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 _____ Болгова А.В.
(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_____

_____Болгова Анастасия Владимировна_____
(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 _____ _____Болгова А.В.
(Подпись, дата) (И.О.Фамилия)

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

Оглавление

1. Задание 4
2. Введение 6
3. Основная часть 6
   Постановка задачи 6
   Описание набора данных 6
   Ход работы 8
      Random Forest 13
      Stochastic gradient descent 15
      Метод ближайших соседей 17
      Support Vector Machines 19
      Градиентный бустинг 21
4. Выводы 23
5. Приложение 24
6. Список использованных источников 26

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/uciml/mushroom-classification

Этот набор данных включает описания гипотетических образцов, соответствующих 23 видам жаберных грибов семейства Agaricus и Lepiota, взятых из полевого руководства общества Audubon по североамериканским грибам (1981). Каждый вид идентифицируется как определенно съедобный, определенно ядовитый или неизвестной съедобности и не рекомендуется. Этот последний класс был объединен с ядовитым. В руководстве ясно сказано, что не существует простого правила для определения съедобности гриба; нет такого правила, как «листочков три, сойдет» для ядовитого дуба и плюща.

Файл *mushrooms.csv* содержит 8124 строки и 23 столбца. В него включены:

Атрибуты:

1-Форма шапки - cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s;

2-Поверхность шапки - cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s;

3-Цвет шапки - cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y;

4-Пятна - bruises: bruises=t, no=f;

5-Запах - odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s;

6-Крепление жабр - gill-attachment: attached=a, descending=d, free=f, notched=n;

7-Расстояние между жабрами - gill-spacing: close=c, crowded=w, distant=d;

8-Размер жабр - gill-size: broad=b, narrow=n;

9-Цвет жабр - gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y;

10-Форма ножки - stalk-shape: enlarging=e, tapering=t;

11-Корень ножки - stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?;

12-Покрытие ножки над кольцом - stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s;

13-Покрытие ножки под кольцом - stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s;

14-Цвет ножки над кольцом - stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y;

15- Цвет ножки под кольцом - stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y;

16- veil-type: partial=p, universal=u;

17- veil-color: brown=n, orange=o, white=w, yellow=y;

18- ring-number: none=n, one=o, two=t;

19- ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z;

20- spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y;

21- population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y;

22- habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d;

Целевой признак (гриб съедобен или ядовит):

23- classes: edible=e, poisonous=p.

Для данного набора данных мы будем решать задачу классификации – определение пригодности гриба в употребление.

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:

```
mushrooms = pd.read_csv('data/mushrooms.csv')
```

Размер датасета:

```
mushrooms.shape
```

```
(8124, 23)
```

Первые пять строк датасета:

```
mushrooms.head()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a

5 rows × 23 columns

Типы столбцов:

```
mushrooms.dtypes
```

```
class                     object
cap-shape                 object
cap-surface               object
cap-color                 object
bruises                   object
odor                      object
gill-attachment           object
gill-spacing              object
gill-size                 object
gill-color                object
stalk-shape               object
stalk-root                object
stalk-surface-above-ring  object
stalk-surface-below-ring  object
stalk-color-above-ring    object
stalk-color-below-ring    object
veil-type                 object
veil-color                object
ring-number               object
ring-type                 object
spore-print-color         object
population                object
habitat                   object
dtype: object
```

Основные статистические характеристики датасета:

```
mushrooms.describe()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	popu
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	...	8124	8124	8124	8124	8124	8124	8124	8124	
unique	2	6	4	10	2	9	2	2	2	12	...	4	9	9	1	4	3	5	9	
top	e	x	y	n	f	n	f	c	b	b	...	s	w	w	p	w	o	p	w	
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	...	4936	4464	4384	8124	7924	7488	3968	2388	

4 rows × 23 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:

```
mushrooms.isnull().sum()
```

```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
dtype: int64
```

Пропущенных значений в наборе данных нет.
Закодируем значения параметров для бинарной классификации:

```
le = LabelEncoder()
for item in mushrooms:
    mushrooms[item] = le.fit_transform(mushrooms[item])
```

```
mushrooms.head()
```

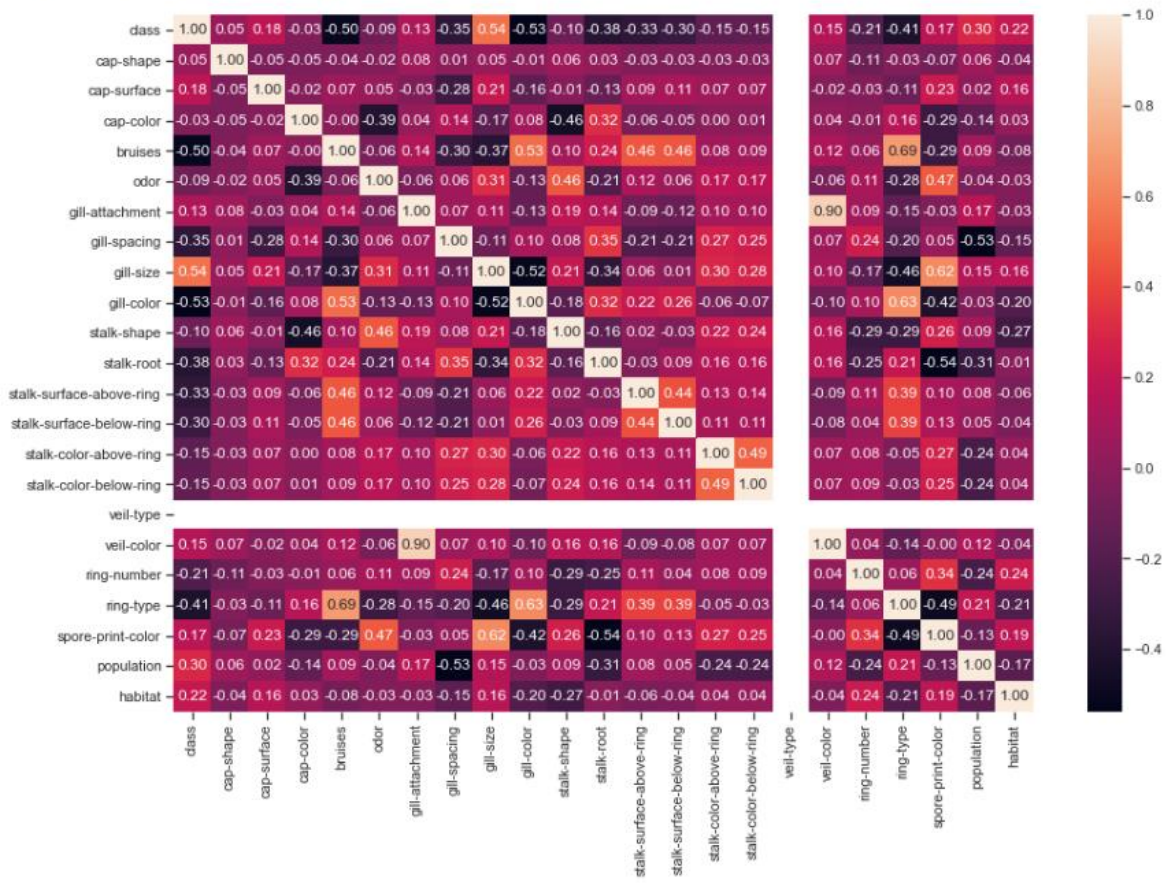
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	1	5	2	4	1	6	1	0	1	4	...	2	7	7	0	2	1	4	2	3
1	0	5	2	9	1	0	1	0	0	4	...	2	7	7	0	2	1	4	3	2
2	0	0	2	8	1	3	1	0	0	5	...	2	7	7	0	2	1	4	3	2
3	1	5	3	8	1	6	1	0	1	5	...	2	7	7	0	2	1	4	2	3
4	0	5	2	3	0	5	1	1	0	4	...	2	7	7	0	2	1	0	3	0

5 rows × 23 columns

Теперь построим корреляционную матрицу:

```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(mushrooms.corr(method='pearson'), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xec76670>
```

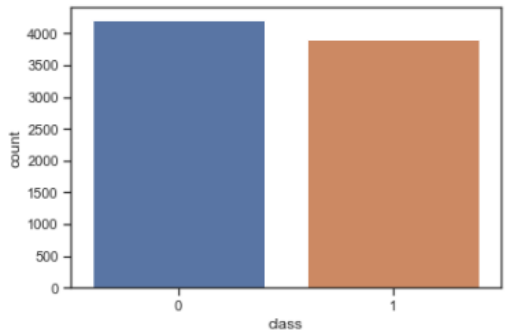


На основе корреляционной матрицы сложно судить о том, насколько качественные модели машинного обучения можно построить, т.к. наиболее коррелирующие признаки с целевым имеют скромные значения, и максимальный из них – gill-size = 0,54.

Распределение данных целевого признака:

```
sns.countplot(mushrooms['class'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x100acbd0>
```



Видно, что дисбаланса классов практически не наблюдается.

Подготовим данные для разделения на обучающую и тестовую выборки:

Разбиение данных на выборки

```
#Подготовка данных
X = mushrooms.drop('class', axis = 1)
y = mushrooms['class']
#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.

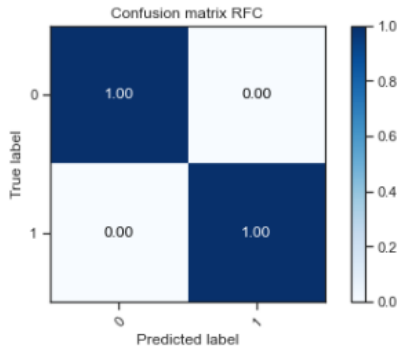
Обучим модель:

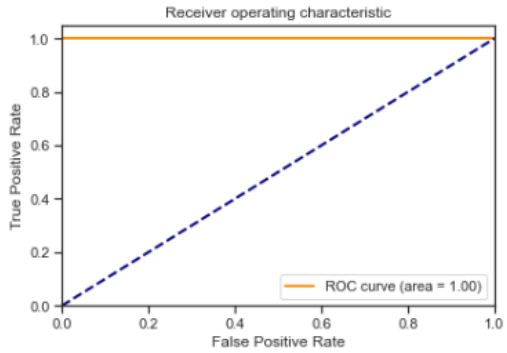
```
rfc = RandomForestClassifier(n_estimators=10)
rfc.fit(X_train, y_train)
pred_rfc = rfc.predict(X_test)
```

Оценим результаты работы нашей модели:

```
#Оценим результат работы нашей 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
```






```
1.0
```

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)
```

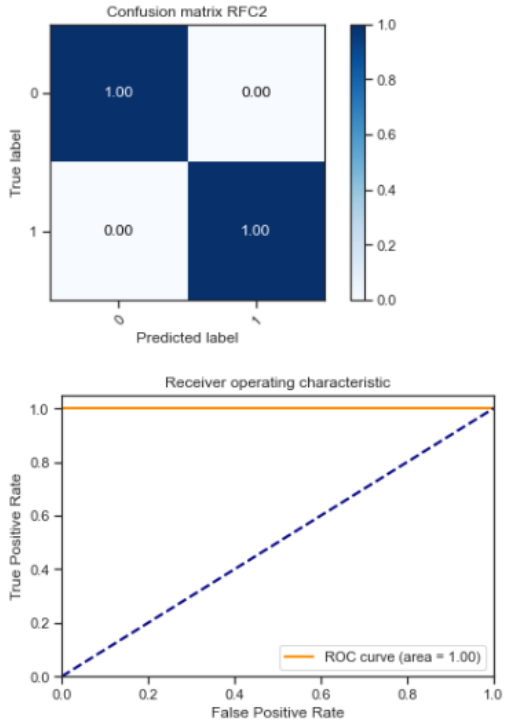
```
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              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)
```

```
#Лучшие параметры для модели RFC
grid_rfc.best_params_
```

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

```
# Вновь запустим наш RFC с лучшими параметрами
rfc2 = RandomForestClassifier(n_estimators=19, max_depth=13, random_state=14)
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)
```

```
Normalized confusion matrix
```




```
1.0
```

Stochastic gradient descent

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

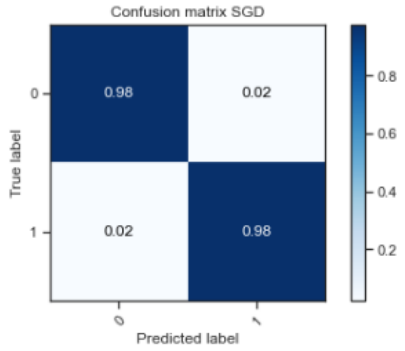
Обучим модель:

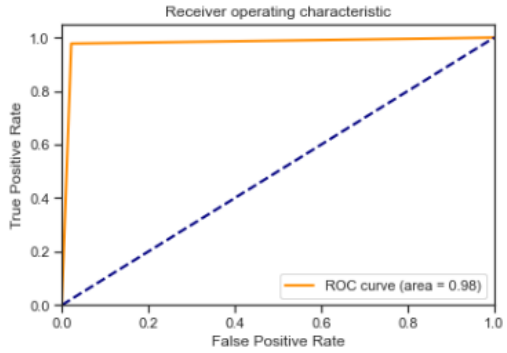
```
sgd = SGDClassifier(penalty=None)
sgd.fit(X_train, y_train)
pred_sgd = sgd.predict(X_test)
```

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)
```

```
Normalized confusion matrix
```






```
0.9778442660203
```

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)
```

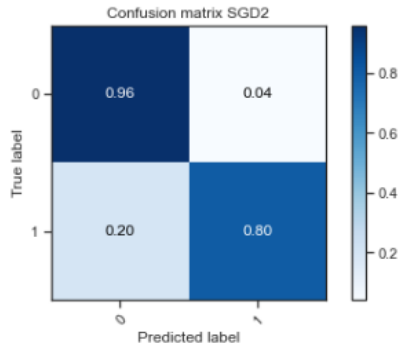
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SGDClassifier(alpha=0.0001, average=False,
                                     class_weight=None, early_stopping=False,
                                     epsilon=0.1, eta0=0.0, fit_intercept=True,
                                     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)
```

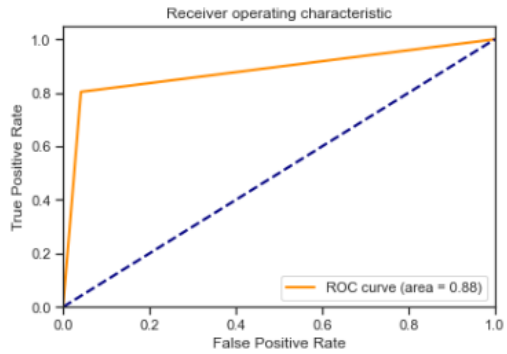
```
#Лучшие параметры для модели SGD
grid_sgd.best_params_
```

```
{'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)
```

```
Normalized confusion matrix
```






```
0.881131646720194
```

Метод ближайших соседей

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

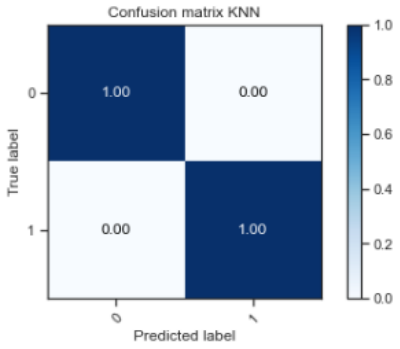
Обучим модель:

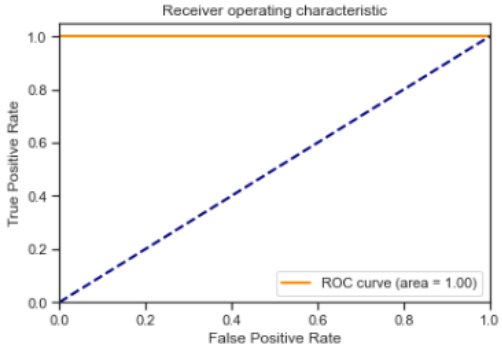
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
pred_knn = knn.predict(X_test)
```

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)
```

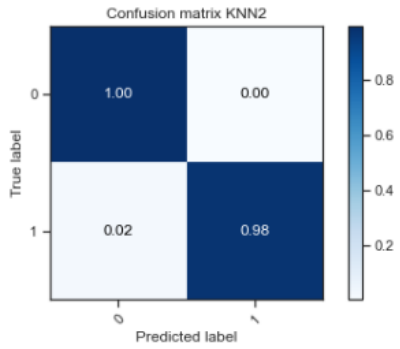
```
GridSearchCV(cv=3, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            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)
```

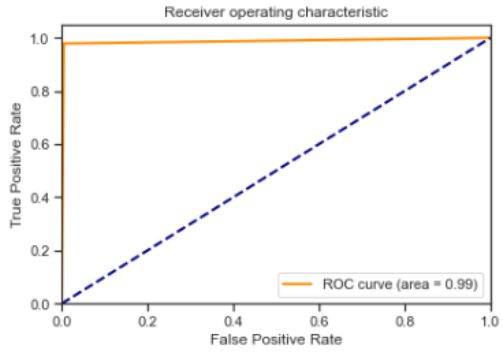
```
#Лучшие параметры для модели KNN
grid_knn.best_params_
```

```
{'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.

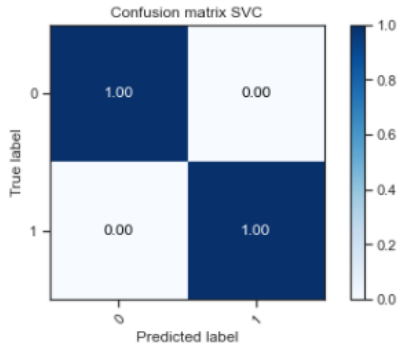
Обучим модель:

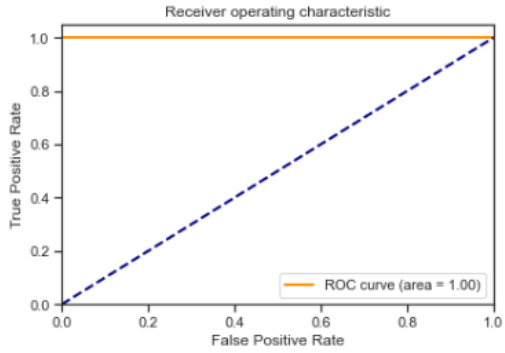
```
svc = SVC()
svc.fit(X_train, y_train)
pred_svc = svc.predict(X_test)
```

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)
```

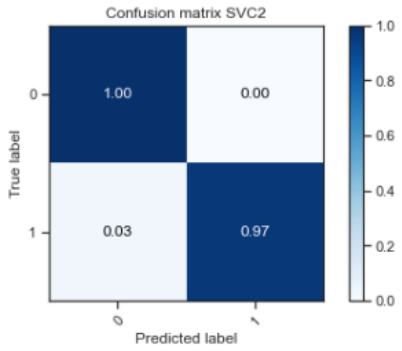
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           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)
```

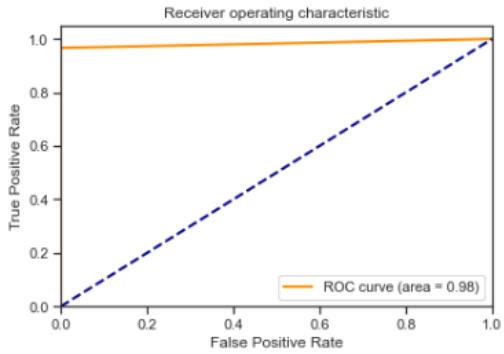
```
#Лучшие параметры для модели SVC
grid_svc.best_params_
```

```
{'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.

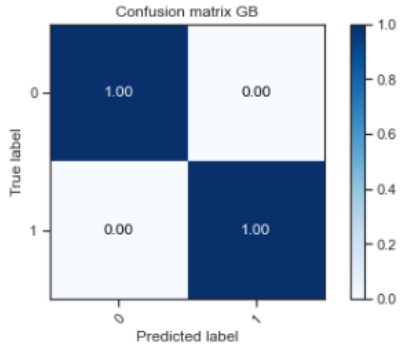
Обучим модель:

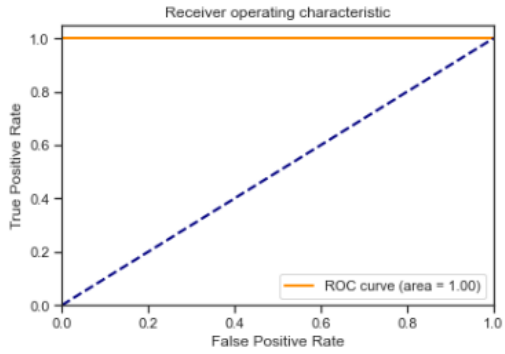
```
gbs = GradientBoostingClassifier()
gbs.fit(X_train, y_train)
pred_gbs = gbs.predict(X_test)
```

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)
```

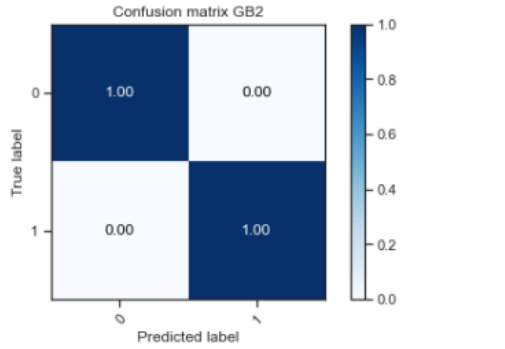
```
GridSearchCV(cv=3, error_score=nan,
             estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                  criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  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)
```

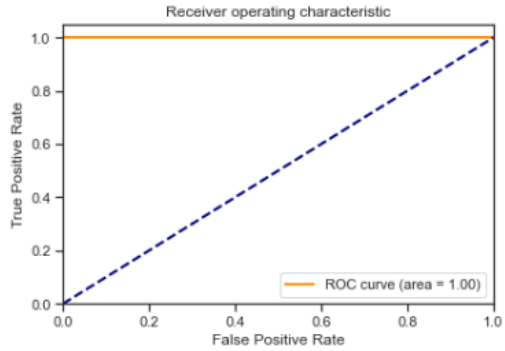
```
#Лучшие параметры для модели GB
grid_gbs.best_params_
```

```
{'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/