МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра прикладной математики (ПМ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

по дисциплине «Технологии и инструментарий машинного обучения»

Практическая работа № 4

Студент группы ИМБО-01-21 Малкина В.В. _____ (подпись)

Старший преподаватель Высоцкая А.А. _____ (подпись)

Отчет представлен «___»_____2023г.

Москва 2023 г.

1 Предобработка данных

Выведем общую информацию о столбцах из набора (Рисунок 1).

```
data = pd.read_csv('/Users/vladamalkina/Downloads/diabetes.csv')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Pregnancies    768 non-null    int64  
 1   Glucose        768 non-null    int64  
 2   BloodPressure  768 non-null    int64  
 3   SkinThickness  768 non-null    int64  
 4   Insulin        768 non-null    int64  
 5   BMI            768 non-null    float64
 6   Pedigree       768 non-null    float64
 7   Age            768 non-null    int64  
 8   Outcome        768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Рисунок 1 — Вывод общей информации

Исследуем данные на выбросы, построим график «ящик с усами» (Рисунок 2-5).

```
for col in data.columns:
    if col != 'Outcome':
        plt.figure(figsize=(5, 3))
        sns.boxplot(x=data[col])
        plt.show()
```

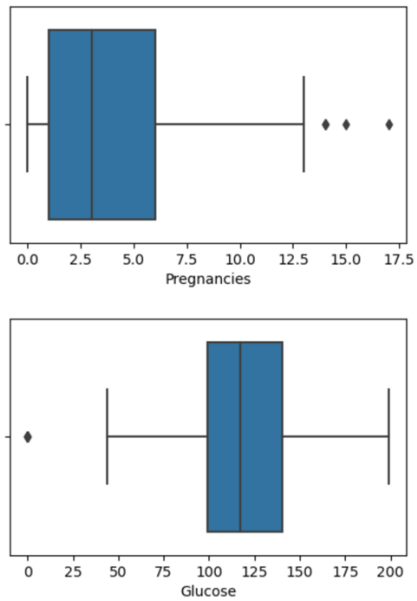

Рисунок 2 — Графики «Ящики с усами»

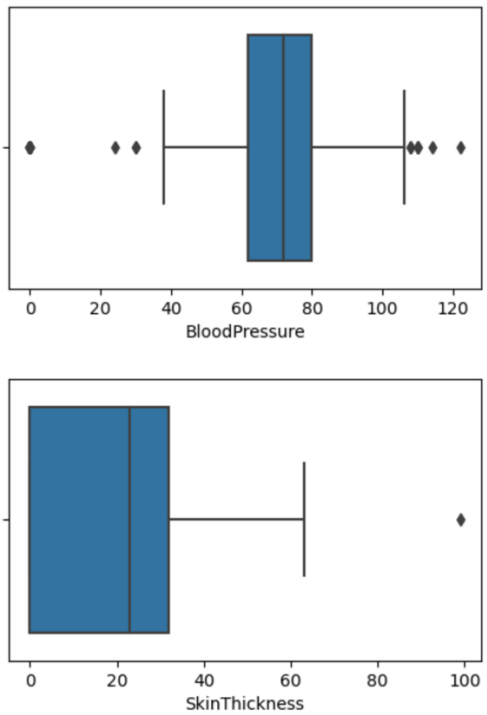

Рисунок 3 — Графики «Ящики с усами»

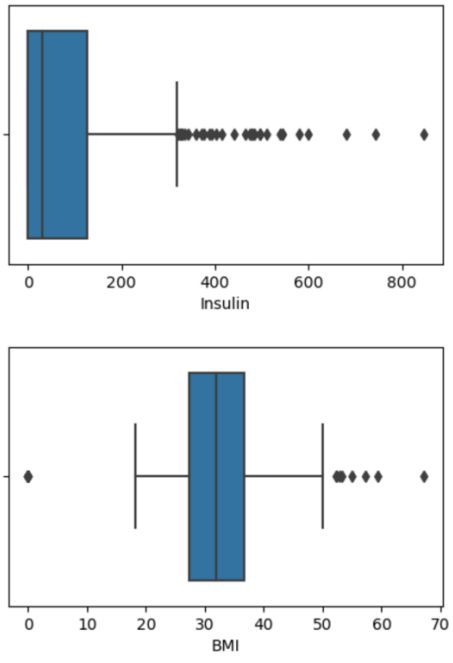

Рисунок 4 — Графики «Ящики с усами»

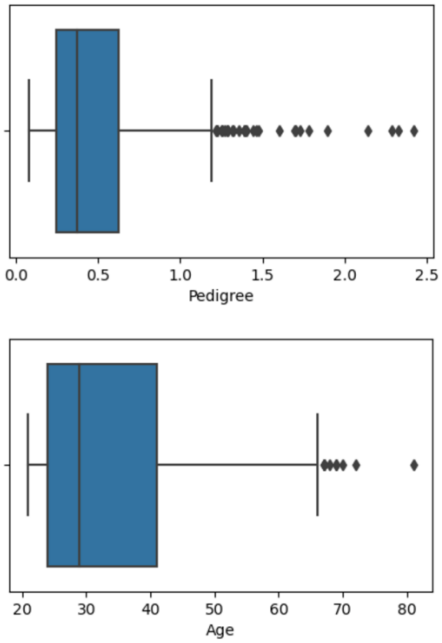

Рисунок 5 — Графики «Ящики с усами»

Были обнаружены выбросы, удалим их (Рисунок 6).

```
for col in data.columns:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    data = data.loc[(data[col] >= Q1-1.5*IQR ) & (data[col] <= Q3+1.5*IQR)]
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 636 entries, 0 to 767
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Pregnancies    636 non-null    int64
 1   Glucose        636 non-null    int64
 2   BloodPressure  636 non-null    int64
 3   SkinThickness  636 non-null    int64
 4   Insulin        636 non-null    int64
 5   BMI            636 non-null    float64
 6   Pedigree       636 non-null    float64
 7   Age            636 non-null    int64
 8   Outcome        636 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 49.7 KB
```

Рисунок 6 — Удаление выбросов

Разделим данные на тренировочную и тестовую выборки (Рисунок 7).

```
x_train = data_train.drop('LeaveOrNot', axis = 1)
y_train = data_train['LeaveOrNot']
x_test = data_test.drop('LeaveOrNot', axis = 1)
y_test = data_test['LeaveOrNot']
```

```
categorical_cols = data.select_dtypes(include=["object"]).columns.tolist()
print(len(categorical_cols))
categorical_cols
```

```
4

['Education', 'City', 'Gender', 'EverBenched']
```

```
numeric_cols = data.select_dtypes(exclude=["object"]).columns.tolist()
print(len(numeric_cols))
numeric_cols
target_name = 'LeaveOrNot'
numeric_cols.remove(target_name)
```

```
5
```

Рисунок 7 — Разделение данных на выборки

Нормализуем численные признаки (Рисунок 8).

```
#сделаем одну выборку с нормализованными численными признаками
from sklearn.preprocessing import StandardScaler

X_train_num = x_train[numeric_cols]
X_test_num = x_test[numeric_cols]

scaler = StandardScaler()
X_train_num_scaled = scaler.fit_transform(X_train_num)
X_test_num_scaled = scaler.transform(X_test_num)
```

```
X_train_num_scaled = pd.DataFrame(X_train_num_scaled)
X_test_num_scaled = pd.DataFrame(X_test_num_scaled)
```

Рисунок 8 — Нормализация численных признаков

Проверим данные на дисбаланс классов ().

```
from sklearn.model_selection import train_test_split
data_train, data_test = train_test_split(data, test_size = 0.2, random_state = 12345)
```

```
target_name = 'Outcome'
class_frequency = data_train[target_name].value_counts(normalize = True)
print(class_frequency)
class_frequency.plot(kind='bar')
```

```
0    0.694882
1    0.305118
Name: Outcome, dtype: float64

<AxesSubplot:>
```

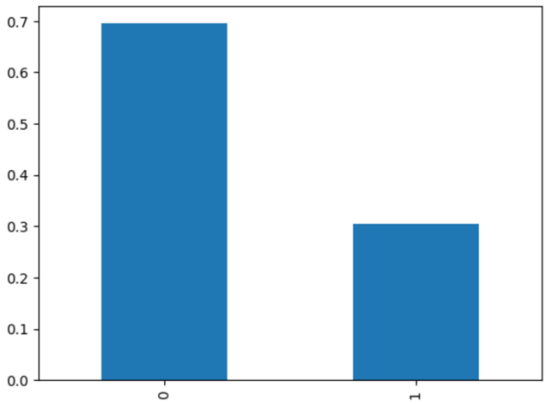

Рисунок 9 — Проверка данных на дисбаланс классов

Для борьбы с дисбалансом используем технику upsample (Рисунок 10).

```
def upsample(features, target, repeat):
    features_zeros = features[target == 0]
    features_ones = features[target == 1]
    target_zeros = target[target == 0]
    target_ones = target[target == 1]
    features_upsampled = pd.concat([features_zeros] + [features_ones] * repeat)
    target_upsampled = pd.concat([target_zeros] + [target_ones] * repeat)
    features_upsampled, target_upsampled = shuffle(features_upsampled, target_upsampled, random_state=12345)
    return features_upsampled, target_upsampled

features_upsampled, target_upsampled = upsample(X_train_num_scaled, y_train, 2)
```

Рисунок 10 — Применение техники upsample

```
class_frequency = target_upsampled.value_counts(normalize = True)
print(class_frequency)
class_frequency.plot(kind='bar')
```

```
0    0.532428
1    0.467572
Name: Outcome, dtype: float64

<AxesSubplot:>
```

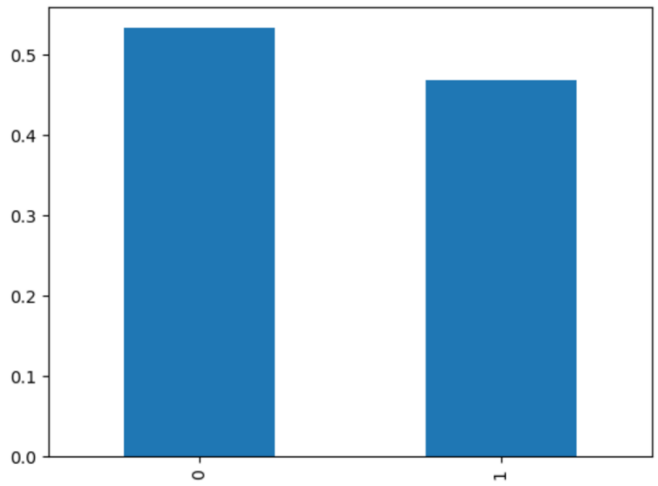

Рисунок 11 — Борьба с дисбалансом

Напишем функцию для вывода метрик, чтобы оценить качество моделей (Рисунок 12).

```
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import classification_report
def metr(y_pred, y_test):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)

    print(classification_report(y_test, y_pred))

    print(sklearn.metrics.confusion_matrix(y_test, y_pred))
```

Рисунок 12 — Функция для вывода метрик

Напишем функции, вычисляющие Энтропию Шеннона и индекс Джинни (Рисунки 13-14).

```
import math
def shannon_entropy(data):
    # Вычисляем количество элементов в наборе данных
    total_count = len(data)
    # Подсчитываем частоту каждого значения в наборе данных
    value_counts = {}
    for value in data:
        if value in value_counts:
            value_counts[value] += 1
        else:
            value_counts[value] = 1
    # Вычисляем энтропию Шеннона
    entropy = 0
    for count in value_counts.values():
        probability = count / total_count
        entropy -= probability * math.log2(probability)
    return entropy
```

Рисунок 13 — Функция для расчета энтропии Шеннона

```
def gini(x):
    # Вычисляем количество элементов в наборе данных
    total_count = len(data)
    # Подсчитываем частоту каждого значения в наборе данных
    value_counts = {}
    for value in data:
        if value in value_counts:
            value_counts[value] += 1
        else:
            value_counts[value] = 1

    g = 0
    for count in value_counts.values():
        probability = count / total_count
        g += probability * (1-probability)
    return g
```

Рисунок 14 — Функция для расчета индекса Джинни

2 Обучение моделей

Обучим дерево решений(Рисунок 15).

DecisionTree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {'max_depth': [3, 4, 5, 6, 7, 8],
              'min_samples_leaf': [2, 3, 4, 5],
              'max_leaf_nodes' : [5, 6, 7, 8, 9]
             }
tree_class = DecisionTreeClassifier(random_state=1024)
grid_search = GridSearchCV(tree_class, param_grid, cv=5, verbose=True)
grid_search.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 120 candidates, totalling 600 fits

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=1024),
             param_grid={'max_depth': [3, 4, 5, 6, 7, 8],
                         'max_leaf_nodes': [5, 6, 7, 8, 9],
                         'min_samples_leaf': [2, 3, 4, 5]},
             verbose=True)
```

Рисунок 15 — Обучение дерева решений

Теперь выведем метрики и матрицу ошибок (Рисунок 16).

```
Accuracy: 0.328125
Precision: 0.1640625
Recall: 0.5
F1 Score: 0.24705882352941178
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        86
           1       0.33      1.00      0.49        42

    accuracy                           0.33       128
   macro avg       0.16      0.50      0.25       128
weighted avg       0.11      0.33      0.16       128

[[ 0 86]
 [ 0 42]]
Entropy: 0.0
Index Gini: 0.014128693485226057
```

Рисунок 16 — Метрики, матрица ошибок

Построим дерево решений (Рисунок 17).

```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
# Plot the decision tree
best_estimator = grid_search.best_estimator_
plt.figure(figsize=(10, 6))
plot_tree(best_estimator, feature_names=data.columns, class_names=['0', '1'], filled=True)
plt.show()
```

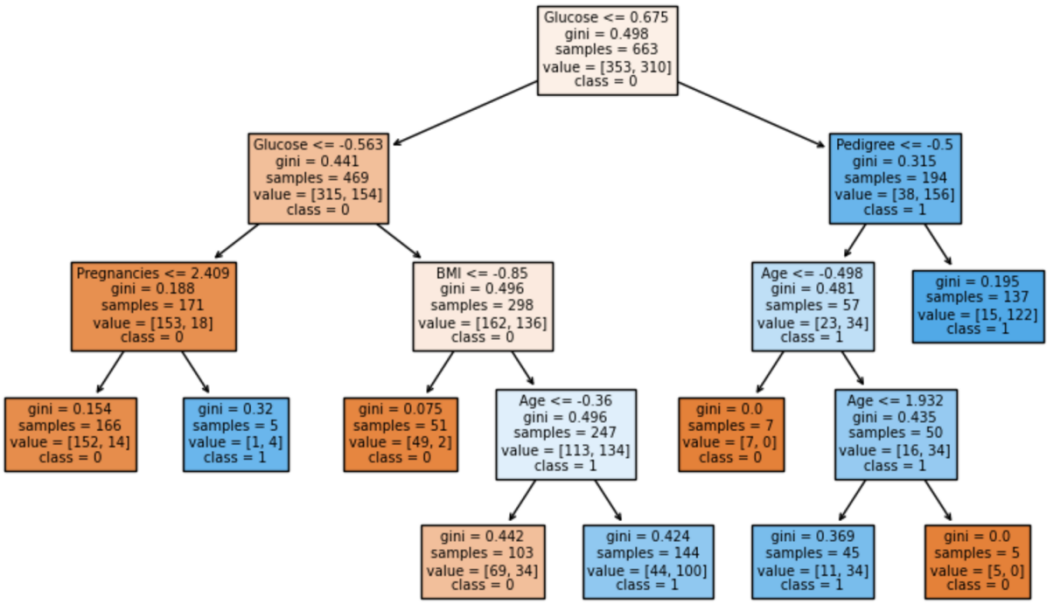

Рисунок 17 — Построение дерева решений

Теперь обучим ансамбль моделей (стекинг) (Рисунок 18).

Stacking

```python
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# Define the base models
base_models = [
    ('lr', LogisticRegression()),
    ('dt', DecisionTreeClassifier()),
    ('svm', SVC())
]

# Define the meta model
meta_model = DecisionTreeClassifier()

# Create the stacking classifier
stacking_classifier = StackingClassifier(estimators=base_models, final_estimator=meta_model)

# Fit the stacking classifier on the training data
stacking_classifier.fit(x_train, y_train)

# Predict the classes for the test data
y_pred = stacking_classifier.predict(x_test)

metr(y_pred, y_test)

entropy = shannon_entropy(y_pred)
print("Entropy:", entropy)

gin = gini(y_pred)
print('Index Gini:', gin)
```

Рисунок 18 — Обучение стекинг модели

Выведем метрики и матрицу ошибок для данной модели (Рисунок 19).

```
Accuracy: 0.421875
Precision: 0.4963768115942029
Recall: 0.49667774086378735
F1 Score: 0.4206019084903352
              precision    recall  f1-score   support

           0       0.67      0.28      0.39        86
           1       0.33      0.71      0.45        42

    accuracy                           0.42       128
   macro avg       0.50      0.50      0.42       128
weighted avg       0.55      0.42      0.41       128

[[24 62]
 [12 30]]
Entropy: 0.8571484374283715
Index Gini: 0.014128693485226057
```

Рисунок 19 — Метрики, матрица ошибок

Теперь обучим ансамбль моделей (бэггинг) (Рисунок 20).

Bagging

```python
from sklearn.ensemble import RandomForestClassifier

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model on the training data
rf_classifier.fit(x_train, y_train)

# Predict the classes for the test data
y_pred = rf_classifier.predict(x_test)

metr(y_pred, y_test)

entropy = shannon_entropy(y_pred)
print("Entropy:", entropy)

gin = gini(y_pred)
print('Index Gini:', gin)
```

Рисунок 20 — Обучение случайного леса

Выведем метрики и матрицу ошибок (Рисунок 21).

```
Accuracy: 0.3203125
Precision: 0.37131050767414403
Recall: 0.46982281284606864
F1 Score: 0.26538689887195727
              precision    recall  f1-score   support

           0       0.43      0.03      0.06        86
           1       0.31      0.90      0.47        42

    accuracy                           0.32       128
   macro avg       0.37      0.47      0.27       128
weighted avg       0.39      0.32      0.20       128

[[ 3 83]
 [ 4 38]]
Entropy: 0.3059848737138347
Index Gini: 0.014128693485226057
```

Рисунок 21 — Метрики, матрица ошибок

Построим случайный лес (Рисунок 22).

```
# Plot the decision tree
plt.figure(figsize=(10, 6))
plot_tree(rf_classifier.estimators_[0], feature_names=data.columns, class_names=['0', '1'], filled=True)
plt.show()
```

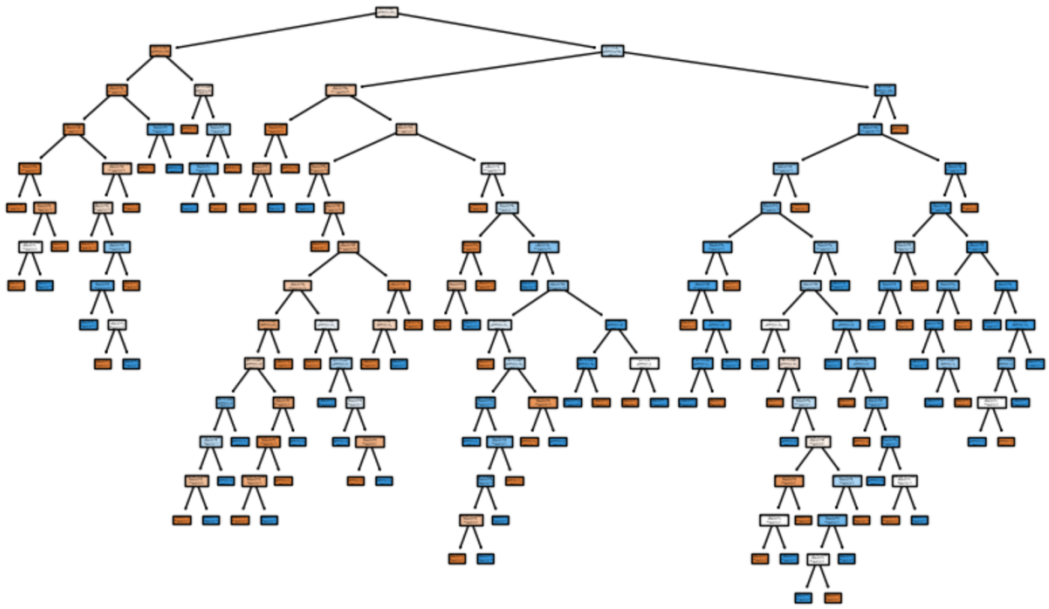

Рисунок 22 — Построение случайного леса

Теперь обучим ансамбль моделей (бустинг) (Рисунок 23).

Boosting

```
import xgboost as xgb
from xgboost import plot_tree
```

```
# Define the feature names
feature_names = x_train.columns
```

```
# Create an XGBoost classifier
clf = xgb.XGBClassifier(feature_names=feature_names)

# Fit the classifier on the data
clf.fit(x_train_boost, y_train_boost)
```

```
/Users/vladamalkina/opt/anaconda3/lib/python3.9/site-packages/xgboost/core.py:160: UserWarning: [1
/Users/runner/work/xgboost/xgboost/src/learner.cc:742:
Parameters: { "feature_names" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
              feature_names=RangeIndex(start=0, stop=8, step=1),
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

Рисунок 23 — Обучение бустинга

Выведем метрики и матрицу ошибок (Рисунок 24).

```
Accuracy: 0.71875
Precision: 0.6829004329004329
Recall: 0.6871539313399779
F1 Score: 0.6848153214774282
              precision    recall  f1-score   support

           0       0.80      0.78      0.79        86
           1       0.57      0.60      0.58        42

    accuracy                           0.72       128
   macro avg       0.68      0.69      0.68       128
weighted avg       0.72      0.72      0.72       128

[[67 19]
 [17 25]]
Entropy: 0.9283620723948676
Index Gini: 0.014128693485226057
```

Рисунок 24 — Метрики, матрица ошибок

Построим модель бустинга (Рисунок 25).

```
# Plot the first tree in the booster
plt.figure(figsize=(1600, 1000))
xgb.plot_tree(clf, num_trees=0)
plt.show()
```

```
<Figure size 160000x100000 with 0 Axes>
```

Рисунок 25 — Построение бустинга

Выводы

Модели	Энтропия Шеннона	Индекс Джинни	accuracy	F1-score
Дерево решений	0	0.01	0.32	0.25
Стекинг	0.85	0.01	0.42	0.42
Бэггинг	0.31	0.01	0.32	0.27
Бустинг	0.93	0.01	0.71	0.68

Мы научились решать задачу классификации используя различные ансамбли. Лучшие результаты классификации получились у бустинга. Вторая по эффективности модель – стекинг. Худшие результаты мы получили на дереве решений и бэггинге.