Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА ИУ5 «Системы обработки информации и управления»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Решение задачи машинного обучения

Студент группы ИУ5-61Б _____ Евдокимов А.А.
(Группа) (Подпись, дата) (И.О.Фамилия)

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

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

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

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

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине «Технологии машинного обучения»_____

Студент группы ИУ5-61Б_____

_____Евдокимов Арсений Аликович_____
(Фамилия, имя, отчество)

Тема курсового проекта _____
_____
_____

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

График выполнения проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Задание: решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично.
_____
_____
_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на __26___ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « 12 » февраля 2020 г.

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

Студент _____ _____Евдокимов А.А.
(Подпись, дата) (И.О.Фамилия)

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

Оглавление

Задание

Схема типового исследования, проводимого студентом в рамках курсовой работы, содержит выполнение следующих шагов:

- Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
- Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
- Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
- Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
- Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
- Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
- Формирование обучающей и тестовой выборок на основе исходного набора данных.
- Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производятся обучение моделей на основе обучающей выборки и оценка качества моделей

на основе тестовой выборки.

- Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать

перебор параметров в цикле, или использовать другие методы.

- Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

- Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового

описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

Введение

Курсовой проект – самостоятельная часть учебной дисциплины «Технологии машинного обучения» – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Курсовой проект опирается на знания, умения и владения, полученные студентом в рамках лекций и лабораторных работ по дисциплине.

В рамках данной курсовой работы необходимо применить навыки, полученные в течение курса «Технологии машинного обучения», и обосновать полученные результаты.

Основная часть

Постановка задачи

В данной курсовой работе ставится задача определения пригодности гриба в употребление по внешним параметрам с помощью методов машинного обучения «Stochastic gradient descent», «Support vector machine», «Метод ближайших соседей», «Gradient boosting» и «Random forest».

Описание набора данных

В данной работе для исследований был выбран следующий набор данных: https://www.kaggle.com/iamhungundji/covid19-symptoms-checker?select=Cleaned-Data.csv

Этот набор данных включает описание различных симптомов вируса COVID-19 и различных параметров, таких как пол, страна и так далее. Из-за объема датасета возникла необходимость в его сокращении до одной страны (Китай). Файл *Clean_data.csv* содержит 31680 строки и 27 столбцов (после сокращения). В него включены:

Атрибуты:

1-Лихорадка - Fever: есть=1, отсутствует=0:

2-Усталость - Tiredness: есть=1, отсутствует=0;

3-Сухой кашель – Dry-Cough: есть=1, отсутствует=0;

4-Раздражение в горле – Sore-throat: есть=1, отсутствует=0;

5-Нет симптомов – None_Synpton: есть=1, отсутствует=0;

6-Боли – Pains: есть=1, отсутствует=0;

7-Заложенность носа – Nasal-Congestion: есть=1, отсутствует=0;

8-Насморк – Runny-Nose: есть=1, отсутствует=0;

9-Диарея - Diarrhea: есть=1, отсутствует=0;

10-Нет заметных симптомов – None_Experiencing: есть=1, отсутствует=0;

11-Возраст 0-9 – Age_0-9: есть=1, отсутствует=0;

12-Возраст 10-19 – Age_10-10: есть=1, отсутствует=0;

13-Возраст 20-24 – Age_20-24: есть=1, отсутствует=0;

14-Возраст 25-59 – Age_25-50: есть=1, отсутствует=0;

15-Возраст 60+ - Age_60+: есть=1, отсутствует=0;

16-Женщина – Gender_Female: есть=1, отсутствует=0;

17-Мужчина – Gender_Male: есть=1, отсутствует=0;

18-Трансгендер – Gender_Transgender: есть=1, отсутствует=0;

19-Слабая тяжесть – Severity_Mild: есть=1, отсутствует=0;

20-Средняя тяжесть – Severity_Moderate: есть=1, отсутствует=0;

21-Нет тяжести – Severity_None: есть=1, отсутствует=0;

22-Большая тяжесть – Severity_Severe: есть=1, отсутствует=0;

23-Контакты неизвестны – Contact_Dont-Know: есть=1, отсутствует=0;

24-Не было контактов – Contact_No: есть=1, отсутствует=0;

25-Были контакты – Contact_Yes: есть=1, отсутствует=0;

26-Страна - Country: есть=1, отсутствует=0;

27-Затруднения в дыхании: есть=1, отсутствует=0;

Для данного набора данных мы будем решать задачу классификации – определение наличия затрудненности дыхания в зависимости от других параметров.

Ход работы

Импортируем необходимые для работы библиотеки:

```
from sklearn.utils.multiclass import unique_labels
from typing import Dict
import numpy as np
import pandas as pd
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split, learning_curve, validation_curve
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, classification_report, confusion_matrix, \
    balanced_accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

Считываем набор данных:

```
# Загрузка датасета
covid = pd.read_csv('./Cleaned-Data.csv', sep=",")
```

Размер датасета:

```
covid.shape
```

```
(31680, 27)
```

Первые пять строк датасета:

```
# Вывод первых пяти строк
covid.head()
```

	Fever	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	None_Sympton	Pains	Nasal-Congestion	Runny-Nose	Diarrhea	...	Gender_Male	Gender_Transgender	Severity_Mild	S
0	1	1	1	1	1	0	1	1	1	1	...	1	0	1	
1	1	1	1	1	1	0	1	1	1	1	...	1	0	1	
2	1	1	1	1	1	0	1	1	1	1	...	1	0	1	
3	1	1	1	1	1	0	1	1	1	1	...	1	0	0	
4	1	1	1	1	1	0	1	1	1	1	...	1	0	0	

Типы столбцов:

```
# Типы данных
covid.dtypes
```

```
Fever                        int64
Tiredness                    int64
Dry-Cough                    int64
Difficulty-in-Breathing      int64
Sore-Throat                  int64
None_Sympton                 int64
Pains                        int64
Nasal-Congestion             int64
Runny-Nose                   int64
Diarrhea                     int64
None_Experiencing            int64
Age_0-9                      int64
Age_10-19                    int64
Age_20-24                    int64
Age_25-59                    int64
Age_60+                      int64
Gender_Female                int64
Gender_Male                  int64
Gender_Transgender           int64
Severity_Mild                int64
Severity_Moderate            int64
Severity_None                int64
Severity_Severe              int64
Contact_Dont-Know            int64
Contact_No                   int64
Contact_Yes                  int64
Country                     object
dtype: object
```

Основные статистические характеристики датасета:

```
# Статистические данные
covid.describe()
```

	Fever	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	None_Sympton	Pains	Nasal-Congestion	Runny-Nose	Diarrhea	...	Gen
count	31680.00000	31680.000000	31680.000000	31680.000000	31680.00000	31680.000000	31680.000000	31680.000000	31680.000000	31680.000000	...	31
mean	0.31250	0.500000	0.562500	0.500000	0.31250	0.062500	0.363636	0.545455	0.545455	0.363636	...	
std	0.46352	0.500008	0.496086	0.500008	0.46352	0.242065	0.481053	0.497937	0.497937	0.481053	...	
min	0.00000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	0.00000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
50%	0.00000	0.500000	1.000000	0.500000	0.00000	0.000000	0.000000	1.000000	1.000000	0.000000	...	
75%	1.00000	1.000000	1.000000	1.000000	1.00000	0.000000	1.000000	1.000000	1.000000	1.000000	...	
max	1.00000	1.000000	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000	1.000000	1.000000	...	

8 rows × 26 columns

Названия колонок:

```
mushrooms.columns
```

```
Index(['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
       'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
       'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
       'stalk-surface-below-ring', 'stalk-color-above-ring',
       'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
       'ring-type', 'spore-print-color', 'population', 'habitat'],
      dtype='object')
```

Проверка набора данных на пропуски:

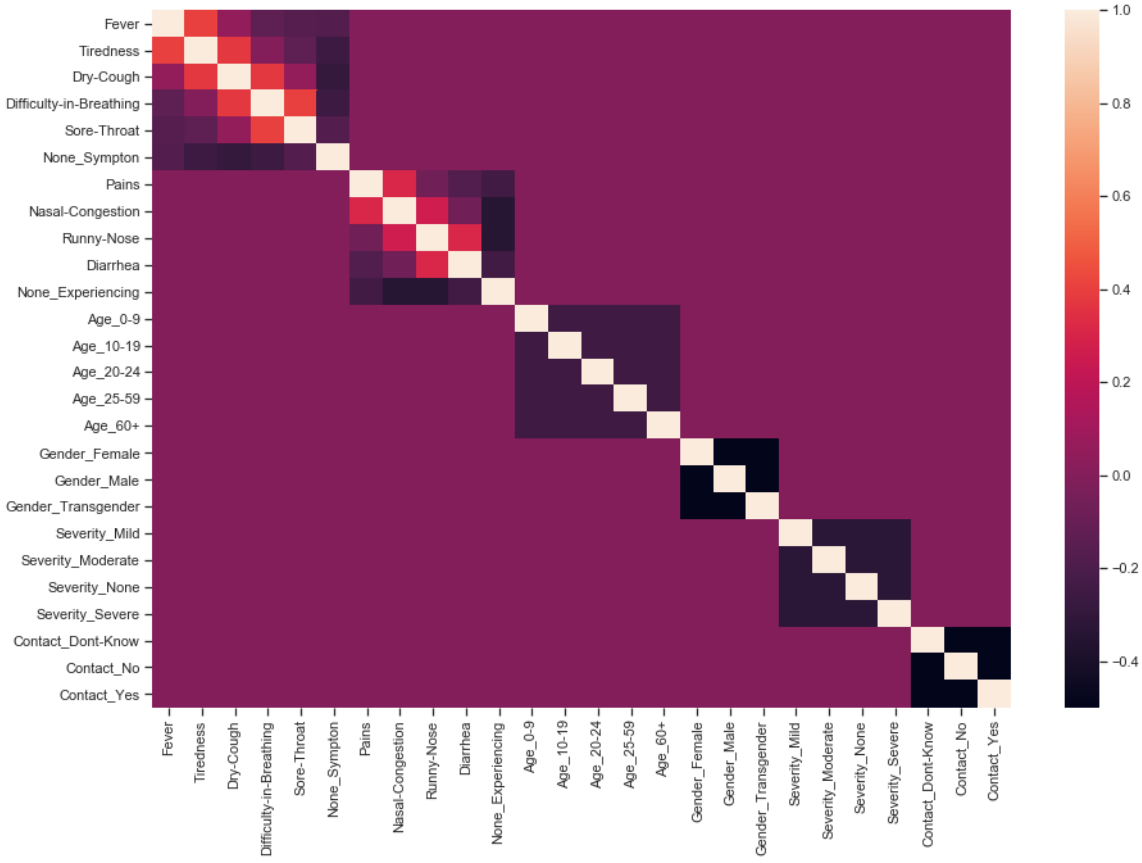
```
# Количество пустых значений
covid.isnull().sum()
```

```
Fever                        0
Tiredness                    0
Dry-Cough                    0
Difficulty-in-Breathing      0
Sore-Throat                  0
None_Sympton                 0
Pains                        0
Nasal-Congestion             0
Runny-Nose                   0
Diarrhea                     0
None_Experiencing            0
Age_0-9                      0
Age_10-19                    0
Age_20-24                    0
Age_25-59                    0
Age_60+                      0
Gender_Female                0
Gender_Male                  0
Gender_Transgender           0
Severity_Mild                0
Severity_Moderate            0
Severity_None                0
Severity_Severe              0
Contact_Dont-Know            0
Contact_No                   0
Contact_Yes                  0
Country                      0
dtype: int64
```

Пропущенных значений в наборе данных нет.
Теперь построим корреляционную матрицу:

```
In [37]: # Матрица корреляции
         fig, ax = plt.subplots(figsize=(15,10))
         sns.heatmap(covid.corr())
```

```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x2247130c148>
```



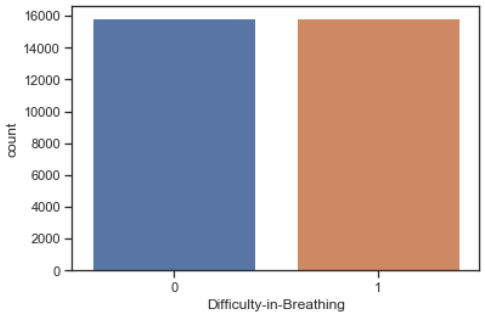
На основе корреляционной матрицы сложно судить о том, насколько

качественные модели машинного обучения можно построить, т.к. наиболее коррелирующие признаки с целевым имеют скромные значения, и максимальный из них – gill-size = 0,54.

Распределение данных целевого признака:

```
In [13]: # Дисбаланс классов по затруднению дыхательности деятельности
         sns.countplot(covid['Difficulty-in-Breathing'])
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x22461506788>
```




```
In [14]: covid['Difficulty-in-Breathing'].value_counts()
```

```
Out[14]: 1    15840
         0    15840
         Name: Difficulty-in-Breathing, dtype: int64
```

Подготовим данные для разделения на обучающую и тестовую выборки:

```
le = LabelEncoder()
covid['Country'] = le.fit_transform(covid['Country'])

# Подготовка данных
X = covid.drop('Difficulty-in-Breathing', axis = 1)
y = covid['Difficulty-in-Breathing']

# Разделение набора данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)

# Применение стандартного масштабирования для оптимизации результата
sc = MinMaxScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Выберем подходящие для нашей задачи метрики:

1. Confusion matrix

Количество верно и ошибочно классифицированных данных, представленное в виде матрицы.

2. ROC-кривая

Используется для оценки качества бинарной классификации. Показывает, какую долю классов алгоритм предсказал неверно. Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

3. Accuracy

Метрика вычисляет процент (долю в диапазоне от 0 до 1) правильно определенных классов. Главная проблема метрики accuracy в том, что она показывает точность по всем классам, но для каждого класса точность может

быть разная. Поэтому более предпочтительной является метрика balanced_accuracy.

Random Forest

Ансамблевый метод, заключается в построении алгоритма машинного обучения на базе нескольких, в данном случае решающих деревьев. это множество решающих деревьев. В задаче регрессии их ответы усредняются, в задаче классификации принимается решение голосованием по большинству. Все деревья строятся независимо по следующей схеме:

- Выбирается подвыборка обучающей выборки размера samplesize (м.б. с возвращением) – по ней строится дерево (для каждого дерева — своя подвыборка).
- Для построения каждого расщепления в дереве просматриваем max_features случайных признаков (для каждого нового расщепления — свои случайные признаки).
- Выбираем наилучшие признак и расщепление по нему (по заранее заданному критерию). Дерево строится, как правило, до исчерпания выборки (пока в листьях не останутся представители только одного класса), но есть параметры, которые ограничивают высоту дерева, число объектов в листьях и число объектов в подвыборке, при котором проводится расщепление.

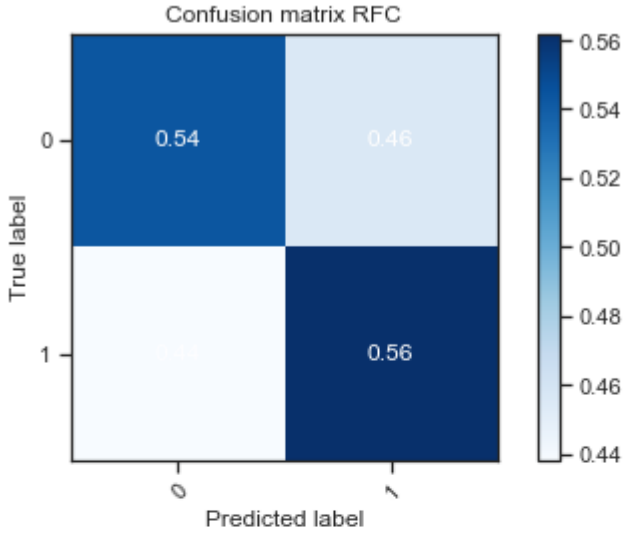
Обучим модель:

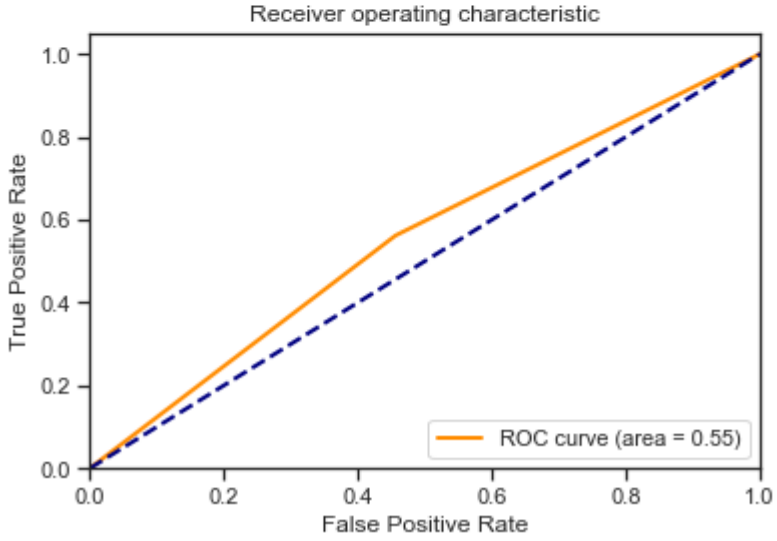
```
rfc = RandomForestClassifier(n_estimators=10)
rfc.fit(X_train, y_train)
pred_rfc = rfc.predict(X_test)
```

Оценим результаты работы нашей модели:

```
# Оценка результата работы модели
# RandomForest
plot_confusion_matrix(y_test, pred_rfc,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix RFC')
draw_roc_curve(y_test.values, pred_rfc)
balanced_accuracy_score(y_test, pred_rfc)
```

```
Normalized confusion matrix
```






```
: 0.5527725476253869
```

Попробуем улучшить качество модели с помощью подбора гиперпараметров при помощи метода GridSearchCV:

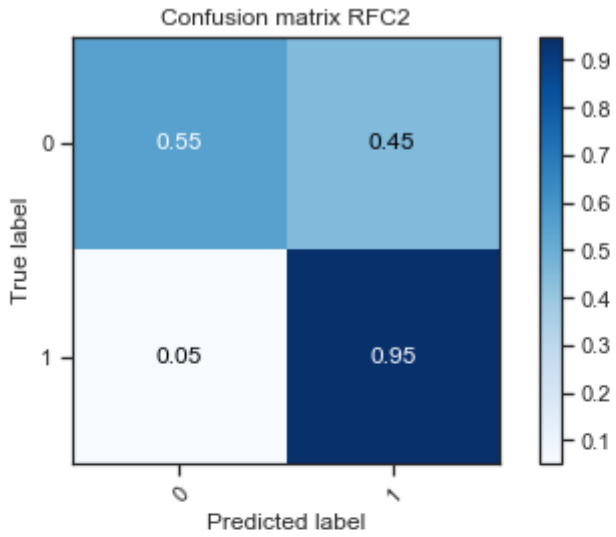
```
param_rfc = {'n_estimators':[1, 3, 5, 7, 10, 13, 16, 19],
    'max_depth':[1, 3, 5, 7, 10, 13, 16, 19],
    'random_state':[0, 2, 4, 6, 8, 10, 12, 14]}
grid_rfc = GridSearchCV(rfc, param_rfc, cv=3, scoring='balanced_accuracy')
grid_rfc.fit(X_train, y_train)
```

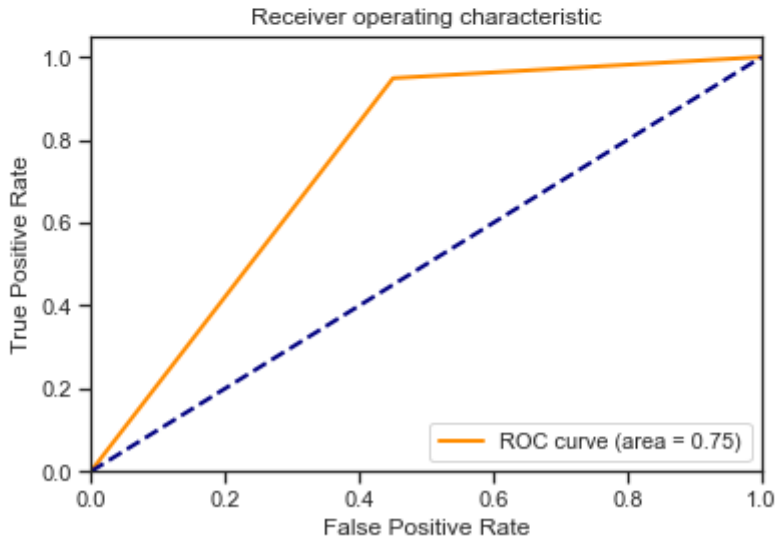
```
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=10, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [1, 3, 5, 7, 10, 13, 16, 19],
                         'n_estimators': [1, 3, 5, 7, 10, 13, 16, 19],
                         'random_state': [0, 2, 4, 6, 8, 10, 12, 14]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

```
grid_rfc.best_params_
```

```
{'max_depth': 3, 'n_estimators': 3, 'random_state': 10}
```

```
rfc2 = RandomForestClassifier(n_estimators=3, max_depth=3, random_state=10)
rfc2.fit(X_train, y_train)
pred_rfc2 = rfc2.predict(X_test)
plot_confusion_matrix(y_test, pred_rfc2,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix RFC2')
draw_roc_curve(y_test.values, pred_rfc2)
balanced_accuracy_score(y_test, pred_rfc2)
```






```
t[64]: 0.7492086349860261
```

Stochastic gradient descent

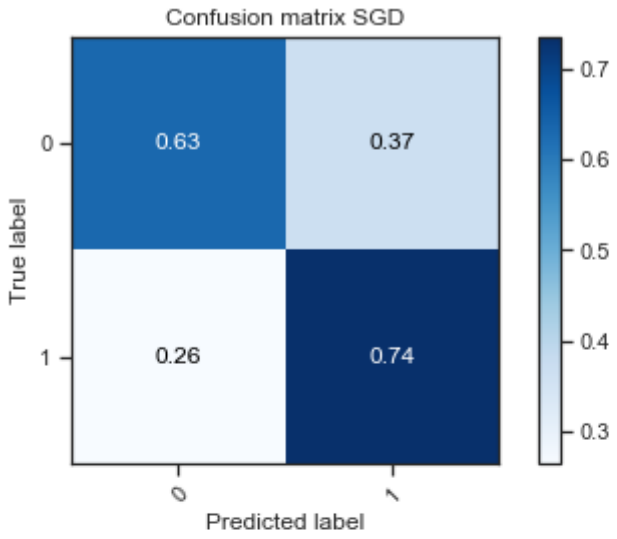
Предполагает, что обучение на каждом шаге происходит не на полном наборе данных, а на одном случайно выбранном примере:

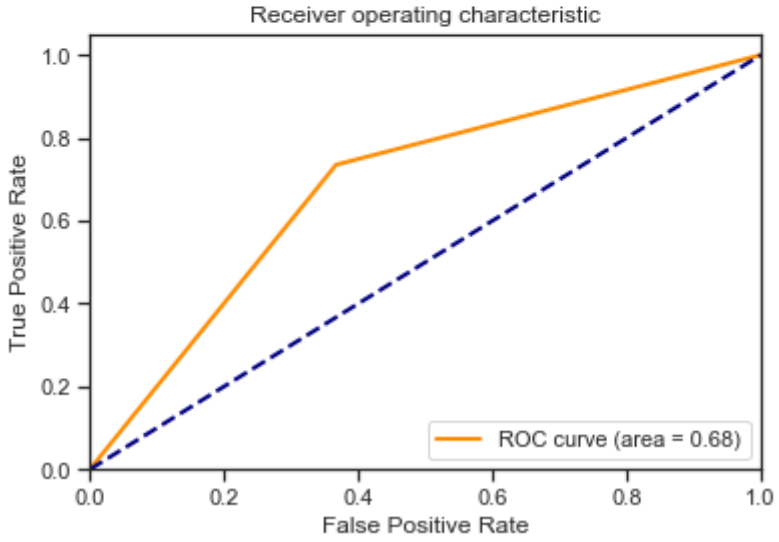
Обучим модель:

```
sgd = SGDClassifier(penalty=None)
sgd.fit(X_train, y_train)
pred_sgd = sgd.predict(X_test)
```

Оценим результаты работы нашей модели:

```
#Оценим результат работы нашей модели
#Стохастический градиентный спуск
plot_confusion_matrix(y_test, pred_sgd,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix SGD')
draw_roc_curve(y_test.values, pred_sgd)
balanced_accuracy_score(y_test, pred_sgd)
```





: 0.6840343154682664

Попробуем улучшить качество модели с помощью подбора гиперпараметров:

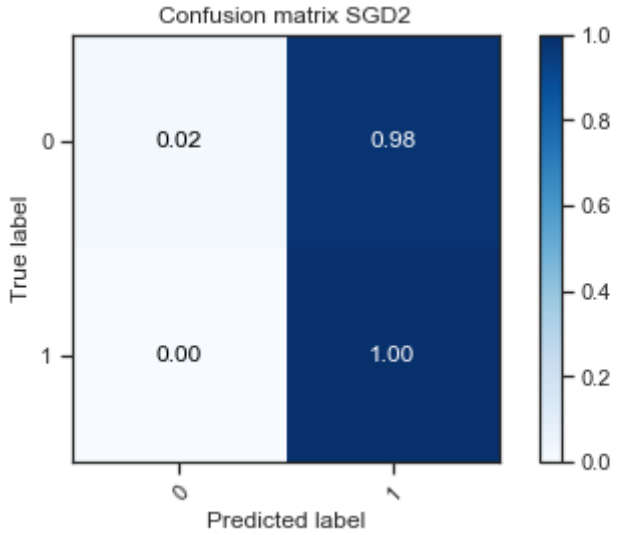
```
param_sgd = {'alpha': [0.5, 0.4, 0.3, 0.2, 0.1]}
grid_sgd = GridSearchCV(sgd, param_sgd, cv=3, scoring='balanced_accuracy')
grid_sgd.fit(X_train, y_train)
```

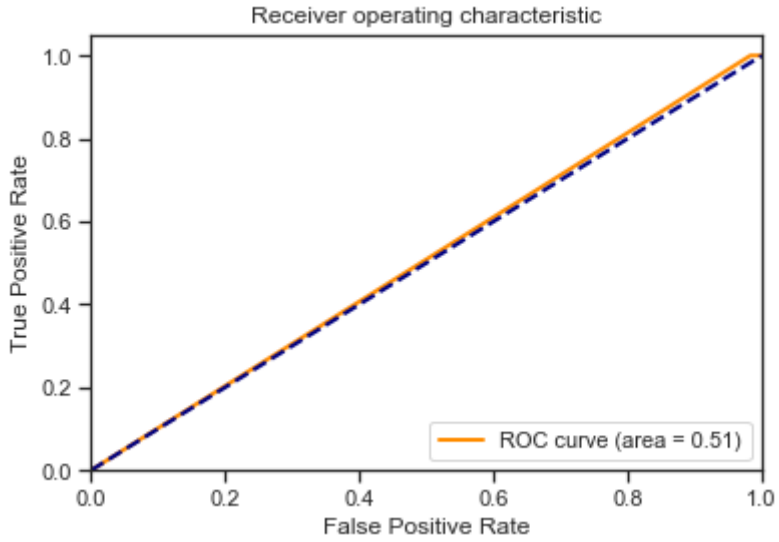
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight=None, early_stopping=False,
                                     epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     l1_ratio=0.15, learning_rate='optimal',
                                     loss='hinge', max_iter=1000,
                                     n_iter_no_change=5, n_jobs=None,
                                     penalty=None, power_t=0.5,
                                     random_state=None, shuffle=True, tol=0.001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.5, 0.4, 0.3, 0.2, 0.1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

```
#Лучшие параметры для модели SGD
grid_sgd.best_params_
```

```
{'alpha': 0.1}
```

```
# Вновь запустим наш SGD с лучшими параметрами
sgd2 = SGDClassifier(alpha=0.4)
sgd2.fit(X_train, y_train)
pred_sgd2 = sgd2.predict(X_test)
plot_confusion_matrix(y_test, pred_sgd2,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix SGD2')
draw_roc_curve(y_test.values, pred_sgd2)
balanced_accuracy_score(y_test, pred_sgd2)
```





Метод ближайших соседей

Исторически является одним из наиболее известных и простых методов классификации. Значение целевого признака определяется на основе значений целевых признаков ближайших объектов.

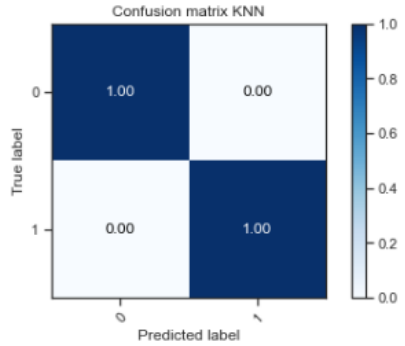
Обучим модель:

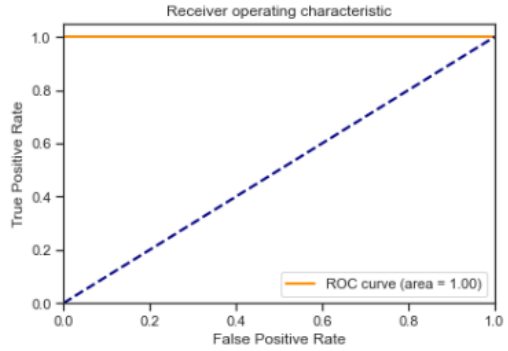
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
pred_knn = knn.predict(X_test)
```

Оценим результаты работы нашей модели:

```
#Оценим результат работы нашей модели
#Ближайшие соседи
plot_confusion_matrix(y_test, pred_knn,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix KNN')
draw_roc_curve(y_test.values, pred_knn)
balanced_accuracy_score(y_test, pred_knn)
```

```
Normalized confusion matrix
```






```
1.0
```

Попробуем улучшить качество модели с помощью подбора гиперпараметров:

```
n_range = np.array(range(1,100,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 1,  6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81,
        86, 91, 96])}]
```

```
grid_knn = GridSearchCV(knn, tuned_parameters, cv=3, scoring='balanced_accuracy')
grid_knn.fit(X_train, y_train)
```

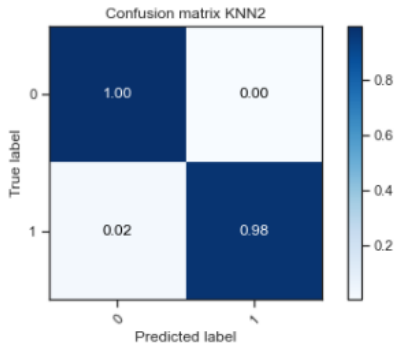
```
GridSearchCV(cv=3, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 1,  6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81,
       86, 91, 96])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

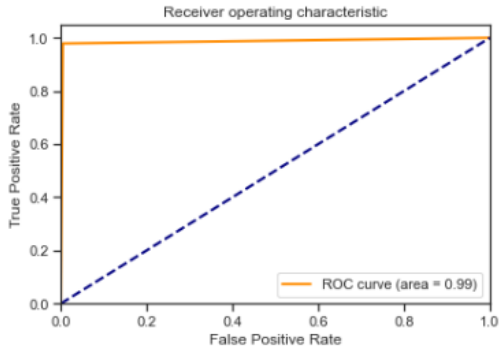
```
#Лучшие параметры для модели KNN
grid_knn.best_params_
```

```
{'n_neighbors': 1}
```

```
# Вновь запустим наш KNN с лучшими параметрами
knn2 = KNeighborsClassifier(n_neighbors=71)
knn2.fit(X_train, y_train)
pred_knn2 = knn2.predict(X_test)
plot_confusion_matrix(y_test, pred_knn2,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix KNN2')
draw_roc_curve(y_test.values, pred_knn2)
balanced_accuracy_score(y_test, pred_knn2)
```

```
Normalized confusion matrix
```






```
0.987001969398576
```

Support Vector Machines

Метод Опорных Векторов или SVM (от англ. Support Vector Machines) — это линейный алгоритм, используемый в задачах классификации и регрессии. Данный алгоритм имеет широкое применение на практике и может решать как линейные, так и нелинейные задачи. Алгоритм создает линию или гиперплоскость, которая разделяет данные на классы. В данной работе будет использоваться метод для решения задачи классификации – SVC.

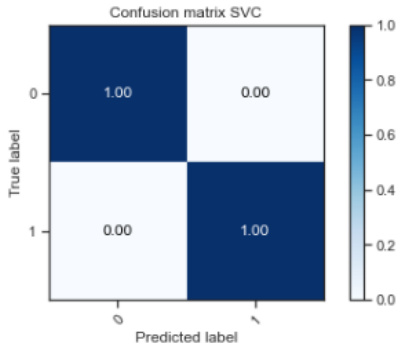
Обучим модель:

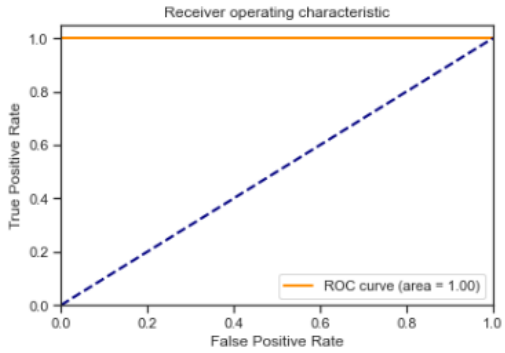
```
svc = SVC()
svc.fit(X_train, y_train)
pred_svc = svc.predict(X_test)
```

Оценим результаты работы нашей модели:

```
#Оценим результат работы нашей модели
#Метод опорных векторов
plot_confusion_matrix(y_test, pred_svc,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix SVC')
draw_roc_curve(y_test.values, pred_svc)
balanced_accuracy_score(y_test, pred_svc)
```

```
Normalized confusion matrix
```






```
1.0
```

Попробуем улучшить качество модели с помощью подбора гиперпараметров:

```
#Поиск оптимальных параметров для модели SVC
param = {
    'C': [0.1,0.8,0.9,1,1.1,1.2,1.3,1.4],
    'kernel':['linear', 'rbf'],
    'gamma': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4]
}
grid_svc = GridSearchCV(svc, param_grid=param, scoring='balanced_accuracy', cv=3)
grid_svc.fit(X_train, y_train)
```

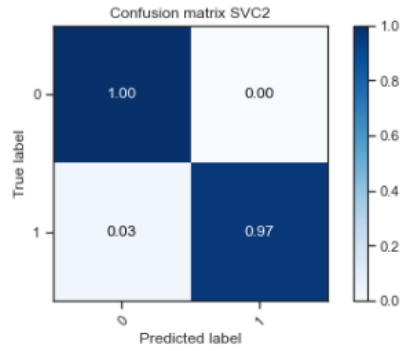
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4],
                         'gamma': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4],
                         'kernel': ['linear', 'rbf']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

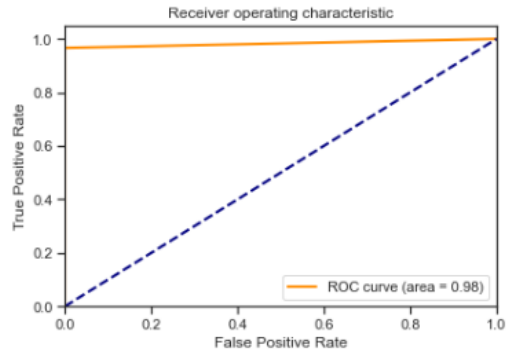
```
#Лучшие параметры для модели SVC
grid_svc.best_params_
```

```
{'C': 0.8, 'gamma': 0.9, 'kernel': 'rbf'}
```

```
# Вновь запустим наш SVC с лучшими параметрами
svc2 = SVC(C = 1.2, gamma =  0.1, kernel= 'rbf')
svc2.fit(X_train, y_train)
pred_svc2 = svc2.predict(X_test)
plot_confusion_matrix(y_test, pred_svc2,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix SVC2')
draw_roc_curve(y_test.values, pred_svc2)
balanced_accuracy_score(y_test, pred_svc2)
```

```
Normalized confusion matrix
```






```
0.9832298136645963
```

Градиентный бустинг

Строится многослойная модель и каждый следующий слой пытается минимизировать ошибку, допущенную на предыдущем слое.

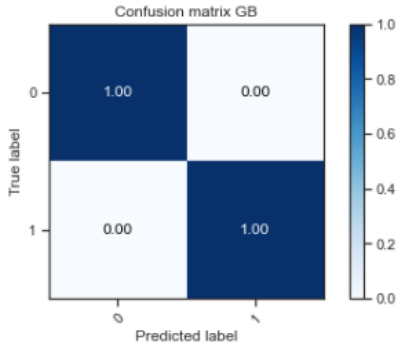
Обучим модель:

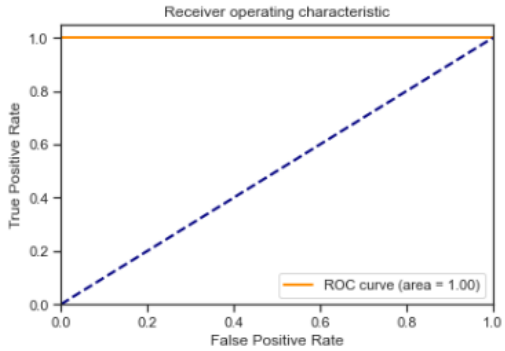
```
gbs = GradientBoostingClassifier()
gbs.fit(X_train, y_train)
pred_gbs = gbs.predict(X_test)
```

Оценим результаты работы нашей модели:

```
#Оценим результат работы нашей модели
#Градиентный бустинг
plot_confusion_matrix(y_test, pred_gbs,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix GB')
draw_roc_curve(y_test.values, pred_gbs)
balanced_accuracy_score(y_test, pred_gbs)
```

```
Normalized confusion matrix
```






```
1.0
```

Попробуем улучшить качество модели с помощью подбора гиперпараметров:

```
param_gbs = {'n_estimators':[1, 3, 5, 7, 10, 13, 16],
                           'max_depth':[1, 3, 5, 7, 10, 13, 16],
                           'learning_rate':[0.01, 0.05, 0.1, 0.5, 2, 3, 4, 5]}
grid_gbs = GridSearchCV(gbs, param_gbs, scoring='balanced_accuracy', cv=3)
grid_gbs.fit(X_train, y_train)
```

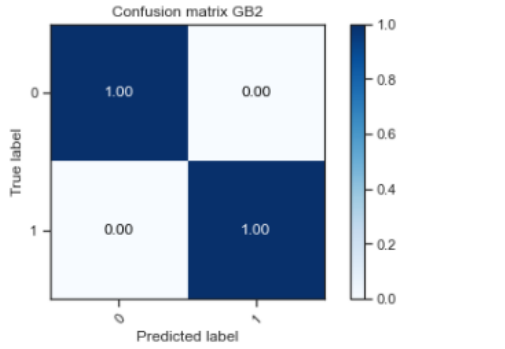
```
GridSearchCV(cv=3, error_score=nan,
             estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                  criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  loss='deviance', max_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_c...
                                                  presort='deprecated',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.01, 0.05, 0.1, 0.5, 2, 3, 4, 5],
                         'max_depth': [1, 3, 5, 7, 10, 13, 16],
                         'n_estimators': [1, 3, 5, 7, 10, 13, 16]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='balanced_accuracy', verbose=0)
```

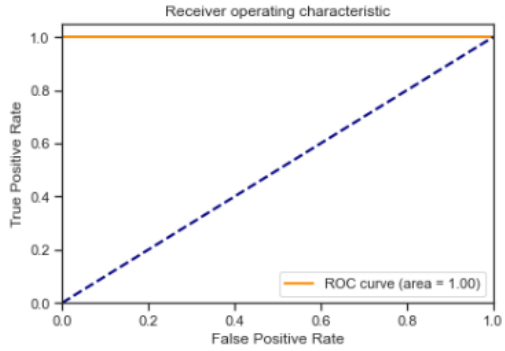
```
#Лучшие параметры для модели GB
grid_gbs.best_params_
```

```
{'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 5}
```

```
# Вновь запустим наш GB с лучшими параметрами
gbs2 = GradientBoostingClassifier(n_estimators=16, max_depth=10, learning_rate=0.5)
gbs2.fit(X_train, y_train)
pred_gbs2 = gbs2.predict(X_test)
plot_confusion_matrix(y_test, pred_gbs2,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix GB2')
draw_roc_curve(y_test.values, pred_gbs2)
balanced_accuracy_score(y_test, pred_gbs2)
```

```
Normalized confusion matrix
```






```
1.0
```

Выводы

В ходе курсовой работы были закреплены полученные в течение курса знания и навыки. Для исследования использовались следующие модели: стохастический градиентный спуск, случайный лес, градиентный бустинг, метод ближайших соседей, метод опорных векторов. Для оценки качества использовались три метрики: ROC-кривая, confusion matrix и balanced_accuracy.

Еще до подбора гиперпараметров почти все модели показали высочайшие характеристики, кроме стохастического градиентного спуска, у которого оказался самый низкий показатель точности. После подбора гиперпараметров несколько методов ухудшили свои показатели: метод опорных векторов, метод ближайших соседей и стохастический градиентный спуск. Все остальные модели смогли показать аналогичное качество.

Приложение

Функции для построения ROC-кривой и матрицы ошибок:

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

```
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
```

```
            rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```

Список использованных источников

1. Конспект лекций по дисциплине «Технологии машинного обучения». 2020: https://github.com/ugapanyuk/ml_course_2020/wiki/COURSE_TMO
2. Документация scikit-learn: https://scikit-learn.org/stable/index.html
3. Метрики в задачах машинного обучения: https://habr.com/ru/company/ods/blog/328372/