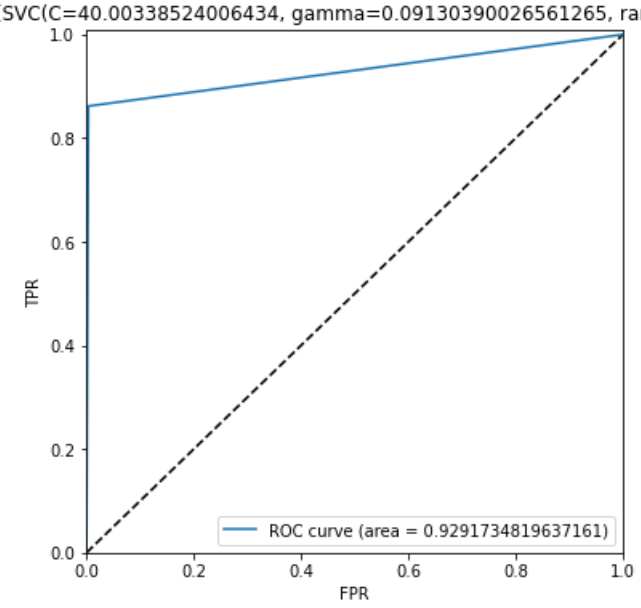
In [114]:

```
svc_rbf_best_prediction = svc_rbf_best.predict(pulsar_x_test)
PrintBinaryClassificationMetrics(pulsar_y_test, svc_rbf_best_prediction)
ShowConfusionMatrix(svc_rbf_best, pulsar_x_test, pulsar_y_test)
PlotBinaryRocAuc(svc_rbf_best, pulsar_y_test, svc_rbf_best_prediction)
```

-Погрешность (accuracy, balanced) = 0.9291734819637161;
-Меткость (precision, класс 1) = 0.9629629629629629;
-Полнота (recall, класс 1) = 0.861878453038674;
-F1 (класс 1) = 0.9096209912536444;
-ROC AUC = 0.9291734819637161.




ROC AUC (SVC(C=40.00338524006434, gamma=0.09130390026561265, random_state=16))



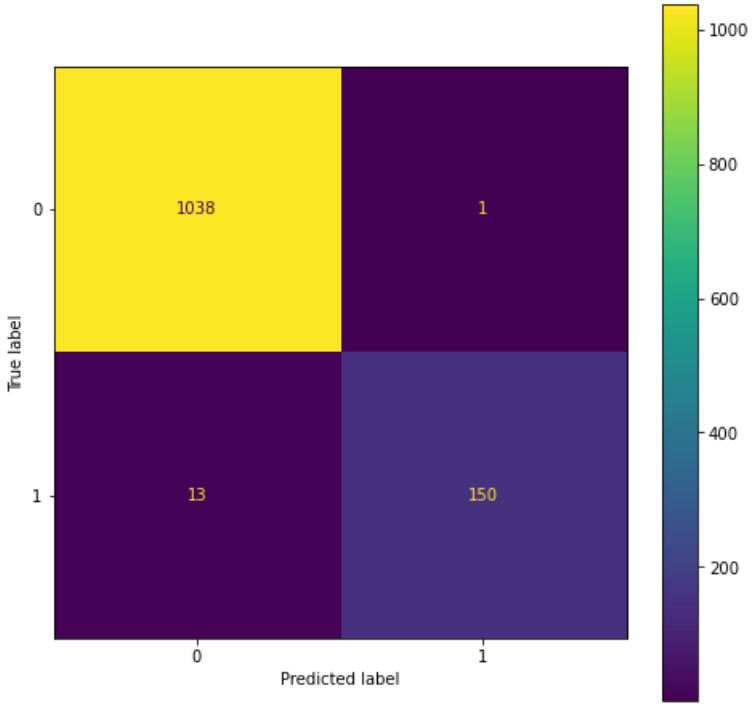

In [115]:

```
svc_rbf_iht, svc_rbf_iht_prediction = FitPredictClassifier(clone(svc_rbf_best),
                                                           pulsar_iht_x_train,
                                                           pulsar_iht_x_test,
                                                           pulsar_iht_y_train,
                                                           pulsar_iht_y_test)
```

In [116]:

```
ShowConfusionMatrix(svc_rbf_iht, pulsar_iht_x_test, pulsar_iht_y_test)
PrintBinaryClassificationMetrics(pulsar_iht_y_test, svc_rbf_iht_prediction)
```

-Погрешность (accuracy, balanced) = 0.9596414674327014;
-Меткость (precision, класс 1) = 0.9933774834437086;
-Полнота (recall, класс 1) = 0.9202453987730062;
-F1 (класс 1) = 0.9554140127388535;
-ROC AUC = 0.9596414674327013.



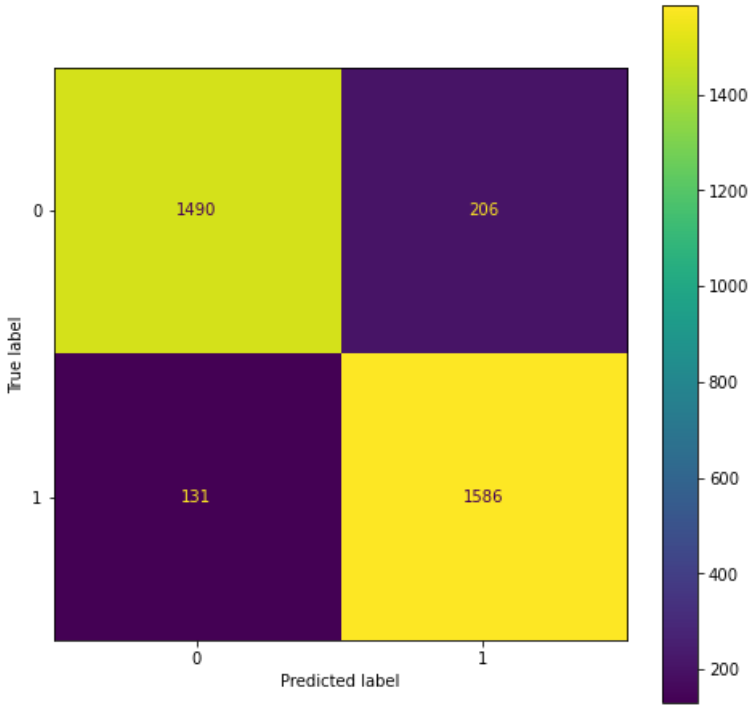

In [117]:

```
svc_rbf_adasyn, svc_rbf_adasyn_prediction = FitPredictClassifier(clone(svc_rbf_best),
                                                                 pulsar_adasyn_x_train,
                                                                 pulsar_adasyn_x_test,
                                                                 pulsar_adasyn_y_train,
                                                                 pulsar_adasyn_y_test)
```

In [118]:

```
ShowConfusionMatrix(svc_rbf_adasyn, pulsar_adasyn_x_test, pulsar_adasyn_y_test)
PrintBinaryClassificationMetrics(pulsar_adasyn_y_test, svc_rbf_adasyn_prediction)
```

-Погрешность (accuracy, balanced) = 0.9011209354842254;
-Меткость (precision, класс 1) = 0.8850446428571429;
-Полнота (recall, класс 1) = 0.9237041351193943;
-F1 (класс 1) = 0.9039612425192364;
-ROC AUC = 0.9011209354842253.



Здесь снова видим блестящий результат на undersampling-е, однако результат oversampling-а теперь не является однозначно наихудшим, поскольку recall здесь - самый высокий, пусть и с точностью погрешности.

**Случайный лес**

In [119]:

```
rf_grid = RandomizedSearchCV(estimator = RandomForestClassifier(random_state = 16),
                             param_distributions = {"n_estimators" : randint(1, 150), "max_depth" : randint(2, 1
                             n_iter = 10,
                             scoring = "recall_macro",
                             n_jobs = 8,
                             refit = True,
                             random_state = 16,
                             error_score = 0)
```

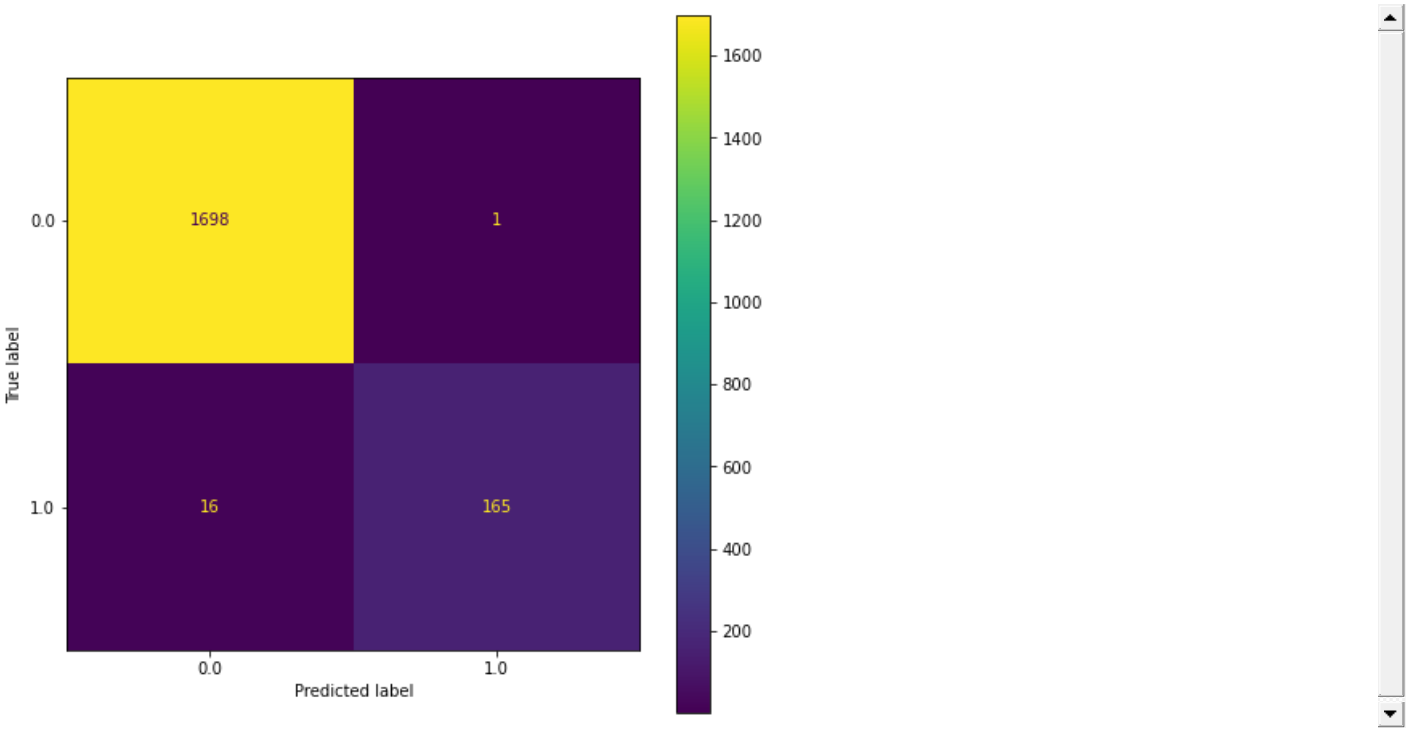
In [120]:

```
rf_best = rf_grid.fit(pulsar_data[pulsar_data.columns.to_list()[:-1]],
                      pulsar_data[pulsar_data.columns.to_list()[-1]]).best_estimator_
```

In [121]:

```
rf_best_prediction = rf_best.predict(pulsar_x_test)
PrintBinaryClassificationMetrics(pulsar_y_test, rf_best_prediction)
ShowConfusionMatrix(rf_best, pulsar_x_test, pulsar_y_test)
#PlotBinaryRocAuc(rf_best, pulsar_y_test, rf_best_prediction)
```

```
-Погрешность (accuracy, balanced) = 0.9555068142131056;
-Меткость (precision, класс 1) = 0.9939759036144579;
-Полнота (recall, класс 1) = 0.9116022099447514;
-F1 (класс 1) = 0.9510086455331411;
-ROC AUC = 0.9555068142131056.
```



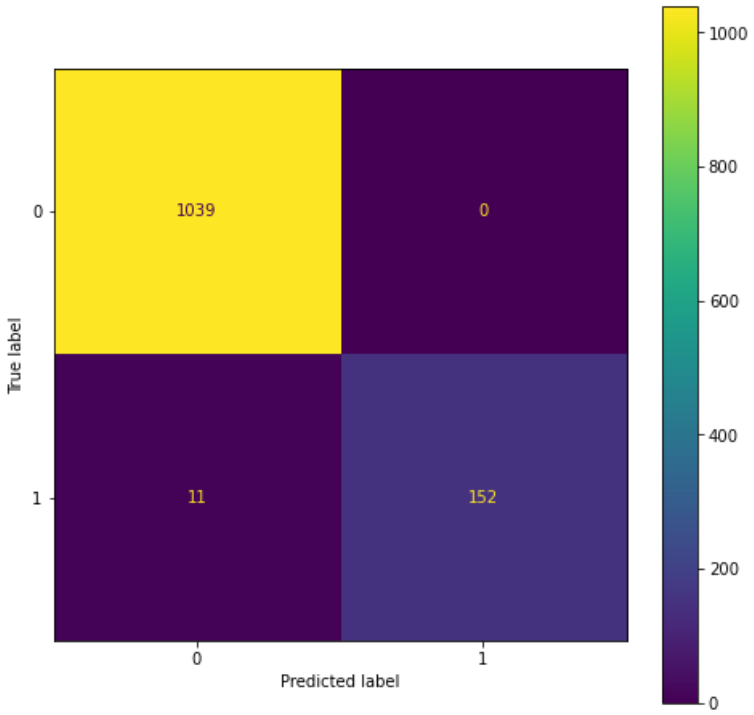

In [122]:

```
rf_iht, rf_iht_prediction = FitPredictClassifier(clone(rf_best),
                                                 pulsar_iht_x_train,
                                                 pulsar_iht_x_test,
                                                 pulsar_iht_y_train,
                                                 pulsar_iht_y_test)
```

In [123]:

```
ShowConfusionMatrix(rf_iht, pulsar_iht_x_test, pulsar_iht_y_test)
PrintBinaryClassificationMetrics(pulsar_iht_y_test, rf_iht_prediction)
```

```
-Погрешность (accuracy, balanced) = 0.9662576687116564;
-Меткость (precision, класс 1) = 1.0;
-Полнота (recall, класс 1) = 0.9325153374233128;
-F1 (класс 1) = 0.9650793650793651;
-ROC AUC = 0.9662576687116564.
```



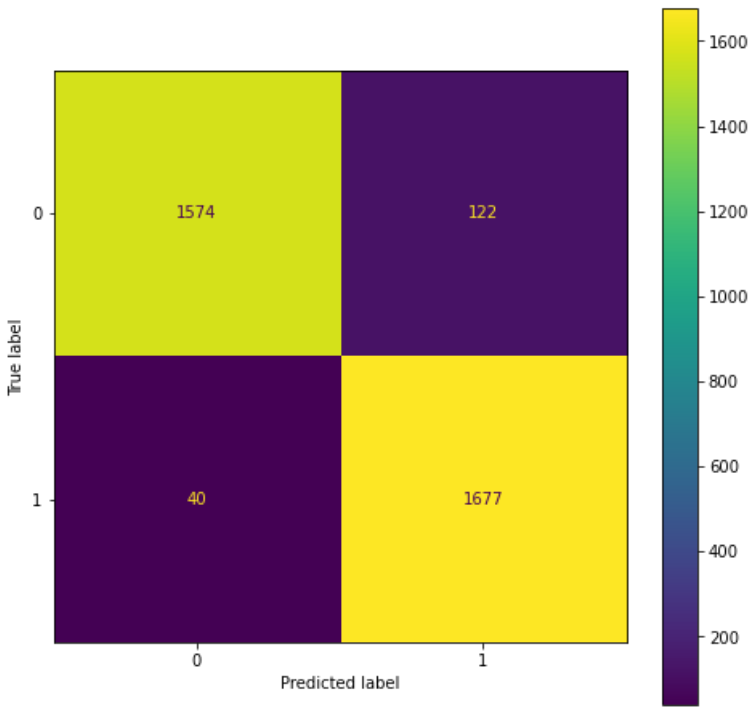

In [124]:

```
rf_adasyn, rf_adasyn_prediction = FitPredictClassifier(clone(rf_best),
                                                       pulsar_adasyn_x_train,
                                                       pulsar_adasyn_x_test,
                                                       pulsar_adasyn_y_train,
                                                       pulsar_adasyn_y_test)
```

In [125]:

```
ShowConfusionMatrix(rf_adasyn, pulsar_adasyn_x_test, pulsar_adasyn_y_test)
PrintBinaryClassificationMetrics(pulsar_adasyn_y_test, rf_adasyn_prediction)
```

```
-Погрешность (accuracy, balanced) = 0.9523847952220306;
-Меткость (precision, класс 1) = 0.9321845469705392;
-Полнота (recall, класс 1) = 0.976703552708212;
-F1 (класс 1) = 0.9539249146757679;
-ROC AUC = 0.9523847952220306.
```




Дерево решений почему-то выигрывает от Oversampling-а: видим прирост метрики *Recall*.

**Градиентный бустинг**

In [132]:

```
from catboost import CatBoost
```

In [126]:

```
cb = CatBoostClassifier(silent = True)
cb.fit(pulsar_x_train, pulsar_y_train)
cb_prediction = cb.predict(pulsar_x_test)
```

In [149]:

```
PrintBinaryClassificationMetrics(pulsar_y_test, cb_prediction)
```

```
-Погрешность (accuracy, balanced) = 0.9307587498658619;
-Меткость (precision, класс 1) = 0.9401197604790419;
-Полнота (recall, класс 1) = 0.8674033149171271;
-F1 (класс 1) = 0.9022988505747126;
-ROC AUC = 0.9307587498658619.
```

In [136]:

```
cb_best = CatBoost({"logging_level" : "Silent"}).randomized_search({"depth" : randint(2, 50),
                                                                    "learning_rate" : expon(scale = 0.01),
                                                                    "n_estimators" : randint(10, 150)
                                                                   },
                                                                   pulsar_data[pulsar_data.columns.to_list()[:-1]],
                                                                   pulsar_data[pulsar_data.columns.to_list()[-1]],
                                                                   cv = 4,
                                                                   n_iter = 30,
                                                                   partition_random_seed = 16,
                                                                   verbose = False)
```

In [139]:

```
cb_best["params"]
```

Out[139]:

```
{'depth': 12.0, 'learning_rate': 0.013281213624198153, 'iterations': 90.0}
```

In [146]:

```
cb_best_model = CatBoostClassifier(learning_rate = cb_best["params"]["learning_rate"], depth = cb_best["params
```

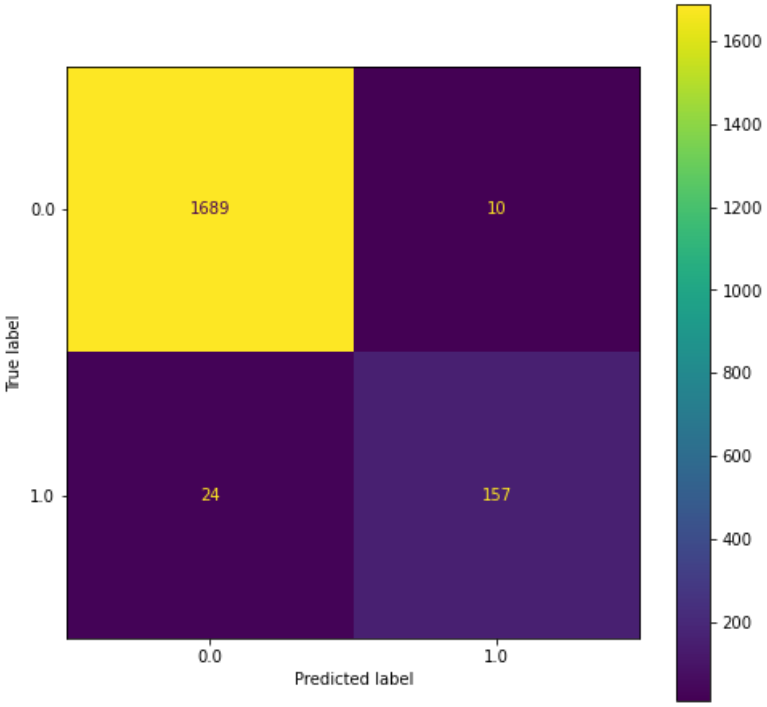
In [147]:

```
cb_best_model.fit(pulsar_x_train, pulsar_y_train)
cb_best_prediction = cb_best_model.predict(pulsar_x_test)
```

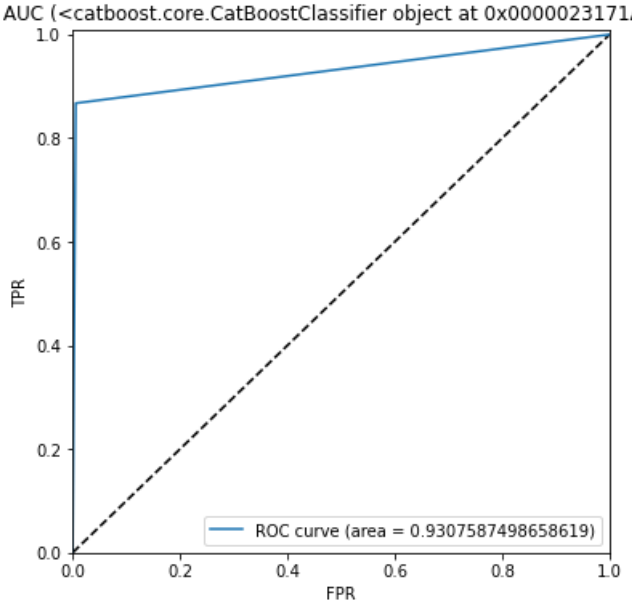
In [148]:

```
PrintBinaryClassificationMetrics(pulsar_y_test, cb_best_prediction)
ShowConfusionMatrix(cb_best_model, pulsar_x_test, pulsar_y_test)
PlotBinaryRocAuc(cb_best_model, pulsar_y_test, cb_prediction)
```

-Погрешность (accuracy, balanced) = 0.9307587498658619;
-Меткость (precision, класс 1) = 0.9401197604790419;
-Полнота (recall, класс 1) = 0.8674033149171271;
-F1 (класс 1) = 0.9022988505747126;
-ROC AUC = 0.9307587498658619.




ROC AUC (<catboost.core.CatBoostClassifier object at 0x0000023171A7F970>)



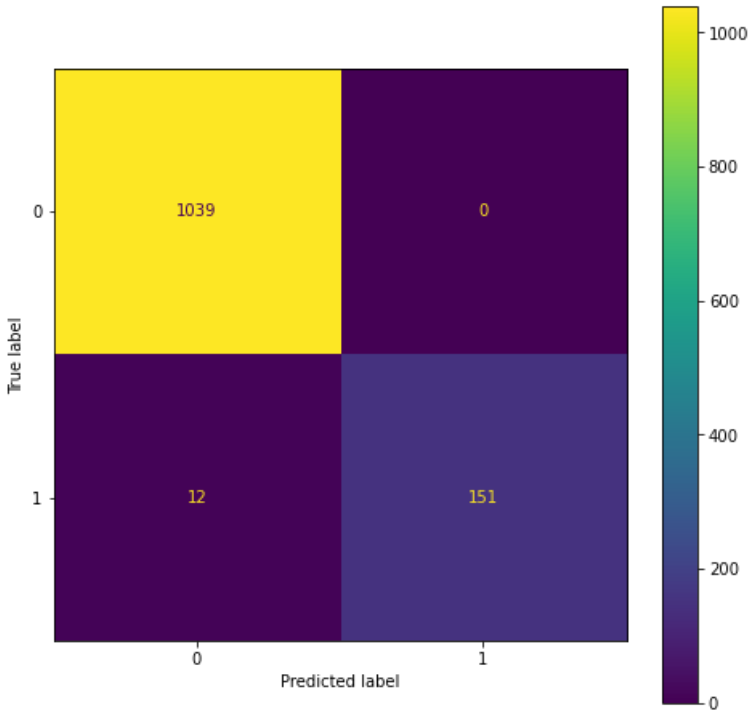

In [150]:

```
cb_iht, cb_iht_prediction = FitPredictClassifier(clone(cb_best_model),
                                                 pulsar_iht_x_train,
                                                 pulsar_iht_x_test,
                                                 pulsar_iht_y_train,
                                                 pulsar_iht_y_test)
```

In [151]:

```
ShowConfusionMatrix(cb_iht, pulsar_iht_x_test, pulsar_iht_y_test)
PrintBinaryClassificationMetrics(pulsar_iht_y_test, cb_iht_prediction)
```

-Погрешность (accuracy, balanced) = 0.9631901840490797;
-Меткость (precision, класс 1) = 1.0;
-Полнота (recall, класс 1) = 0.9263803680981595;
-F1 (класс 1) = 0.9617834394904459;
-ROC AUC = 0.9631901840490797.



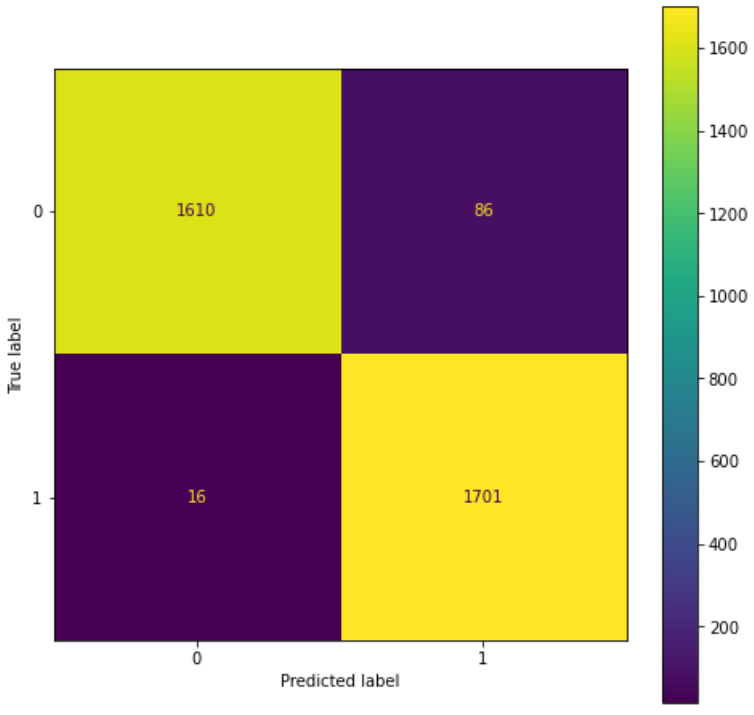

In [152]:

```
cb_adasyn, cb_adasyn_prediction = FitPredictClassifier(clone(cb_best_model),
                                                       pulsar_adasyn_x_train,
                                                       pulsar_adasyn_x_test,
                                                       pulsar_adasyn_y_train,
                                                       pulsar_adasyn_y_test)
```

In [153]:

```
ShowConfusionMatrix(cb_adasyn, pulsar_adasyn_x_test, pulsar_adasyn_y_test)
PrintBinaryClassificationMetrics(pulsar_adasyn_y_test, cb_adasyn_prediction)
```

-Погрешность (accuracy, balanced) = 0.9699869369567368;
-Меткость (precision, класс 1) = 0.9518746502518187;
-Полнота (recall, класс 1) = 0.9906814210832848;
-F1 (класс 1) = 0.970890410958904;
-ROC AUC = 0.9699869369567368.



**Выводы**

Датасет с проблемой дисбаланса и при этом отсутствие нехватки объектов minor-класса ожидаемо лучше предсказывается после обработки методами Undersampling-а. При этом качество после сравнивания числа образцов путём Oversampling-а тоже улучшается, но только на моделях, основанных на деревьях решений. В целом (за исключением деревьев) рекомендуется использоваться только первую группу методов, так как она более стабильна.