СОДЕРЖАНИЕ

Цель: обучить модели для прогноза результатов боев UFC с помощью методов машинного обучения машинного обучения

Задачи

1. Подготовить данные, содержащие информацию о результатах спортивных соревнований
2. Осуществить программную реализацию выбранных методов
3. Провести сравнительный анализ моделей и выбрать наилучшую модель.

ЗАГРУЗКА ДАННЫХ И ИХ ОЦЕНКА

Загрузка данных

Загрузим данные и проведем их анализ. Используем датасет содержащий в себе данные о турнирах UFC и их результатах. Он содержит в себе 137 предикторов. Эти предикторы имеет такие данные как, коэффициент ставки на определенных игроков, среднее значение прибыли от ставок на этих игроков в букмекерских конторах, возраст, рост, вес, количество побед подряд, количество побед и поражений, место рождения игрока, его пол и др. В последующем я выберу наиболее определяющие предикторы и на их основе построю прогнозирующие модели, которые будут определять победу определенного участника.

Загрузим библиотеки для работы с данными и сами данные и посмотрим, что они из себя представляют для дальнейшей работы. Также оценим количество побед каждой из сторон и размер обучающей выборки. У нас достаточно данных для обучения, и они примерно равномерно распределены для обучения.

```
Ввод [102]: ufc_data.head()
```

Out[102]:

	R_fighter	B_fighter	R_odds	B_odds	R_ev	B_ev	date	location	country	Winner	title_bout	weight_class	gender	no_of_rounds	B_cu
0	Paul Felder	Rafael Dos Anjos	165	-200	165.000000	50.000000	11/14/2020	Las Vegas, Nevada, USA	USA	0	False	Lightweight	MALE	5	
1	Abdul Razak Alhassan	Khaos Williams	-240	185	41.666667	185.000000	11/14/2020	Las Vegas, Nevada, USA	USA	0	False	Welterweight	MALE	3	
2	Kay Hansen	Cory McKenna	-230	180	43.478261	180.000000	11/14/2020	Las Vegas, Nevada, USA	USA	0	False	Women's Strawweight	FEMALE	3	
3	Brendan Allen	Sean Strickland	-118	-106	84.745763	94.339623	11/14/2020	Las Vegas, Nevada, USA	USA	0	False	Catch Weight	MALE	3	
4	Ashley Yoder	Miranda Granger	135	-167	135.000000	59.880240	11/14/2020	Las Vegas, Nevada, USA	USA	1	False	Women's Strawweight	FEMALE	3	

Рисунок – 6 Входные данные

Листинг

```
from sklearn import tree
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
%matplotlib inline
```

```
from sklearn.tree import plot_tree
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV

import sys, warnings, os

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import OneHotEncoder


#from graphviz  import Source
from IPython.display import SVG, display, HTML
style = "<style>svg{width: 70% !important; height: 60% !important;} </style>"


ufc_data = pd.read_csv(r"C:\Users\User\Downloads\ufc-master2.csv")
ufc_data['Winner'].value_counts()
```

Оценка корреляции

Необходимо определить корреляцию между входными данными и выходной переменной Winner для дальнейшего анализа нашего набора данных. Нужно отбросить входные данные с низкой корреляцией и оставить только более значимые, чтобы не засорять дерево. Для этого используем встроенную библиотек sklearn[6].

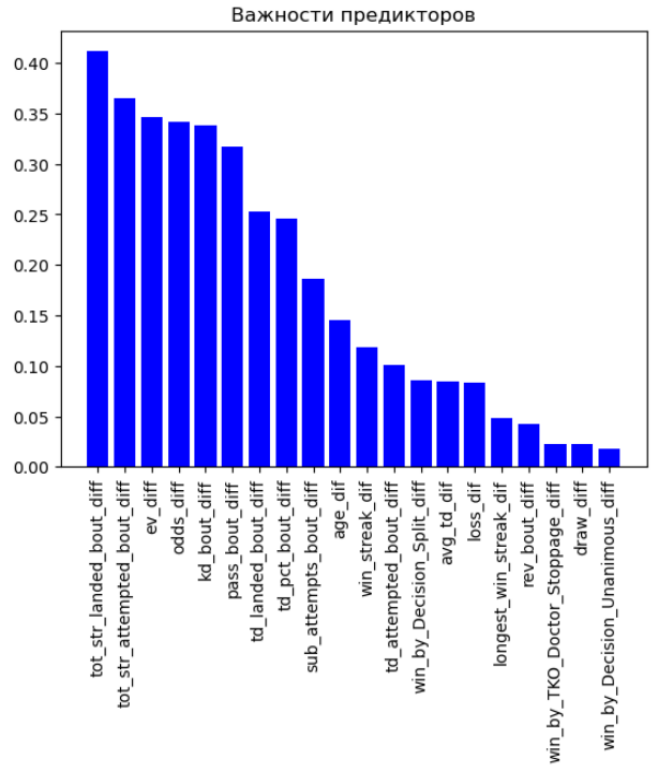

Рисунок – 7 Оценка корреляции

Из рисунка можно видеть, что некоторые предикторы имеют низкую оценку важности. Наиболее важными предикторы здесь будут tot_str_landen_bout_diff, tot_str_attempted_bout_diff, ev_diff, odds_diff и др. Или же среднее количество удачных ударов за бой, среднее количество предпринятых ударов за бой, среднее значение прибыли от ставки, средняя ставка на бойца, и др. Дальнейшем я исключу предикторы, которые не оказывают влияния на предсказательную способность.

Листинг

```
num_corr = [col for col in X_train.columns if X_train[col].dtype == 'int64' or X_train[col].dtype == 'float64']
corr_ = {}
for col in num_corr:
    corr_[col] = abs(X_train[col].corr(ufc_data['Winner']))
array = np.zeros(20, dtype=float)
i = 0
for w in sorted(corr_, key=corr_.get, reverse = True):
  print(w, corr_[w])
  if(i>20):
```

```
        break
    array[i] = corr_[w]
    i=i+1
indices = np.argsort(array)[::-1]
df_in = pd.DataFrame.from_dict(corr_, orient='index', columns=['A'])
df_in = df_in.T
df_in = df_in.sort_values(by='A', ascending=False)
feat_labels = df_in.columns
plt.title('Важности предикторов')
plt.bar(range(X_train.shape[1]), array[indices], color='blue', align='center')
plt.xticks(range(X_train.shape[1]), feat_labels[indices], rotation = 90)
```

Визуальный анализ данных

Построим диаграммы рассеивания. И визуализируем данные с помощью графика рассеивания. Для построения графика рассеивания выберем два важных предиктора B_odds и R_odds. Эти предикторы содержат в себе ставки на определенных игроков.

На диаграммах рассеяния ряд точек, размещенных в декартовой системе координат, отображает значения по двум переменным. Присвоив каждой оси переменную, можно определить, существуют ли отношения или корреляция между этими двумя переменными.

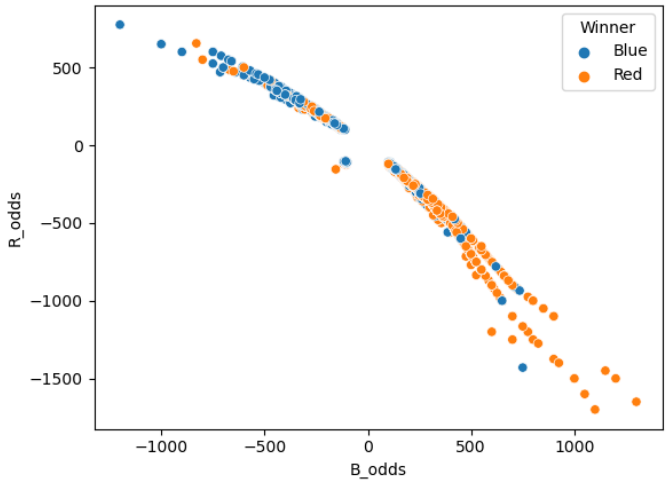

Рисунок – 8 График зависимости предикторов R_odds и B_odds

Можно наблюдать что, по мере увеличения B_odds “красных” победителей становится больше. Тоже самое и для “синих” победителей, за некоторым исключением. Из этого можно сделать вывод, что все победители это те на которых делается ставка.

Также похожий результат покажет запрос подсчёта победителей при значении B_odds больше единицы. Количество “красных” победителей растет с увеличением параметра B_odds.

```
Ввод [12]: ufc_data["Winner"].loc[ufc_data["B_odds"]>1].value_counts()

Out[12]: Red     1906
         Blue     846
         Name: Winner, dtype: int64
```

Рисунок – 9 График подсчёта победителей при B_odds > 1

Проанализируем зависимость между предикторами B_ev и R_ev. Эти предикторы содержат в себе среднее значение прибыли от ставок на этих игроков в букмекерских конторах.

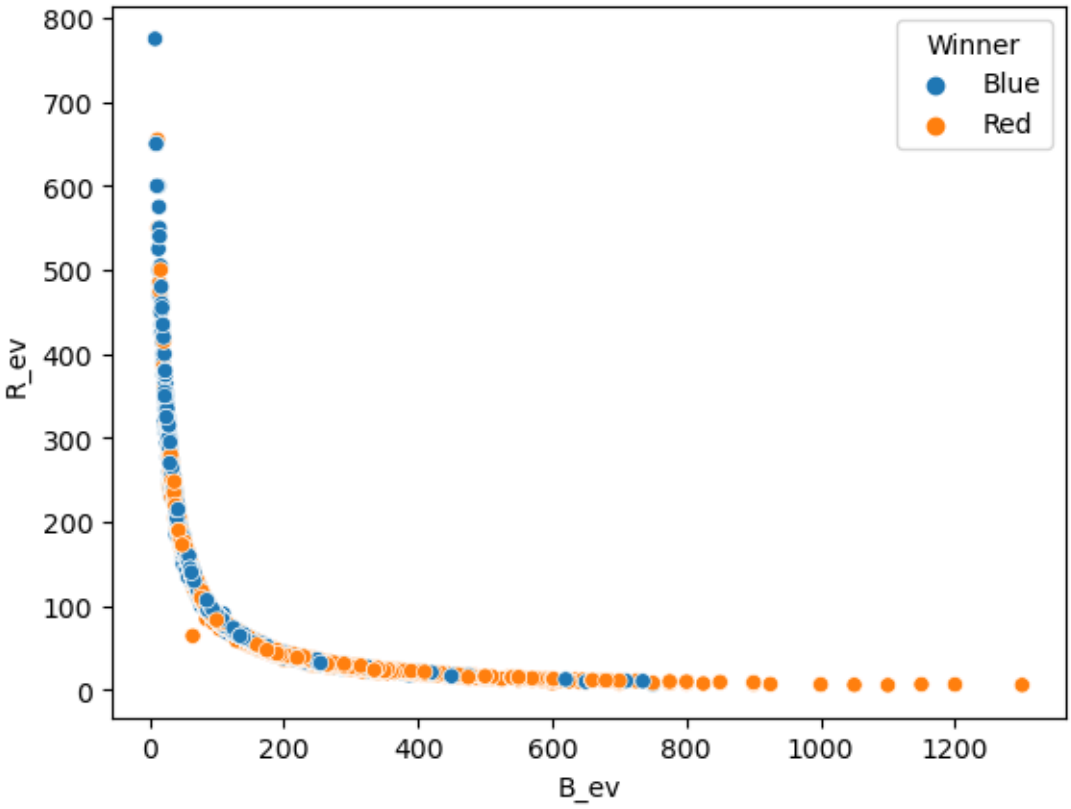

Рисунок – 10 График зависимости предикторов R_ev и B_ev

Заметно что, с увеличением предиктора B_ev увеличивается шанс на победу “красных” игроков и наоборот, с ростом R_ev увеличатся число “синих”

Посмотрим, как влияет стойка игроков на их победу. Для этого построим столбчатую диаграмму.

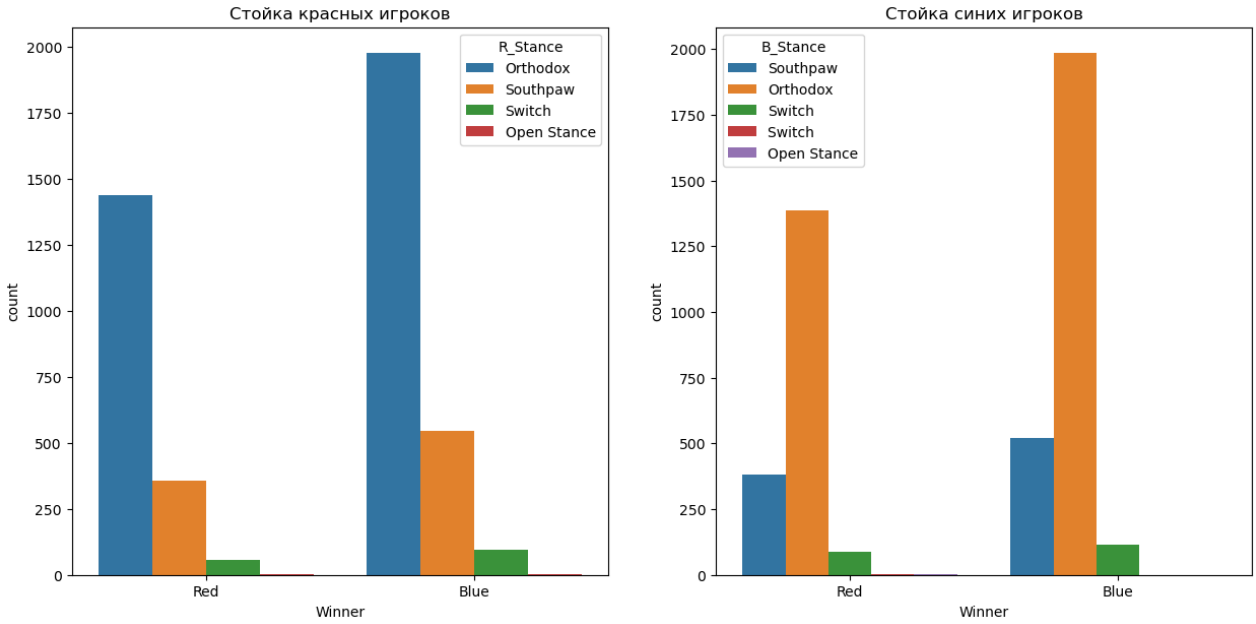

Рисунок – 11 Столбчатые диаграммы стоек

Заметно, что левосторонняя стойка имеет наибольшее преимущество перед другими стойками.

Проанализируем зависимость между средним значением удачных ударов и средним значением предпринятых ударов.

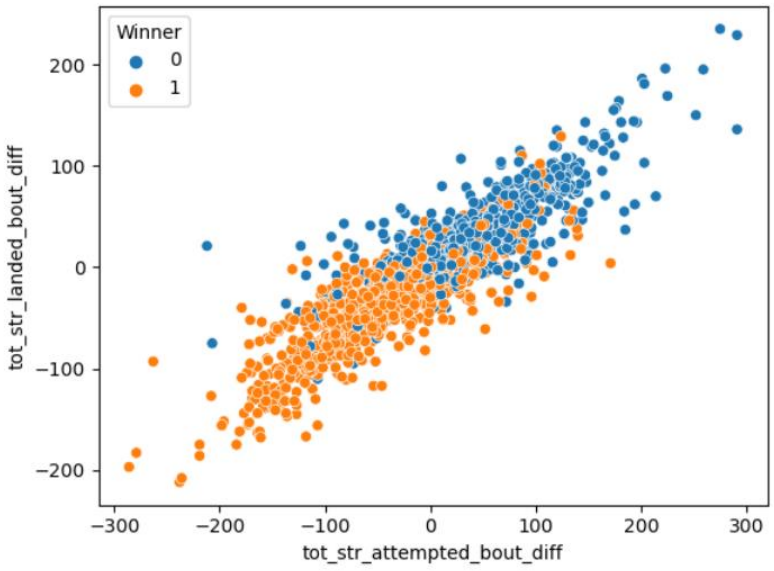

Рисунок – 12 График зависимости среднего значения удачных ударов и среднего значения предпринятых ударов

Можно заметить за некоторым исключением, что при росте среднего значения удачных ударов и среднего значения предпринятых ударов возрастает вероятность победы “синего” бойца. И наоборот при уменьшении этих предикторов увеличивается шанс победы “красного” бойца. Другими словами, чем больше боец делает удачных ударов и пытается провести удар, тем больше вероятность его победы. Стоит отметить, что эти параметры представляют собой разницу между средними значениями каждых игроков. И если у параметра значение положительное, то оно идёт в пользу “синего” игрока, и, если отрицательное, то идёт в пользу “красного” игрока.

Листинг

```
sns.scatterplot(x="B_odds", y="R_odds", hue="Winner", data = ufc_data);
sns.scatterplot(x="tot_str_attempted_bout_diff", y="tot_str_landed_bout_diff", hue="Winner", data = ufc_data);
ufc_data["Winner"].loc[ufc_data["B_odds"]>1].value_counts()
sns.scatterplot(x='B_ev', y='R_ev',hue = 'Winner', data=ufc_data);
fig, ax = plt.subplots(1,2, figsize=(15,7))
sns.countplot(ufc_data['Winner'], hue = ufc_data['R_Stance'], ax=ax[0])
sns.countplot(ufc_data['Winner'], hue = ufc_data['B_Stance'], ax=ax[1])
ax[0].title.set_text('Стойка красных игроков')
ax[1].title.set_text('Стойка синих игроков')
fig.show()
```

Подготовка датасета к машинному обучению

Подготовим данные к машинному обучению. Для этого одинаковые численные предикторы, которые содержатся у каждого бойца отнимем друг от друга. После этого удалим столбцы, в которых содержатся одинаковые предикторы. Также отбросим предикторы с низкой корреляцией и оставить только более значимые. Помимо этого, необходимо преобразовать категориальные переменные в подходящую форму для машинного обучения.

Листинг

```
ufc_data['draw_diff'] = ( ufc_data['B_draw'] - ufc_data['R_draw'])
ufc_data['avg_sig_str_pct_diff'] = (ufc_data['B_avg_SIG_STR_pct']-
ufc_data['R_avg_SIG_STR_pct'])
ufc_data['avg_TD_pct_diff'] = ( ufc_data['B_avg_TD_pct']- ufc_data['B_avg_TD_pct'])
ufc_data['win_by_Decision_Majority_diff'] = (ufc_data['B_win_by_Decision_Majority']-
ufc_data['R_win_by_Decision_Majority'])
ufc_data['win_by_Decision_Split_diff'] = (ufc_data['B_win_by_Decision_Split']-
ufc_data['R_win_by_Decision_Split'])
ufc_data['win_by_Decision_Unanimous_diff'] =
(ufc_data['B_win_by_Decision_Unanimous']-
ufc_data['R_win_by_Decision_Unanimous'])
ufc_data['win_by_TKO_Doctor_Stoppage_diff'] =
(ufc_data['B_win_by_TKO_Doctor_Stoppage']-
ufc_data['R_win_by_TKO_Doctor_Stoppage'])
ufc_data['odds_diff'] = (ufc_data['B_odds'] - ufc_data['R_odds'])
ufc_data['ev_diff'] = (ufc_data['B_ev']-ufc_data['R_ev'])
ufc_data['kd_bout_diff']=(ufc_data['B_kd_bout']-ufc_data['R_kd_bout'])
ufc_data['sig_str_landed_bout_diff']=(ufc_data['B_sig_str_landed_bout']-
ufc_data['R_sig_str_landed_bout'])
ufc_data['sig_str_attempted_bout_diff']=(ufc_data['B_sig_str_attempted_bout']-
ufc_data['R_sig_str_attempted_bout'])
ufc_data['sig_str_attempted_bout_diff']=(ufc_data['B_sig_str_attempted_bout']-
ufc_data['R_sig_str_attempted_bout'])
ufc_data['sig_str_pct_bout_diff']=(ufc_data['B_sig_str_pct_bout']-
ufc_data['R_sig_str_pct_bout'])
ufc_data['tot_str_landed_bout_diff']=(ufc_data['B_tot_str_landed_bout']-
ufc_data['R_tot_str_landed_bout'])
ufc_data['tot_str_attempted_bout_diff']=(ufc_data['B_tot_str_attempted_bout']-
ufc_data['R_tot_str_attempted_bout'])
```

```
ufc_data['td_landed_bout_diff']=(ufc_data['B_td_landed_bout']-
ufc_data['R_td_landed_bout'])
ufc_data['td_attempted_bout_diff']=(ufc_data['B_td_attempted_bout']-
ufc_data['R_td_attempted_bout'])
ufc_data['td_pct_bout_diff']=(ufc_data['B_td_pct_bout']-ufc_data['R_td_pct_bout'])
ufc_data['td_pct_bout_diff']=(ufc_data['B_td_pct_bout']-ufc_data['R_td_pct_bout'])
ufc_data['sub_attempts_bout_diff']=(ufc_data['B_sub_attempts_bout']-
ufc_data['R_sub_attempts_bout'])
ufc_data['pass_bout_diff']=(ufc_data['B_pass_bout']-ufc_data['R_pass_bout'])
ufc_data['rev_bout_diff']=(ufc_data['B_rev_bout']-ufc_data['R_rev_bout'])

var_drop = [
'B_odds', 'R_odds', 'B_ev', 'R_ev', 'R_kd_bout', 'B_kd_bout', 'R_sig_str_landed_bout',
'B_sig_str_landed_bout', 'R_sig_str_attempted_bout', 'B_sig_str_attempted_bout',
'R_sig_str_pct_bout', 'B_sig_str_pct_bout', 'R_tot_str_landed_bout',
'B_tot_str_landed_bout', 'R_tot_str_attempted_bout', 'B_tot_str_attempted_bout',
'R_td_landed_bout', 'B_td_landed_bout', 'R_td_attempted_bout', 'B_td_attempted_bout',
'R_td_pct_bout', 'B_td_pct_bout', 'R_sub_attempts_bout', 'B_sub_attempts_bout',
'R_pass_bout', 'B_pass_bout', 'R_rev_bout', 'B_rev_bout', 'B_current_lose_streak',
'R_current_lose_streak', 'B_current_win_streak', 'R_current_win_streak',
'B_longest_win_streak', 'R_longest_win_streak', 'B_wins', 'R_wins', 'B_losses', 'R_losses',
'B_total_rounds_fought', 'R_total_rounds_fought', 'B_total_title_bouts',
'R_total_title_bouts', 'B_win_by_KO/TKO', 'R_win_by_KO/TKO',
'B_win_by_Submission', 'R_win_by_Submission', 'B_Height_cms', 'R_Height_cms',
'B_Reach_cms', 'R_Reach_cms', 'B_age', 'R_age', 'B_avg_SIG_STR_landed',
'R_avg_SIG_STR_landed', 'B_avg_SUB_ATT', 'R_avg_SUB_ATT', 'B_avg_TD_landed',
'R_avg_TD_landed',
'B_draw','B_avg_SIG_STR_pct','B_avg_TD_pct','B_win_by_Decision_Majority','B_win_
by_Decision_Split','B_win_by_Decision_Unanmous','B_win_by_TKO_Doctor_Stoppage',
```

```
'R_draw','R_avg_SIG_STR_pct','R_avg_TD_pct','R_win_by_Decision_Majority','R_win_
by_Decision_Split','R_win_by_Decision_Unamous','R_win_by_TKO_Doctor_Stoppage']
ufc_data.drop(var_drop, axis=1, inplace = True)

comm_drop = [
'date','location','country','weight_class','gender','no_of_rounds','empty_arena','constant_1','f
inish','finish_details','finish_round','finish_round_time','total_fight_time_secs','B_Weight_l
bs','R_Weight_lbs']
ufc_data.drop(comm_drop, axis=1, inplace = True)

ufc_data.loc[:,'B_match_weightclass_rank':'better_rank'].isnull().sum()
ufc_data.drop(ufc_data.loc[:,'B_match_weightclass_rank':'B_Pound-for-Pound_rank'],
axis=1, inplace = True)

cat_co = ['R_Stance', 'B_Stance', 'title_bout', 'better_rank']
one_hot_coder = OneHotEncoder()
t = one_hot_coder.fit_transform(ufc_data[cat_co]).toarray()
encodings = pd.DataFrame(columns = one_hot_coder.get_feature_names_out(), data = t)
encodings = encodings.astype(int)
ufc_data = pd.concat([ufc_data, encodings ], axis = 1)
ufc_data.drop(categorical_columns, axis = 1, inplace = True)

from sklearn.preprocessing import LabelEncoder
cat_col = ['R_fighter', 'B_fighter']
enc = LabelEncoder()
for i in ufc_data[cat_col]:
    ufc_data[i] = enc.fit_transform(ufc_data[i])
```

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

Программная реализация дерева решений

Алгоритм построения дерева решений

Основное действие при построении дерева решений заключается в последовательном и рекурсивном разбиении. Этот процесс продолжается до тех пор, пока все узлы на концах ветвей не станут листьями.

Узел становится листом в двух случаях:

- естественно - когда он содержит один объект или объект только одного класса;
- после достижения заданного условия остановки алгоритм, такой как минимально допустимое количество экземпляров в узле или максимальная глубина дерева.

Построение основано на «жадных» алгоритмах, которые допускают локально оптимальные решения на каждом шаге (разбиения в узлах), что приводит к окончательному оптимальному решению. То есть, когда один атрибут выбран и разделен на подмножества, алгоритм не может вернуться назад и выбрать другой атрибут, даже если это дает лучший окончательный раздел. Поэтому на этапе построения дерева решений нельзя однозначно сказать, удастся ли получить оптимальное разбиение.

Теоретико-информативный критерий основан на информационной энтропии:

$$H = -\sum_{i=1}^{n} \frac{N_i}{N} \log(\frac{N_i}{N})$$

Основные этапы построения дерева решение, Строительство ведется в 4 этапа:

1 Выберите атрибут для разделения в этом узле.

2 Определите критерий прекращения обучения.

3 Выбрать метод обрезки.

4 Оценить точность построенного дерева.

Итак, рассмотрим каждую подробнее.

Энтропия рассматривается как мера неоднородности подмножества с точки зрения представляемых им классов. И хотя классы представлены в равных пропорциях и неопределенность классификации наибольшая, энтропия также наибольшая. Логарифм единицы преобразует энтропию в ноль, если все экземпляры узла относятся к одному классу [8].

Если выбранный атрибут раздела A_j обеспечивает наибольшее снижение энтропии результирующего подмножества по отношению к его родителю, его можно считать лучшим.

Но на самом деле энтропия упоминается редко. Эксперты обращают внимание на информацию, получающуюся в процессе разбиения предикторов. В этом случае лучшим атрибутом будет тот, который обеспечивает максимальный информационный прирост результирующего узла по сравнению с исходным:

$$Gain(A) = Info(S) - Info(S_A)$$

где $Info(S)$ — информация, связанная с подмножеством S до разделения, $Info(S_A)$ — информация, связанная с подмножеством, полученным во время разделения атрибута A.

Задача выбора атрибута в такой ситуации состоит в том, чтобы максимизировать значение выигрыша (A), которое называется информационным выигрышем. Поэтому информационно-теоретический подход также известен как «критерий получения информации».

Критерий остановки алгоритма

Алгоритм обучения может работать до «чистых» подмножеств с примерами того же класса. В этом случае вы, скорее всего, получите дерево, в котором для каждого примера будет создан отдельный лист. Такое дерево нельзя применять на практике из-за перетренированности. У каждого экземпляра будет свой уникальный путь в дереве. Результатом является набор правил, относящихся только к этому примеру.

Специалисты решают принудительно остановить строительство дерева, чтобы оно не «перетренировалось».

Отсечение ветвей

Без ограничения «роста» дерево решений станет слишком большим и сложным, что сделает невозможной дальнейшую интерпретацию. А если создать решающие правила создания узлов, которые будут включать 2-3 примера, они не потеряют своей практической ценности.

Поэтому многие специалисты предпочитают альтернативный вариант: построить все возможные деревья, а затем выбрать те, которые на разумной глубине обеспечивают приемлемый уровень ошибки распознавания. Основная задача в такой ситуации — найти наиболее выгодный баланс между сложностью и точностью дерева.

Но и здесь есть проблема: такая задача относится к классу NP-полных задач, а эффективных решений они, как известно, не имеют. По этой причине прибегают к методу обрезки ветвей, который реализуется в 3 шага:

Построение полного дерева, в котором листья содержат экземпляры одного класса.

Определение двух показателей: относительной точности модели (отношение количества правильно распознанных примеров к общему количеству примеров) и абсолютной ошибки (количества неправильно классифицированных примеров).

Исключение листьев и узлов, потеря которых минимально повлияет на точность модели и увеличит погрешность.

Обрезку ветвей производят в обратном порядке росту дерева, то есть снизу-вверх, последовательно превращая узлы в листья [13].

Построение дерева решений

Разделим наши данные в отношении один к трём для обучения и тестирования. Также в нашем датасете есть данные с пропущенными значениями. Для их заполнения воспользуемся функцией, которая заполняет эти пропущенные значения используя стратегию с самыми часто встречаемыми значениями. И наконец-то посадим, и вырастим наше дерево.

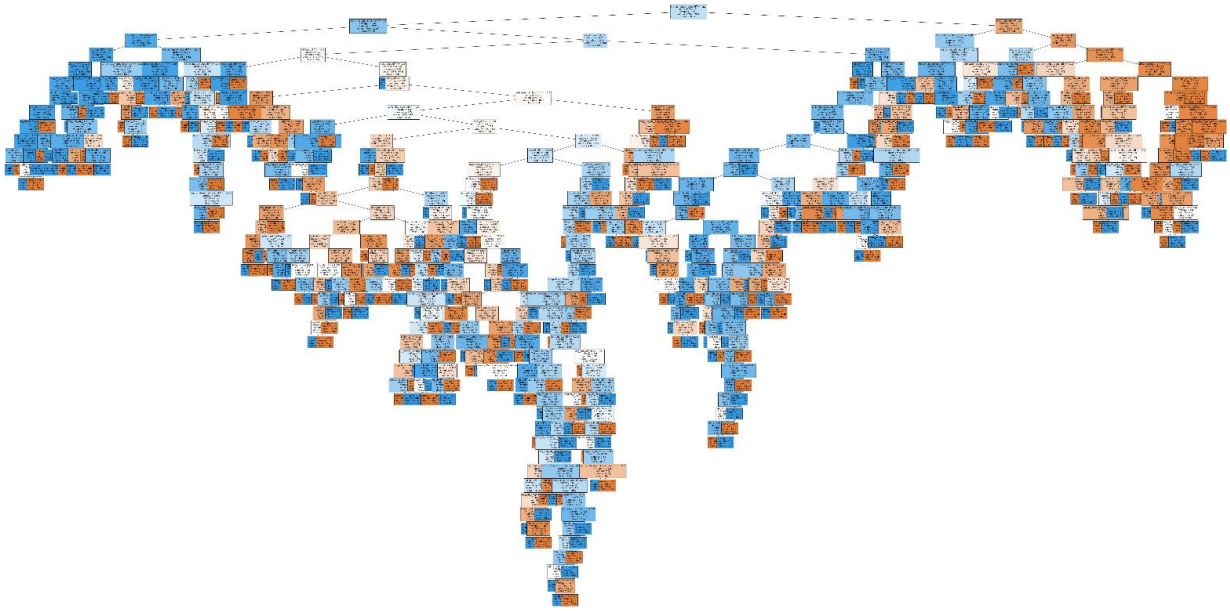

Рисунок – 14 Дерево классификации

Листинг

```
X_train, X_valid, y_train, y_valid = train_test_split(ufc_data, label, random_state = 0, test_size = 0.3)
X_train.isnull().sum().sort_values(ascending=False)
cat_col = [col for col in X_train.columns if X_train[col].dtypes == 'object']
num_col = [col for col in X_train.columns if col not in cat_col]
imp = SimpleImputer(strategy='most_frequent')
imp.fit(X_train[num_col])
X_train[num_col] = imp.transform(X_train[num_col])
X_valid[num_col] = imp.transform(X_valid[num_col])
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
plt.figure(figsize=(63, 20))
tree.plot_tree(clf, fontsize=10, feature_names=list(ufc_data), filled=True, class_names = ['Red','Blue'])
```

Подбор параметров дерева решений для повышения предсказательной точности

Можно наблюдать что наше дерево переобучилось и нам необходимо самостоятельно подобрать параметры дерева классификаций. Для этого напишем

скрипт, который при выбранном нами параметре будет записывать в датафрейм точность тестовой, кросс-валидационной и обучающей выборки. И выберем наилучшие параметры исходя из наилучшей точности тестовой и кросс-валидационной выборки.

Просмотрим всевозможные точности нашей модели при глубинах дерева от 1 до 100:

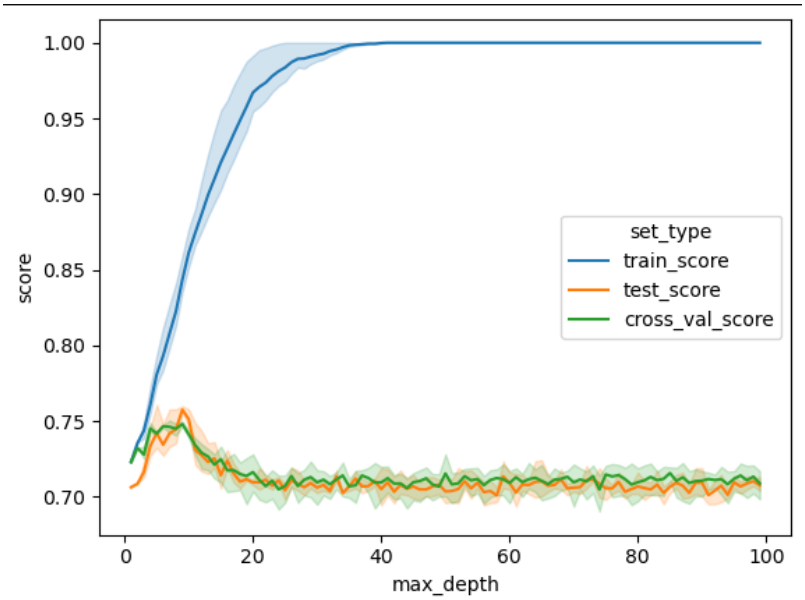

Рисунок – 15 Зависимость точности от глубины дерева

Можно видеть, что наилучшая точность получается при глубине дерева равной 9. Воспользуемся готовой функцией чтобы проверить это утверждение:

```
Ввод [86]: grid_search_cv_clf.best_params_
Out[86]: {'criterion': 'entropy', 'max_depth': 8}
```

Рисунок – 16 Наилучшие параметры глубины дерева

Готовая функция предсказала близкое значение равное 8 и можно сделать вывод, что наша функция работает корректно.

Теперь посмотрим всевозможные точности нашей модели при минимально возможном значении разделения выборки от 1 до 20:

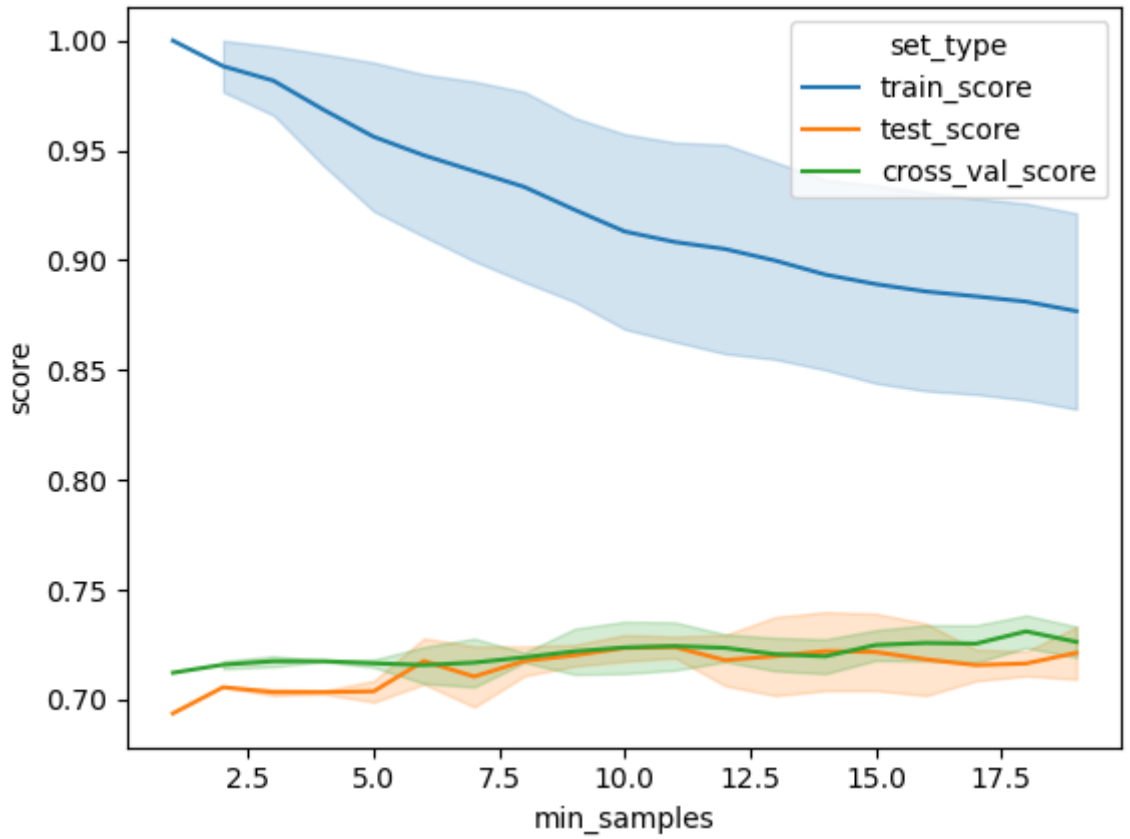

Рисунок – 17 Зависимость точности от разделения выборки

Особой разницы при разных значения разбиения выборки нет.

```
Ввод [107]: grid_search_cv_clf.best_params_
Out[107]: {'criterion': 'entropy', 'max_depth': 8, 'min_samples_split': 2}
```

Рисунок – 18 Зависимость точности от разделения выборки

Тот же самый результат показывает готовая функция по поиску наилучших параметров

Попробуем вырастить теперь наше дерево с наилучшими параметрами, подобранными нашей функцией:

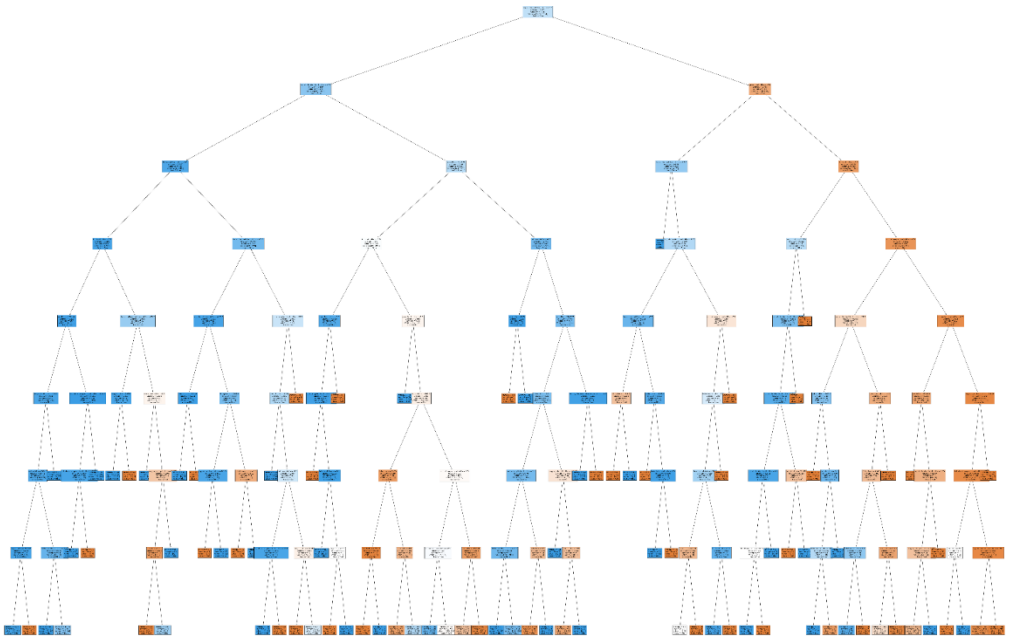

Рисунок – 19 Обученное дерево классификаций

Теперь наше дерево выглядит гораздо лучше и обладает наилучшей предсказательной способностью.

Листинг

```
scores_data = pd.DataFrame()
for max_depth in max_depth_values:
    clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=max_depth)
    clf.fit(X_train, y_train)
    train_score = clf.score(X_train, y_train)
    test_score = clf.score(X_valid, y_valid)
    mean_cross_val_score = cross_val_score(clf, X_train, y_train, cv=5).mean()
    temp_score_data = pd.DataFrame({'max_depth': [max_depth],
                                    'train_score': [train_score],
                                    'test_score': [test_score],
                                    'cross_val_score': [mean_cross_val_score]})
    scores_data = pd.concat([scores_data, temp_score_data])
scores_data_long = pd.melt(scores_data, id_vars = ['max_depth'], value_vars = ['train_score','test_score', 'cross_val_score'], var_name = 'set_type', value_name = 'score')
sns.lineplot(x='max_depth', y='score', hue='set_type', data=scores_data_long)
```

clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth = 8, min_samples_split = 2)

plt.figure(figsize=(100, 70))

tree.plot_tree(clf, fontsize=10, feature_names=list(ufc_data), filled=True, class_names = ['Red','Blue'])

Программная реализация логистической регрессии

Алгоритм логистической регрессии

Есть несколько способов найти коэффициенты логистической регрессии. На практике часто используется метод максимального правдоподобия. Он используется в статистике для получения более качественных параметров генеральной совокупности из выборочных данных. В основе метода лежит функция правдоподобия, выражающая плотность вероятности (вероятность) совместного появления результатов выборки

$$L(Y_1, Y_2, \dots Y_k; \theta) = p(Y_1; \theta) \cdot \dots \cdot p(Y_1; \theta)$$

Согласно методу максимального правдоподобия в качестве оценки неизвестного параметра θ принимается следующее значение $\theta = \theta(\mathrm{Y}_1, \mathrm{Y}_2, \dots \mathrm{Y}_\mathrm{k})$, которое максимизирует функцию L.

Нахождение оценки упрощается, если сама функция не максимизируется L ,а максимизируется ее натуральный логарифм $ln(L)$, так как максимум обеих функций достигается при одном и том же значении θ:

$$L(Y; \theta) = ln(L(Y; \theta)) \rightarrow max,$$

В простейшем случае минимальное количество времени, чтобы мой язык в логистической регрессии был высокопроизводителен. Обозначим через $P_i = Prob(Y_i = 1)$. Эта вероятность зависит от $X_i W$, где X_i строка матрицы регрессоров, W вектор коэффициентов регрессии:

$$P_i = F(X_i W), F(z) = \frac{1}{1 + e^{-z}}$$

Логарифмическая функция правдоподобия равна:

$$L^* = \sum_{i=1}^{k} [Y_i \, ln \, P_i \, (W) + (1 - Y_i) ln(1 - ln \, P_i \, (W))]$$

Где I_0, $I_1 -$ множества наблюдений, для которых $Y_i = 0$, $Y_i = 1$ соответственно.

Таким образом мы решаем задачу оптимизации функции. Для расчета коэффициентов логистической регрессии можно применять любые градиентные методы, например метод сопряженных градиентов.

Логистическую регрессию можно представить в виде однослойной нейронной сети с сигмоидальной функцией активации, веса которой есть коэффициенты логистической регрессии, а вес поляризации — константа регрессионного уравнения [15].

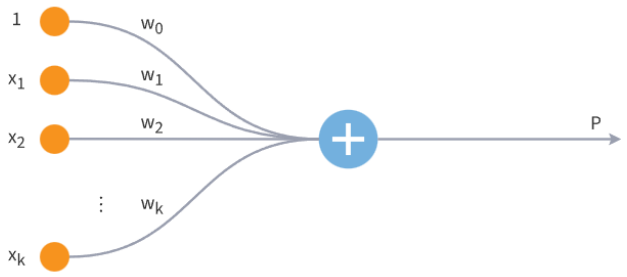

Рисунок 21 — Представление логистической регрессии в виде нейронной сети

Программная реализация логистической регрессии

Построим модель логистической регрессии используя библиотеки sklearn в Jupyter notebook. Для этого подключим класс логистической регрессии используя инструментарий sklearn и создадим объект класса и сразу же обучим его, используя метод из этого класса:

```
In [31]: from sklearn.linear_model import LogisticRegression

In [32]: LogR = LogisticRegression().fit(X_train, y_train)
```

Рис. 22 — Подключение логистической

Оценим точность обученной мною модели используя метрику точности на обучающих и тестовых данных:

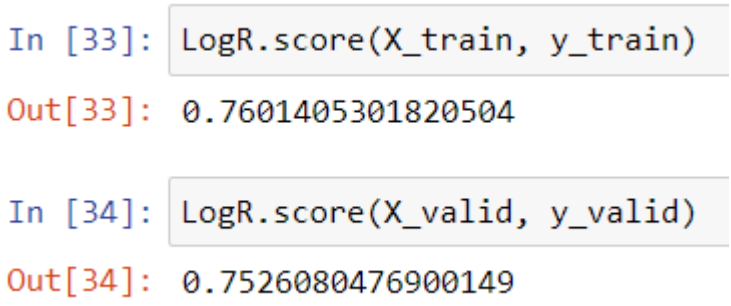

Рис. 23 — Точность на обучающих и тестовых данных

Можно наблюдать что 75 из 100 объектов классифицируется верно. Попробуем подобрать наиболее оптимальные параметры нашей модели для повышения качества модели и увеличить качество прогнозирующей способности.

Листинг

```
from sklearn.linear_model import LogisticRegression
LogR = LogisticRegression().fit(X_train, y_train)
LogR.score(X_train, y_train)
LogR.score(X_valid, y_valid)
```

Подбор параметров логистической регрессии для повышения предсказательной точности

Воспользуемся написанному ранее мной скриптом, отредактировав его для задачи логистической регрессии. И выберем наилучшие параметры исходя из наилучшей точности тестовой и кросс-валидационной выборки. Просмотрим всевозможные точности нашей модели при количестве итераций от 1 до 400, необходимых для сходимости решателей:

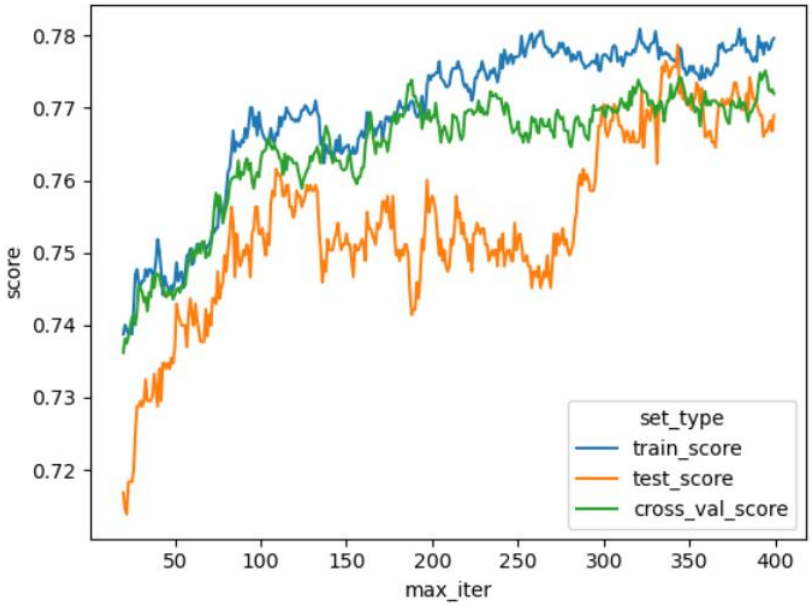

Рис. 24 — Зависимость точности от количества итераций

Сказать определенно при какой точности достигается наилучшая точность из этого проблематично, поэтому из этого датафрейма воспользуюсь библиотекой pandas найдем при скольких итерациях достигается наибольшее значение точности. Наилучшая точность достигается при 394 итерациях.

Out[96]:

	max_iter	set_type	score
1134	394	cross_val_score	0.77515

Рис. 25 — Наибольшая точность в зависимости от итераций

Воспользуемся функцией для подбора лучшего алгоритма, обеспечивающего наивысшую точность.

```
In [46]: parametrs={'max_iter': range(394, 400, 2), 'solver': ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']}

In [47]: grid_search_cv_clf = GridSearchCV(LogR, parametrs, cv=5)

In [48]: grid_search_cv_clf.fit(X_train, y_train)

Out[48]: GridSearchCV
         estimator: LogisticRegression
             LogisticRegression

In [50]: grid_search_cv_clf.best_params_

Out[50]: {'max_iter': 394, 'solver': 'newton-cg'}
```

Рис. 26 — Функция, подбирающая наилучшую точность в зависимости от алгоритма

Можно видеть, что наилучшим решателем в нашем случае будет метод ньютона.

Листинг

```
scores_data = pd.DataFrame()
max_iter_values = range(20, 400)
for max_iter in max_iter_values:
    LogR = LogisticRegression(max_iter=max_iter, random_state=152)
    LogR.fit(X_train, y_train)
    train_score = LogR.score(X_train, y_train)
    test_score = LogR.score(X_valid, y_valid)
    mean_cross_val_score = cross_val_score(LogR, X_train, y_train, cv=5).mean()
    temp_score_data = pd.DataFrame({'max_iter': [max_iter],
                                    'train_score': [train_score],
                                    'test_score': [test_score],
                                    'cross_val_score': [mean_cross_val_score]})
    scores_data = pd.concat([scores_data, temp_score_data])
scores_data_long = pd.melt(scores_data, id_vars = ['max_iter'], value_vars = ['train_score','test_score', 'cross_val_score'], var_name = 'set_type', value_name = 'score')
sns.lineplot(x='max_iter', y='score', hue='set_type', data=scores_data_long)
scores_data_long.loc[scores_data_long['set_type'] == 'cross_val_score'].loc[scores_data_long['score'] == scores_data_long.loc[scores_data_long['set_type'] == 'cross_val_score'].score.max()]
parametrs={'max_iter': range(394, 400, 2), 'solver': ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']}
grid_search_cv_clf = GridSearchCV(LogR, parametrs, cv=5)
grid_search_cv_clf.fit(X_train, y_train)
grid_search_cv_clf.best_params_
```

Программная реализация метода k-ближайших соседей

Алгоритм метода k-ближайших соседей

Алгоритм k-ближайших соседей состоит из двух этапов: обучения и классификации. На этапе обучения алгоритм запоминает векторы признаков и метки классов наблюдений, а также задается параметр k, который определяет количество

ближайших соседей для использования в классификации. На этапе классификации алгоритм определяет к какому классу отнести новый объект, выбирая класс, у которого больше всего k ближайших предварительно выбранных наблюдений. Важность примеров для определения класса зависит от их удаленности от классифицируемого объекта, поэтому используется взвешенное голосование. Чем дальше пример от классифицируемого объекта в пространстве признаков, тем меньше его значимость для определения класса.

В методе взвешенных k-ближайших соседей используется идея введения штрафа для класса, который зависит от расстояния между связанными примерами и классифицируемым объектом. Этот штраф выражается в виде суммы обратных квадратов расстояний от примеров j-го класса до классифицируемого объекта. Чем дальше находятся связанные примеры от объекта, тем больше штраф и меньше вес класса при вычислении взвешенной оценки. Это позволяет учитывать близость примеров к классифицируемому объекту при принятии решения о классификации:

$$Q_i = \sum_{i=1}^{n_i} \frac{1}{D^2(x, a_{ij})}$$

где D — оператор вычисления расстояния, x — вектор признаков классифицируемого объекта, a_{ij} — й пример j-го класса. Таким образом, «побеждает» тот класс, для которого величина Q_i окажется наибольшей. При этом также снижается вероятность того, что классы получат одинаковое число голосов [14].

Программная реализация метода k-ближайших соседей

Построим модель k-ближайших соседей используя библиотеки sklearn в Jupyter notebook. Также подключим класс метода k-ближайших соседей используя инструментарий sklearn и создадим объект класса и обучим его, используя метод из этого класса:

```
In [71]: from sklearn.neighbors import KNeighborsClassifier

In [76]: neigh = KNeighborsClassifier()

In [77]: neigh.fit(X_train, y_train)
Out[77]: KNeighborsClassifier
         KNeighborsClassifier()
```

Рис. 27 — Подключение логистической

Оценим точность обученной модели используя метрику точности на обучающих и тестовых данных:

```
In [78]: neigh.score(X_train, y_train)
Out[78]: 0.7713190673906101

In [79]: neigh.score(X_valid, y_valid)
Out[79]: 0.6341281669150521
```

Рис. 28 — Точность на обучающих и тестовых данных

Видно, что наша модель плохо справляется с задачей классификации, верно, прогнозируются только 63 из 100 примеров из тестовой выборки. Необходимо подобрать наиболее оптимальные параметры нашей модели для повышения качества модели и увеличить качество прогнозирующей способности.

Листинг

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=64, weights='uniform', algorithm='brute', p=1)
neigh.fit(X_train, y_train)
neigh.score(X_train, y_train)
neigh.score(X_valid, y_valid)
```

Подбор параметров метода k-ближайших соседей для повышения предсказательной точности

Для метода взвешенных k-ближайших соседей важно выбрать оптимальное значение параметра k - количество соседей, используемых для классификации. Если значение k слишком мало, возникает эффект переобучения, при котором решение о классификации принимается на основе небольшого числа примеров и имеет низкую значимость. Это похоже на переобучение в деревьях решений, где существует множество правил для небольшого числа примеров. Кроме того, использование малых значений k увеличивает влияние шума на результаты, когда небольшие изменения в данных приводят к большим изменениям в конечном результате.

Опять прибегнем написанному ранее мной скриптом, отредактировав его для задачи логистической регрессии. И выберем наилучшие параметры исходя из наилучшей точности тестовой и кросс-валидационной выборки. Просмотрим всевозможные точности нашей модели при количестве соседей от 1 до 400:

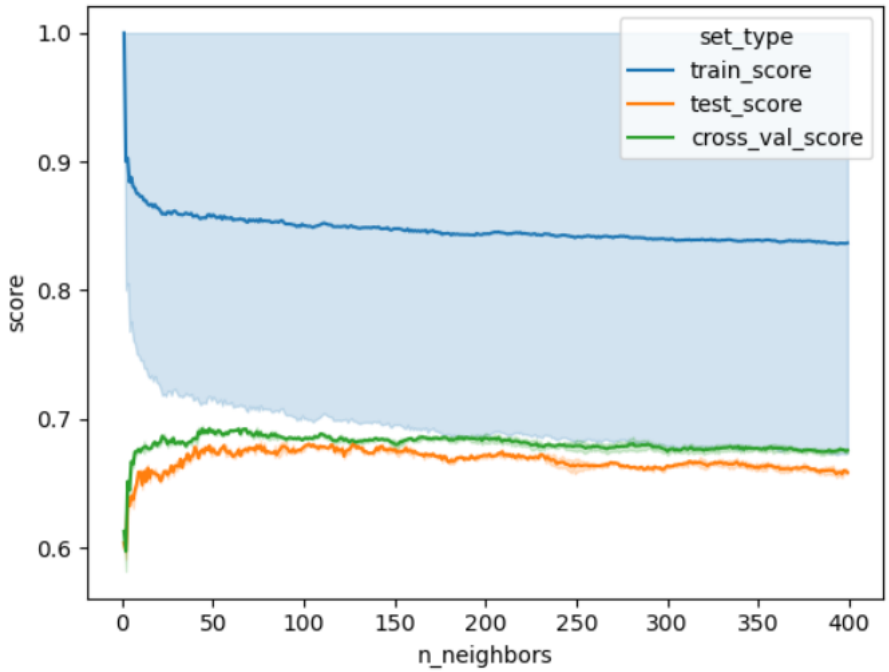

Рис. 29 — Точность на обучающих и тестовых данных

С помощью библиотеки pandas выбираем наибольшее значений столбца датафрейма узнаем, что это значение равно 40.

Воспользуемся функцией для подбора количества соседей, алгоритм отвечающий за голосование, а также как будем взвешивать дистанцию между объектами.

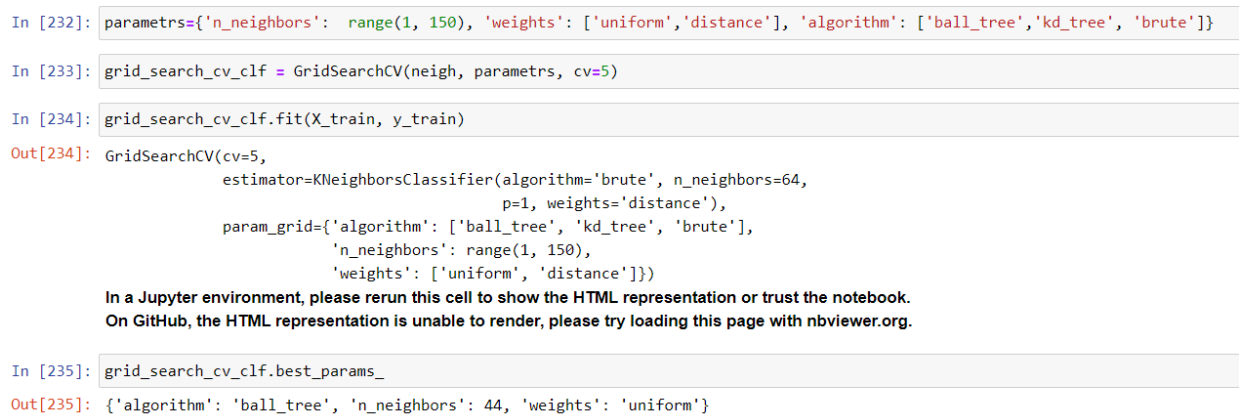

Рис. 30 — Функция, подбирающая наилучшую точность в зависимости от параметров

Можно видеть, что наибольшая точность всего в нашем случае будет получена при 44 соседях, при однородных дистанциях, алгоритм, отвечающий за голосование — brute.

Листинг

```
n_neighbors_values = range(1,100)
scores_data = pd.DataFrame()
for n_neighbors in n_neighbors_values:
    neigh = KNeighborsClassifier(weights='uniform', algorithm='ball_tree', p=1, n_neighbors=n_neighbors)
    neigh.fit(X_train, y_train)
    train_score = neigh.score(X_train, y_train)
    test_score = neigh.score(X_valid, y_valid)
    mean_cross_val_score = cross_val_score(neigh, X_train, y_train, cv=5).mean()
    temp_score_data = pd.DataFrame({'n_neighbors': [n_neighbors],
                                    'train_score': [train_score],
                                    'test_score': [test_score],
                                    'cross_val_score': [mean_cross_val_score]})
```

scores_data = pd.concat([scores_data, temp_score_data])

scores_data_long = pd.melt(scores_data, id_vars = ['n_neighbors'], value_vars = ['train_score','test_score', 'cross_val_score'], var_name = 'set_type', value_name = 'score')

sns.lineplot(x='n_neighbors', y='score', hue='set_type', data=scores_data_long)

scores_data_long.loc[scores_data_long['set_type'] == 'cross_val_score'].loc[scores_data_long['score'] == scores_data_long.loc[scores_data_long['set_type'] == 'cross_val_score'].score.max()]

Программная реализация линейного дискриминантного анализа

Алгоритм линейного дискриминантного анализа

Рассмотрим линейную функцию, называемую канонической дискриминантной функцией:

$$d_{km} = a_0 + a_1 x_{1km} + a_2 x_{2km} + \cdots + a_p x_{pkm}$$

Где d_{km} — значение дискриминирующей функции для m-наблюдения k-ой группы, $m = 1 \dots n, k = 1 \dots g, p -$ число дискриминантных признаков (размерность многомерного пространства).

Задача заключается в подборе коэффициентов, таким образом, чтобы построенная нами гиперплоскость, разбивающая пространство признаков таким образом, чтобы расстояние между центроидами результирующих подмножеств было максимально возможным. Эту задачу можно осуществить с помощью метода наименьших квадратов или других похожих методов минимизации отклонения значений, построенной функций от истинных значений [11].

Программная реализация линейного дискриминантного анализа

Построим модель логистической регрессии используя библиотеки sklearn в Jupyter notebook. Для этого подключим класс логистической регрессии используя инструментарий sklearn и создадим объект класса и сразу же обучим его, используя метод из этого класса. Также сразу подберём параметры для нашей модели ввиду того, что числовых параметров тут нет, а также их не так много мною вручную были подобраны параметры, следующие:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
LDA = LinearDiscriminantAnalysis(solver='svd', store_covariance = True)
```

Рис. 31 — Подключение линейного дискриминантного анализа

Оценим точность обученной мною модели используя метрику точности на обучающих и тестовых данных:

```
In [142]: LDA.score(X_train, y_train)
Out[142]: 0.7911210475886298

In [143]: LDA.score(X_valid, y_valid)
Out[143]: 0.7771982116244411
```

Рис. 32 — Точность на обучающих и тестовых данных

Можно наблюдать что 77 из 100 объектов классифицируется верно. У нас высокая точность предсказательной модели

Листинг

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
LDA = LinearDiscriminantAnalysis(solver='svd', store_covariance = True)
LDA.fit(X_train, y_train)
LDA.score(X_train, y_train)
LDA.score(X_valid, y_valid)
```

Программная реализация случайного леса

Алгоритм случайного леса

Идея случайного леса заключается в том, что каждое дерево на общем наборе данных переобучается строя модель конкретно под этот набор данных, но, если построить много деревьев, каждое под определенный набор данных и признаков и усреднить их оценку, мы получим наиболее корректный результат и уменьшим влияние переобучения. Деревья должны отличаться друг от друга в некоторой степени, не должны коррелировать друг с другом. Для решения этой задачи мы для каждого дерева будем отбирать случайные признаки, а также случайные наблюдения. Отсюда и название этого метода [17].

Математически алгоритм случайного леса можно описать следующим образом:

1. Для b=1,2…, B, где B-количество деревьев

a) Извлечь выборку S размера N из обучающих данных

b) По выборке S построить полное дерево T_b, рекурсивно повторяя следующие шаги для каждого терминального узла, пока не будет достигнуто минимальное количество наблюдений в нём (для классификации – одно наблюдение)

i. Из первоначального набора M предикторов случайно выбрать m предикторов;

ii. Из m предикторов выбрать предиктор, который обеспечивает наилучшее расщепление;

iii. Расщепить узел на два узла потомка

2. В результате получаем ансамбль деревьев решений $\{T_b\}_{b=1}^{B}$

3. Пусть $\widehat{C_b}(x)$ - класс, спрогнозированный деревом решений T_b, то есть $T_b(x) = \widehat{C_b}(x)$; тогда $\widehat{C}_{rf}^{B}(x)$ - это класс, наиболее часто встречающийся в множестве $\{\widehat{C_b}(x)\}_{b=1}^{B}$

Программная реализация леса случайных решений

Вырастим случайный лес решений используя библиотеки sklearn в Jupyter notebook. Также подключим класс случайного леса решений используя инструментарий sklearn и создадим объект класса и обучим его, используя метод из этого класса:

```
In [ ]: from sklearn.ensemble import RandomForestClassifier

In [150]: forest = RandomForestClassifier()

In [151]: forest.fit(X_train, y_train)

Out[151]: ▾ RandomForestClassifier
          RandomForestClassifier()
```

Рис. 33 — Подключение логистической

Оценим точность обученной модели используя метрику точности на обучающих и тестовых данных:

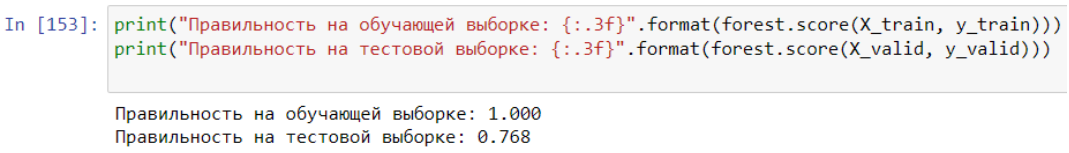

Рис. 34 — Точность на обучающих и тестовых данных

Видно, что наша модель хорошо справляется с задачей классификации, верно, прогнозируются только 76 из 100 примеров из тестовой выборки. Попробуем подобрать наиболее оптимальные параметры нашей модели для повышения качества модели и увеличить качество прогнозирующей способности.

Листинг

```
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(max_depth = 8,random_state=152, min_samples_split = 8, n_estimators=300)
forest.fit(X_train, y_train)
forest.score(X_train, y_train)
print("Правильность на обучающей выборке: {:.3f}".format(forest.score(X_train, y_train)))
print("Правильность на тестовой выборке: {:.3f}".format(forest.score(X_valid, y_valid)))
```

Подбор параметров леса случайных решений для повышения предсказательной точности

Прибегнем написанному ранее мной скриптом, отредактировав его для задачи случайного леса решений. И выберем наилучшие параметры исходя из наилучшей точности тестовой и кросс-валидационной выборки. Просмотрим всевозможные точности нашей модели при количестве деревьев в ансамбле от 1 до 100:

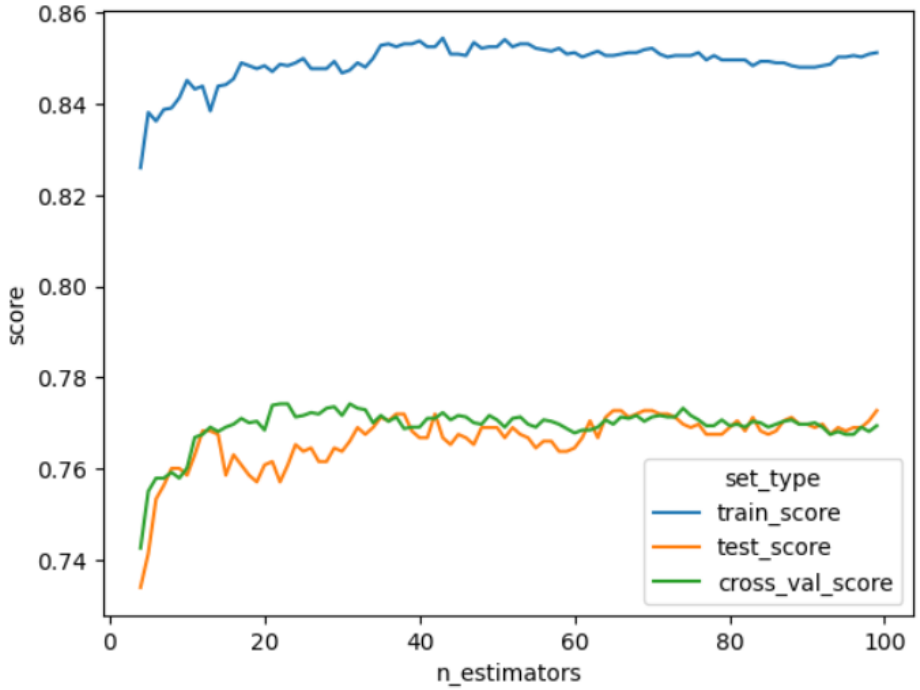

Рис. 35 — Зависимость точности от количества деревьев в ансамбле

С помощью библиотеки pandas выбираем наибольшее значений столбца датафрейма узнаем, что это значение равно 31.

Воспользуемся функцией для подбора количества соседей, алгоритм, отвечающий за голосование, а также как будем взвешивать дистанцию между объектами.

Остальные параметры подберем таким возьмем из метода дерева решений, ввиду того что они уже там подобраны наилучшим образом.

Листинг

```
scores_data = pd.DataFrame()
n_estimators_values = range(4, 300)

for n_estimators in n_estimators_values:
    forest = RandomForestClassifier(n_estimators=n_estimators, max_depth =
8,random_state=152, min_samples_split = 2)
    forest.fit(X_train, y_train)
    train_score = forest.score(X_train, y_train)
    test_score = forest.score(X_valid, y_valid)
    mean_cross_val_score = cross_val_score(forest, X_train, y_train, cv=5).mean()
    temp_score_data = pd.DataFrame({'n_estimators': [n_estimators],
                                    'train_score': [train_score],
                                    'test_score': [test_score],
                                    'cross_val_score': [mean_cross_val_score]})
    scores_data = pd.concat([scores_data, temp_score_data])
```

```
scores_data_long = pd.melt(scores_data, id_vars = ['n_estimators'], value_vars = ['train_score','test_score', 'cross_val_score'], var_name = 'set_type', value_name = 'score')
sns.lineplot(x='n_estimators', y='score', hue='set_type', data=scores_data_long)
scores_data_long.loc[scores_data_long['set_type'] == 'cross_val_score'].loc[scores_data_long['score'] == scores_data_long.loc[scores_data_long['set_type'] == 'cross_val_score'].score.max()]
```

ТЕСТИРОВАНИЕ, АНАЛИЗ РЕЗУЛЬТАТОВ

Проверка предсказательной способности моделей машинного обучения

Предсказательная способность дерева решений

Теперь можно оценить эти метрики.

```
Ввод [177]: clf.score(X_train, y_train)
Out[177]: 0.7745129351644842

Ввод [178]: clf.score(X_valid, y_valid)
Out[178]: 0.732488822652757
```

Рисунок – 38 Accuracy тренировочной и тестовой выборки

У нас достаточно высокая предсказательная точность нашего дерево классификации. Теперь оценим такие метрики как recall и precision и F–мера.

```
Ввод [206]: from sklearn.metrics import precision_score, recall_score

Ввод [207]: precision_score(y_valid, y_pred)
Out[207]: 0.8155802861685215

Ввод [208]: recall_score(y_valid, y_pred)
Out[208]: 0.6794701986754967

Ввод [155]: f1_score(y_valid, y_pred)
Out[155]: 0.7407942238267148
```

Рисунок – 39 recall, precision и F–мера

Модель показывает высокий показатель precision метрики и средний показатель recall

Посмотрим, как выглядит матрица ошибок в нашем случае:

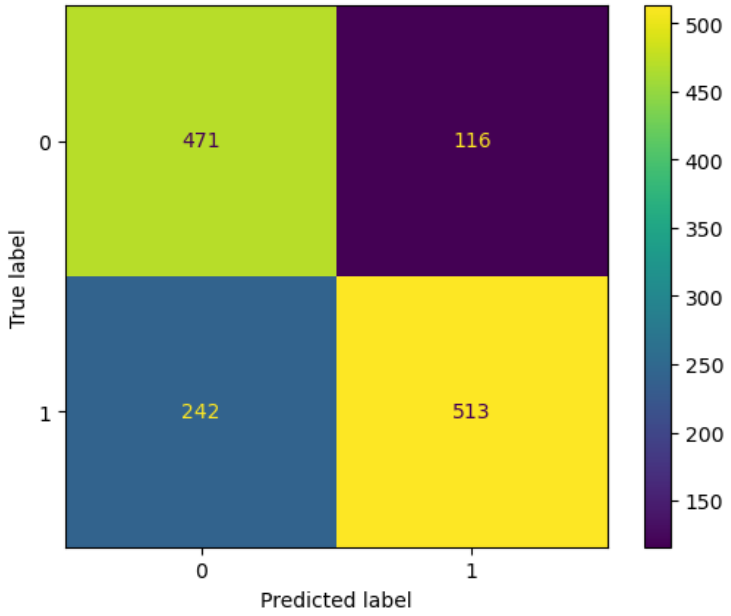

Рисунок – 40 Матрица ошибок

И наконец, посмотрим, как выглядит ROC-кривая для нашего случая:

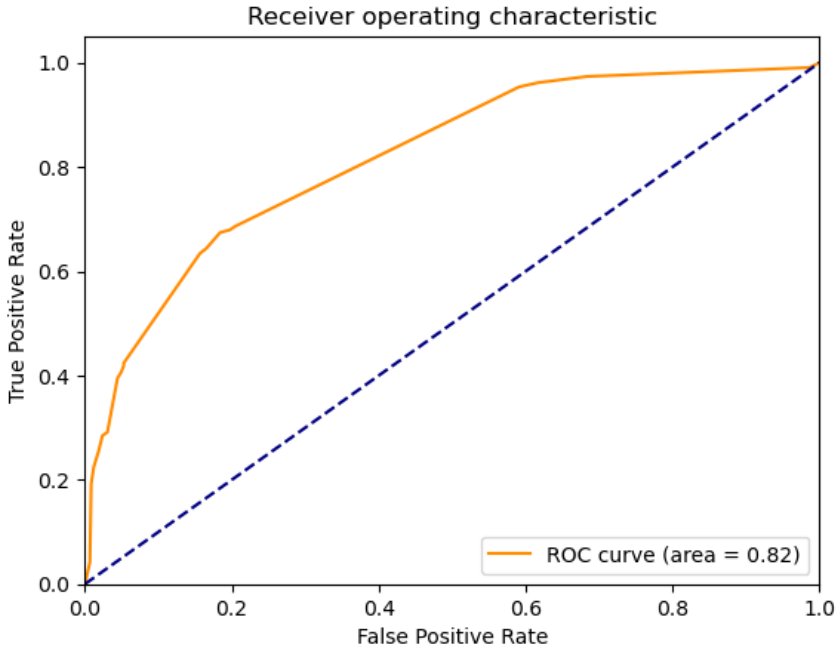

Рисунок – 41 ROC-кривая нашей модели

Площадь нашей ROC-кривой имеет достаточно большое значение, что говорит о хорошей предсказательной точности нашей модели.

Листинг

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_valid, y_predicted_prob[:,1])
roc_auc= auc(fpr, tpr)
```

```
plt.figure()
plt.plot(fpr, tpr, color='darkorange',  label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy',  linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
clf.score(X_train, y_train)
clf.score(X_valid, y_valid)
y_pred = clf.predict(X_valid)
from sklearn.metrics import precision_score, recall_score
precision_score(y_valid, y_pred)
recall_score(y_valid, y_pred)
f1_score(y_valid, y_pred)
from sklearn.metrics import confusion_matrix
metrics.plot_confusion_matrix(clf, x_Valid, y_valid)
```

Предсказательная способность логистической регрессии

Оценим качество нашей модели. Для этого воспользуемся разными метриками качества результата машинного обучения.

```
In [53]: LogR.score(X_train, y_train)
Out[53]: 0.7962312360268284

In [54]: LogR.score(X_valid, y_valid)
Out[54]: 0.7809239940387481
```

Рисунок – 42 Accuracy тренировочной и тестовой выборки.

У нас высокая предсказательная точность нашей логистической регрессии. Кроме того, можно наблюдать что подбор параметров повысил качество нашей модели. Теперь оценим такие метрики как recall и precision и F–мера.

```
In [207]: recall_score(y_valid, y_pred)
Out[207]: 0.8635761589403973

In [208]: precision_score(y_valid, y_pred)
Out[208]: 0.7734282325029656

In [209]: f1_score(y_valid, y_pred)
Out[209]: 0.8160200250312891
```

Рисунок – 43 recall, precision и F–мера

Модель показывает высокий показатель precision метрики и очень высокий показатель recall

Посмотрим, как выглядит матрица ошибок в нашем случае:

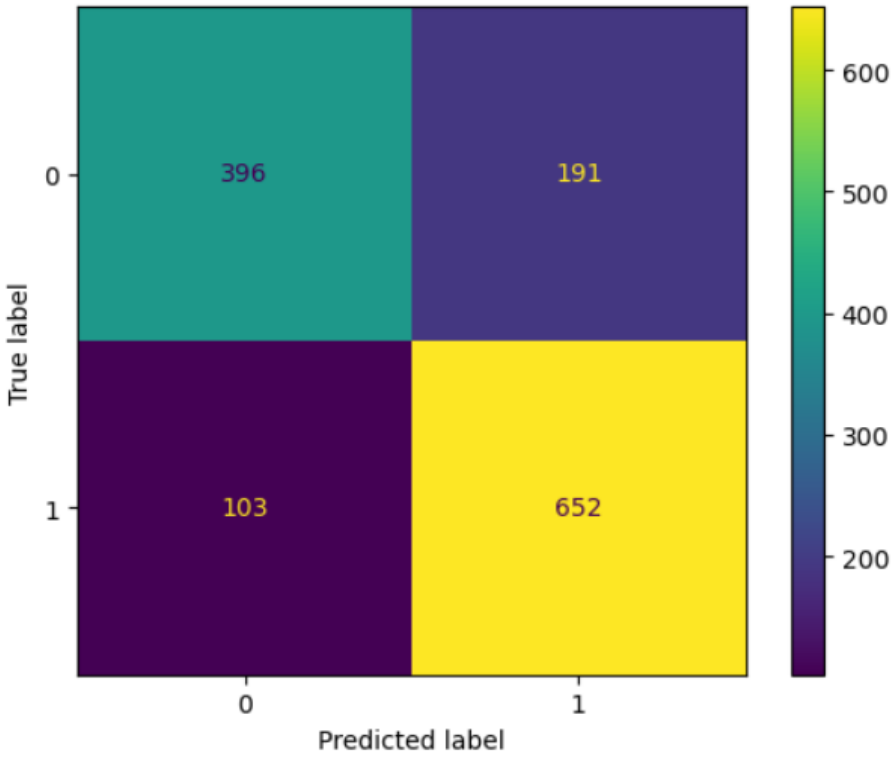

Рисунок – 44 Матрица ошибок

И наконец, посмотрим, как выглядит ROC-кривая для нашего случая:

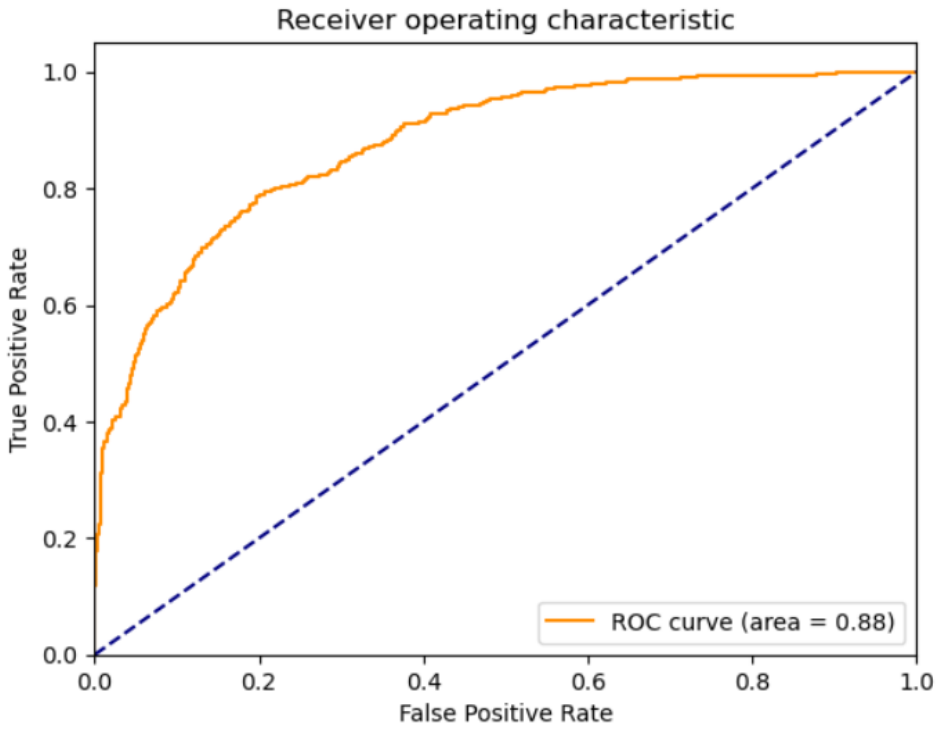

Рисунок – 45 ROC-кривая нашей модели

Площадь нашей ROC-кривой имеет очень большое значение, что говорит о отличной предсказательной возможности логистической регрессии.

Листинг

```
LogR.score(X_train, y_train)
LogR.score(X_valid, y_valid)
y_pred = LogR.predict(X_valid)
recall_score(y_valid, y_pred)
precision_score(y_valid, y_pred)
f1_score(y_valid, y_pred)
y_predicted_prob = LogR.predict_proba(X_valid)
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_valid, y_predicted_prob[:,1])
roc_auc= auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange',  label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy',  linestyle='--')
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
predictions = LogR.predict(X_valid)
cm = confusion_matrix(y_valid, predictions, labels=LogR.classes_)
y_predicted_prob = LogR.predict_proba(X_valid)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
...                               display_labels=LogR.classes_)
disp.plot()
```

Предсказательная способность метода k-ближайших соседей

Оценим качество нашей модели. Для этого воспользуемся разными метриками качества результата машинного обучения.

```
In [112]: neigh.score(X_train, y_train)
Out[112]: 0.7099968061322262

In [113]: neigh.score(X_valid, y_valid)
Out[113]: 0.6788375558867362
```

Рисунок – 46 Accuracy тренировочной и тестовой выборки

У нас средняя точность нашей предсказательной модели. С помощью подбора параметров удалось повысить предсказательную точность. Теперь оценим такие метрики как recall и precision и F–мера.

```
In [218]: precision_score(y_valid, y_pred)
Out[218]: 0.6828442437923251

In [219]: recall_score(y_valid, y_pred)
Out[219]: 0.8013245033112583

In [220]: f1_score(y_valid, y_pred)
Out[220]: 0.7373552711761122
```

Рисунок – 47 recall, precision и F-мера

Модель показывает средний показатель precision метрики и высокий показатель recall

Посмотрим, как выглядит матрица ошибок в нашем случае:

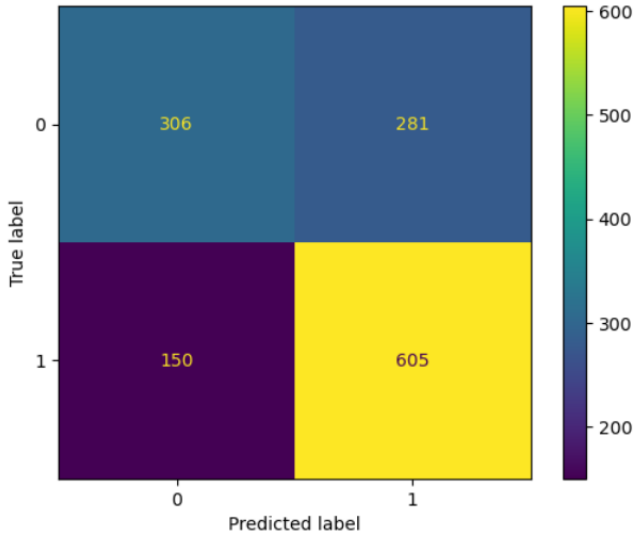

Рисунок – 48 Матрица ошибок

И наконец, посмотрим, как выглядит ROC-кривая для нашего случая:

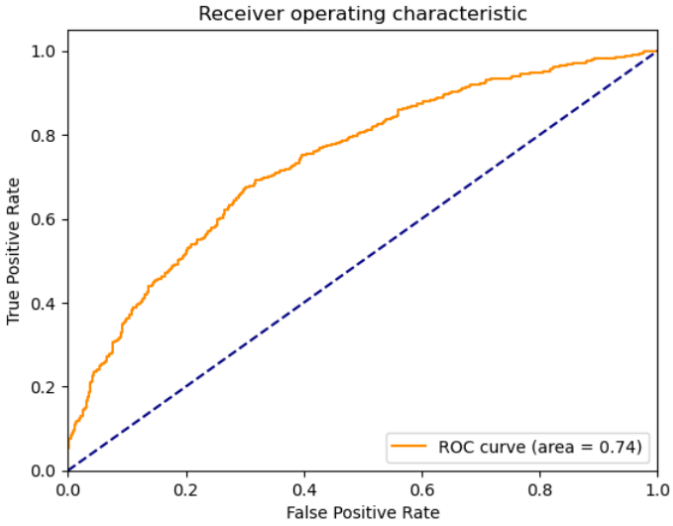

Рисунок – 49 ROC-кривая нашей модели

Площадь нашей ROC-кривой имеет выше среднего, что говорит об неплохой предсказательной возможности метода ближайших k-соседей.

Листинг

```
neigh.score(X_train, y_train)
neigh.score(X_valid, y_valid)
y_pred = neigh.predict(X_valid)
precision_score(y_valid, y_pred)
recall_score(y_valid, y_pred)
f1_score(y_valid, y_pred)
y_predicted_prob = neigh.predict_proba(X_valid)
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_valid, y_predicted_prob[:,1])
roc_auc= auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange',  label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy',  linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
predictions = neigh.predict(X_valid)
cm = confusion_matrix(y_valid, predictions, labels=neigh.classes_)
y_predicted_prob = neigh.predict_proba(X_valid)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
...                               display_labels=neigh.classes_)
disp.plot()
```

Предсказательная способность линейного дискриминантного анализа

Оценим качество нашей модели. Для этого воспользуемся разными метриками качества результата машинного обучения.

Оценим такие метрики как recall и precision и F–мера.

```
In [28]: precision_score(y_valid, y_pred)
Out[28]: 0.7608695652173914

In [29]: recall_score(y_valid, y_pred)
Out[29]: 0.8807947019867549

In [30]: f1_score(y_valid, y_pred)
Out[30]: 0.816451810926949
```

Рисунок – 50 recall, precision и F–мера

Модель показывает средний высокий precision метрики и очень высокий показатель recall. Также у нас отличная f-мера.

Посмотрим, как выглядит матрица ошибок в нашем случае:

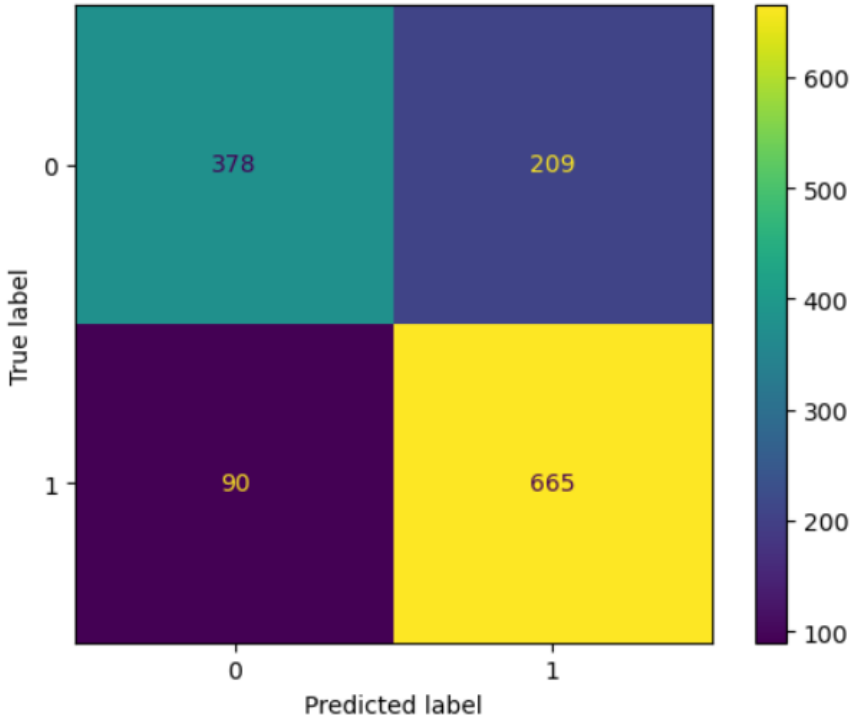

Рисунок – 51 Матрица ошибок

И наконец, посмотрим, как выглядит ROC-кривая для нашего случая:

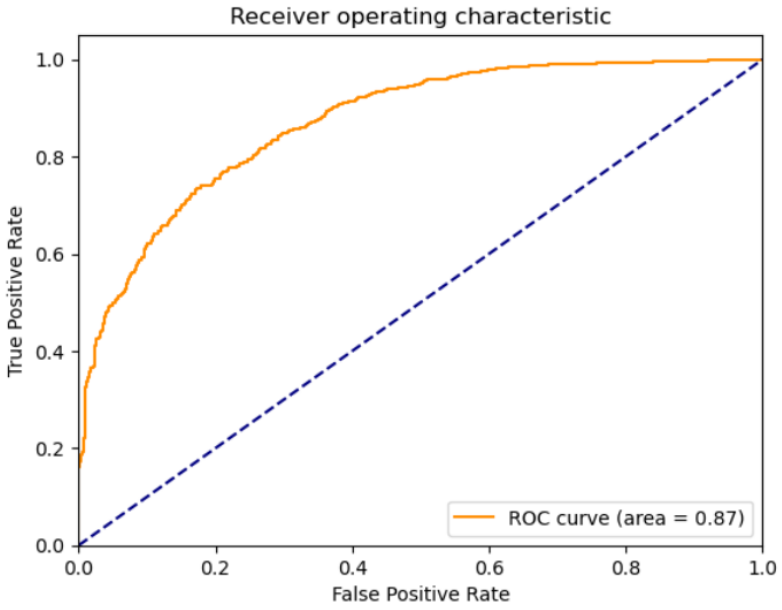

Рисунок – 52 ROC-кривая нашей модели

Площадь нашей ROC-кривой имеет очень большое значение, что говорит о отличной предсказательной возможности линейного дискриминантного анализа.

Листинг

```
precision_score(y_valid, y_pred)
recall_score(y_valid, y_pred)
f1_score(y_valid, y_pred)
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_valid, y_predicted_prob[:,1])
roc_auc= auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange',  label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy',  linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
```

```
plt.show()
predictions = LDA.predict(X_valid)
cm = confusion_matrix(y_valid, predictions, labels=LDA.classes_)
y_predicted_prob = neigh.predict_proba(X_valid)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
...                               display_labels=LDA.classes_)
disp.plot()
```

Проверка предсказательной способности леса случайных решений

Оценим качество нашей модели. Для этого воспользуемся разными метриками качества результата машинного обучения.

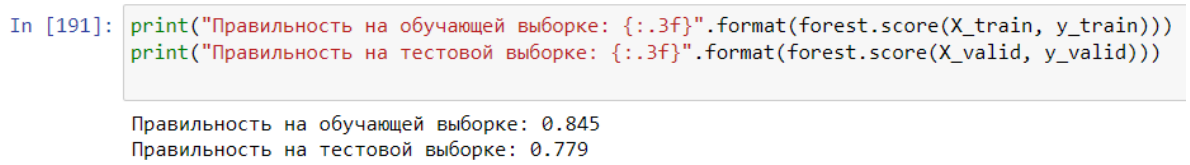

Рисунок – 53 Accuracy тренировочной и тестовой выборки.

У нас высокая предсказательная точность нашего дерево классификации. Кроме того, можно наблюдать что подбор параметров повысил качество нашей модели. Теперь оценим такие метрики как recall и precision и F–мера.

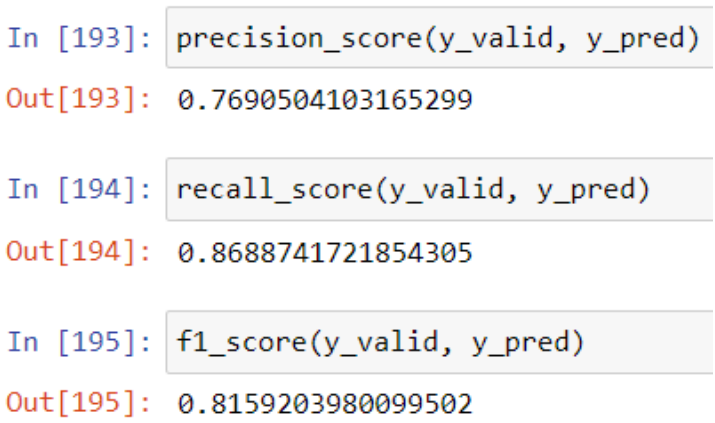

Рисунок – 54 recall, precision и F–мера

Модель показывает высокие показатели всех метрик, в частности, метрики recall.

Посмотрим, как выглядит матрица ошибок в нашем случае:

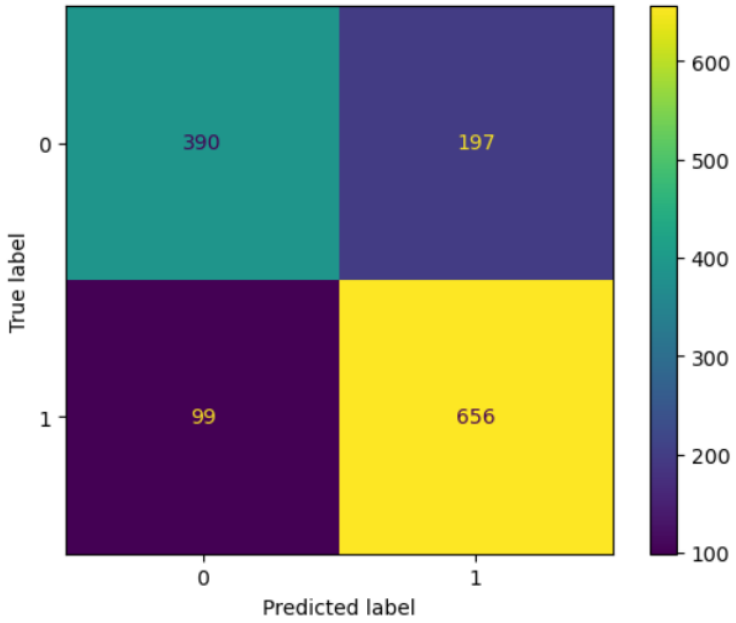

Рисунок – 55 Матрица ошибок

И наконец, посмотрим, как выглядит ROC-кривая для нашего случая:

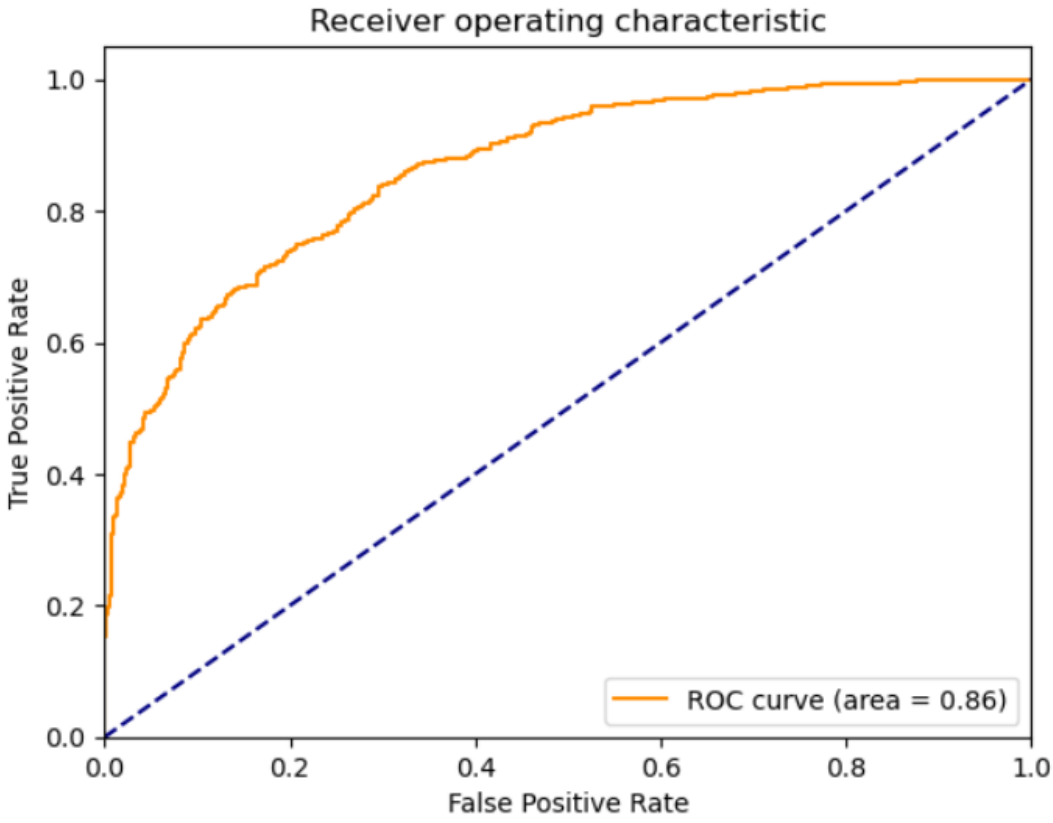

Рисунок – 56 ROC-кривая нашей модели

Площадь нашей ROC-кривой имеет очень большое значение, что говорит о отличной предсказательной возможности случайного леса решений.

Листинг

```
print("Правильность на обучающей выборке: {:.3f}".format(forest.score(X_train, y_train)))
print("Правильность на тестовой выборке: {:.3f}".format(forest.score(X_valid, y_valid)))
precision_score(y_valid, y_pred)
recall_score(y_valid, y_pred)
f1_score(y_valid, y_pred)
predictions = forest.predict(X_valid)
cm = confusion_matrix(y_valid, predictions, labels=forest.classes_)
y_predicted_prob = forest.predict_proba(X_valid)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
...                               display_labels=forest.classes_)
from sklearn.metrics import roc_curve, auc
disp.plot()
fpr, tpr, thresholds = roc_curve(y_valid, y_predicted_prob[:,1])
roc_auc= auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange',  label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy',  linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

Сравнительный анализ методов машинного обучения

Проведём сравнительный анализ методов машинного обучения. Для этого построим таблицу с метриками качества модели и выберем самую лучшую модель.

Таблица 1. Сравнительный анализ методов машинного обучения

	Точность	Полнота	Чувствительность	F-мера	ROC-AUC
Дерево решений	0.732	0.815	0.679	0.741	0.82
Логистическая регрессия	0.781	0.863	0.773	0.816	0.88
Метод k-ближайших соседей	0.678	0.682	0.802	0.737	0.74
Линейный дискриминатный анализ	0.777	0.761	0.881	0.816	0.87
Случайный лес решений	0.779	0.769	0.868	0.816	0.86

Лучше всего в нашей задаче себя проявила логистическая регрессия. Чуть похоже ЛДА и лес случайных решений примерно на одной уровне, чуть послабее выступило дерево решений и хуже всего себя проявил метод k-ближайших соседей. Лучше всего себя проявляет логистическая регрессия из-за того, что у нас много непрерывных признаков. Дерево решений себя проявляет не так хорошо ввиду того, что у нас не так много категориальных предикторов, с которыми хорошо справляется дерево решений. Впрочем, лес решений решает эту проблему и повышает точность практически до точности логистической регрессии. Метод k-ближайших соседей плохо себя проявляет ввиду того, что она не предназначена для задач с таким большим количеством предикторов. Кроме того, наш задача имеет сложные взаимосвязи, которые невозможно вычленить с помощью этого метода.

ЗАКЛЮЧЕНИЕ

В данной работе мною была построена модель, на основе которой я могу прогнозировать результат UFC матча основываясь на некоторых входных данных.

В ходе написания построения модели мною была написана функция, позволяющая оптимизировать методы машинного обучения и подобрать оптимальные параметры для него.

Выходными данными является результат матча в виде единицы или нуля, где единица победа “красного” игрока и ноль победа “синего” игрока.

Была проверена возможность предсказательной возможности наших моделей, дана оценка наших модели с помощью разных метрик. Мы провели сравнительный анализ этих моделей и выбрали наилучший справляющийся с нашей задачей.