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 ИУ_____

КАФЕДРА ИУ5_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

_____Решение задачи машинного обучения_____

Студент группы ИУ5-61Б___ _____ _____ Ерохин И. 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____

_____Ерохин Иван Алексеевич_____
(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 __23___ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « 12 » февраля 2020 г.

Руководитель курсового проекта _____ _____Ерохин И.А.
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____Гапанюк Ю.Е.
(Подпись, дата) (И.О.Фамилия)

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

Оглавление

Задание установленного образца

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.

Основная часть. Описание постановки задачи и последовательности действий по решению поставленной задачи

Описание набора данных

В данной работе для исследований был выбран датасет, содержащий набор данных о различных стальных деталях. Результаты показывают лишь общее качество продукции и результаты проведения некоторого краштеста.

	X_Maximum	Y_Maximum	Pixels_Areas	Sum_of_Luminosity	Empty_Index	Square_Index	LogOfAreas	SigmoidOfAreas	Stains	Dirtiness	Bumps	Other
0	50	270944	267	24220	0.2415	0.1818	2.4265	0.5822	0	0	0	0
1	651	2538108	108	11397	0.3793	0.2069	2.0334	0.2984	0	0	0	0
2	835	1553931	71	7972	0.3426	0.3333	1.8513	0.2150	0	0	0	0
3	860	369415	176	18996	0.4413	0.1556	2.2455	0.5212	0	0	0	0
4	1306	498335	2409	246930	0.4486	0.0662	3.3818	1.0000	0	0	0	0

Damage (оценка от 0 до 4 баллов).

Для данного набора данных мы будем решать задачу классификации – определение повреждений деталей.

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#,
from sklearn.preprocessing import LabelEncoder, StandardScaler
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:

```
details = pd.read_csv('data/faults.csv')
```

Размер датасета:

```
details.shape
```

```
(1941, 13)
```

Первые пять строк датасета:

```
details.head()
```

	X_Maximum	Y_Maximum	Pixels_Areas	Sum_of_Luminosity	Empty_Index	Square_Index	LogOfAreas	SigmoidOfAreas	Stains	Dirtiness	Bumps	Other
0	50	270944	267	24220	0.2415	0.1818	2.4265	0.5822	0	0	0	0
1	651	2538108	108	11397	0.3793	0.2069	2.0334	0.2984	0	0	0	0
2	835	1553931	71	7972	0.3426	0.3333	1.8513	0.2150	0	0	0	0
3	860	369415	176	18996	0.4413	0.1556	2.2455	0.5212	0	0	0	0
4	1306	498335	2409	246930	0.4486	0.0662	3.3818	1.0000	0	0	0	0

Типы столбцов:

```
details.dtypes
```

```
X_Maximum              int64
Y_Maximum              int64
Pixels_Areas           int64
Sum_of_Luminosity      int64
Empty_Index          float64
Square_Index         float64
LogOfAreas           float64
SigmoidOfAreas       float64
Stains                 int64
Dirtiness              int64
Bumps                  int64
Other                  int64
Damage                 int64
dtype: object
```

Основные статистические характеристики датасета:

```
details.describe()
```

	X_Maximum	Y_Maximum	Pixels_Areas	Sum_of_Luminosity	Empty_Index	Square_Index	LogOfAreas	SigmoidOfAreas	Stains	Dirtiness
count	1941.000000	1.941000e+03	1941.000000	1.941000e+03	1941.000000	1941.000000	1941.000000	1941.000000	1941.000000	1941.000000
mean	617.964451	1.650739e+06	1893.878413	2.063121e+05	0.414203	0.570767	2.492388	0.585420	0.037094	0.028336
std	497.627410	1.774590e+06	5168.459560	5.122936e+05	0.137261	0.271058	0.788930	0.339452	0.189042	0.165973
min	4.000000	6.724000e+03	2.000000	2.500000e+02	0.000000	0.006300	0.301000	0.119000	0.000000	0.000000
25%	192.000000	4.712810e+05	84.000000	9.522000e+03	0.315800	0.361300	1.924300	0.246200	0.000000	0.000000
50%	467.000000	1.204136e+06	174.000000	1.920200e+04	0.412100	0.555600	2.240600	0.506300	0.000000	0.000000
75%	1072.000000	2.183084e+06	822.000000	8.301100e+04	0.501600	0.818200	2.914900	0.999800	0.000000	0.000000
max	1713.000000	1.298769e+07	152655.000000	1.159141e+07	0.943900	1.000000	5.183700	1.000000	1.000000	1.000000

Названия колонок:

```
details.columns
```

```
Index(['X_Maximum', 'Y_Maximum', 'Pixels_Areas', 'Sum_of_Luminosity',
       'Empty_Index', 'Square_Index', 'LogOfAreas', 'SigmoidOfAreas', 'Stains',
       'Dirtiness', 'Bumps', 'Other', 'Damage'],
      dtype='object')
```

Пропуски в данных:

```
details.isnull().sum()
```

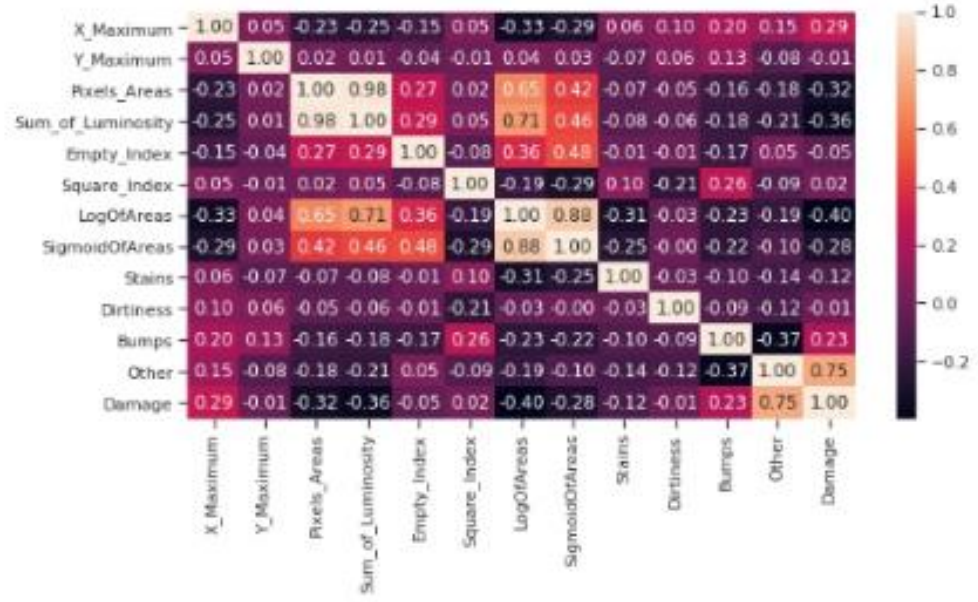
```
X_Maximum            0
Y_Maximum            0
Pixels_Areas         0
Sum_of_Luminosity    0
Empty_Index          0
Square_Index         0
LogOfAreas           0
SigmoidOfAreas       0
Stains               0
Dirtiness            0
Bumps                0
Other                0
Damage               0
dtype: int64
```

Как видно, необходимость заполнения пропусков отсутствует.

Корреляционная матрица:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(details.corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0a67c17b8>
```

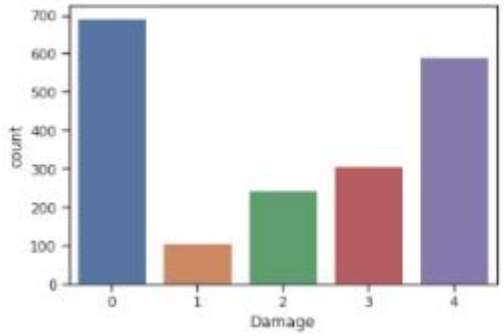


На основе корреляционной матрицы сложно судить о том, насколько качественные модели машинного обучения можно построить, т.к. наиболее коррелирующие признаки с целевым имеют скромные значения.

Распределение данных целевого признака:

```
sns.countplot(details['Damage'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0a659f0f0>
```



Приведем целевой признак к бинарному значению. Для этого обозначим детали с повреждением 2 и ниже хорошими, а 3 и выше – плохим. Это будет говорить о том, что наша задача классификации является задачей бинарной классификации:

```
#Бинарная классификация>
#Деление детали на испорченую или нет в зависимости от уровня повреждения

details['Damage'] = pd.cut(details['Damage'], (-0.1, 2.1, 5), ['1', '0'])
#Используем LabelEncoder для кодирования целевого признака
label_quality = LabelEncoder()
#Испорченая деталь - 0; неиспорченая - 1
details['Damage'] = label_quality.fit_transform(details['Damage'])
details['Damage'].value_counts()
```

```
0    1042
1     899
Name: Damage, dtype: int64
```

Видно, что дисбаланса классов практически не наблюдается.

Подготовим данные для разделения на обучающую и тестовую выборки, а также применим стандартное масштабирование на основе Z-оценки:

```
#Подготовка данных
X = details.drop('Damage', axis = 1)
y = details['Damage']
#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 = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Выберем подходящие для нашей задачи метрики:

1) **Accuracy.**

Метрика вычисляет процент (долю в диапазоне от 0 до 1) правильно определенных классов.

Эту метрику обычно переводят как "точность", но перевод не является удачным, потому что совпадает с другой метрикой - "precision".

Чтобы не сталкиваться с неточностями перевода, названия метрик можно не переводить.

Главная проблема метрики accuracy в том, что она показывает точность по всем классам, но для каждого класса точность может быть разная. Поэтому мы будем использовать **balanced_accuracy**.

2) **Confusion matrix.**

Количество верно и ошибочно классифицированных данных, представленное в виде матрицы. В случае бинарной классификации матрица ошибок выглядит следующим образом:

Предсказанное/истинное значение	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive (TP)	False Positive (FP)
$\hat{y} = 0$	False Negative (FN)	True Negative (TN)

3) **ROC-кривая.**

Используется для оценки качества бинарной классификации. Основана на вычислении следующих характеристик:

$TPR = \frac{TP}{TP+FN}$ - True Positive Rate, откладывается по оси ординат. Совпадает с recall.

$FPR = \frac{FP}{FP+TN}$ - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно. Идеальная ROC-кривая проходит через точки (0,0)-(0,1)-(1,1), то есть через верхний левый угол графика. Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

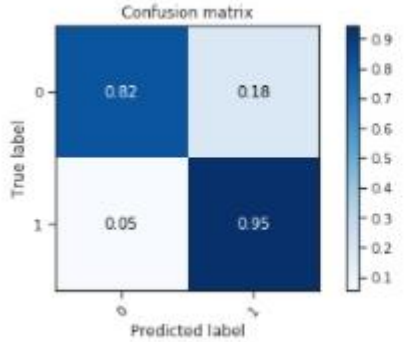
A. Стохастический градиентный спуск

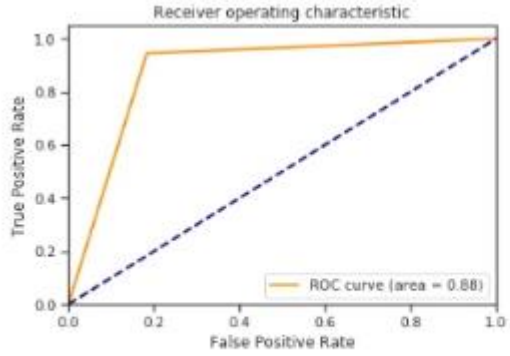
Используем стохастический градиентный спуск (предполагает, что обучение на каждом шаге происходит не на полном наборе данных, а на одном случайно выбранном примере):

Обучим модель:

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

Оценим результаты работы нашей модели:






```
0.8766066838046273
```

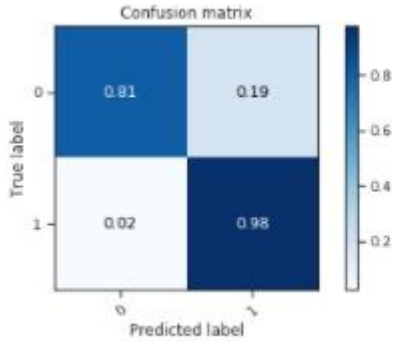
Попробуем улучшить качество модели с помощью подбора лучших гиперпараметров при помощи метода GridSearchCV:

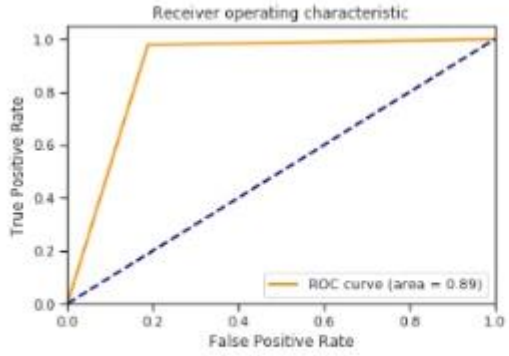
```
GridSearchCV(cv=3, error_score='raise',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=None,
       n_jobs=1, penalty=None, power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'alpha': [0.5, 0.4, 0.3, 0.2, 0.1]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

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

```
{'alpha': 0.2}
```

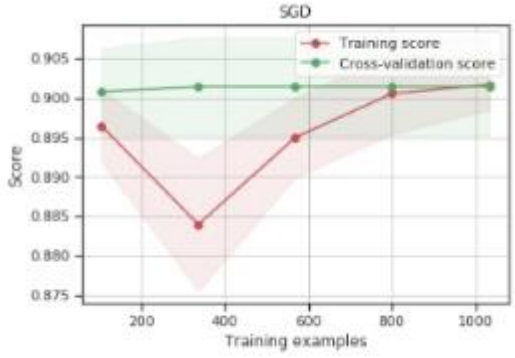
Результат:

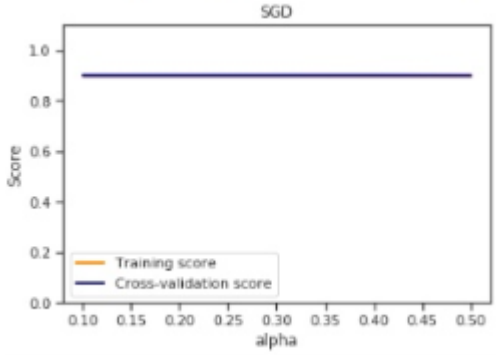




: 0.8894601542416453

Кривые обучения и валидации:





Б. Случайный лес

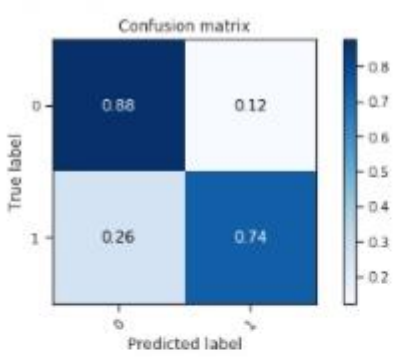
Ансамблевый метод. Каждое решающее дерево строится на случайно выбранном подмножестве признаков. Классификация объектов проводится путём голосования: каждое дерево относит классифицируемый объект к одному из классов, и побеждает класс, за который проголосовало наибольшее число деревьев.

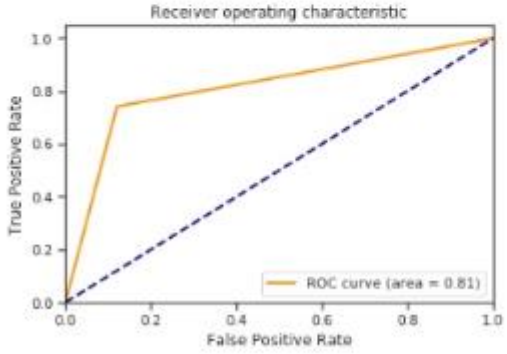
n_estimators - число деревьев, max_depth - максимальная глубина деревьев;

Обучим модель:

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

Оценим результаты работы нашей модели:






```
: 0.8149100257069408
```

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

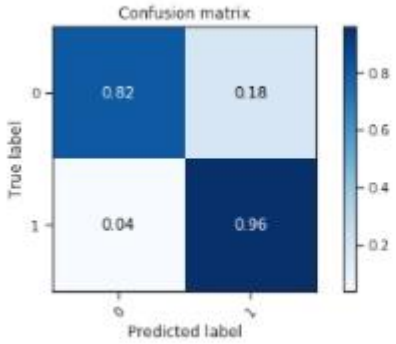
```
param_rfc = {'n_estimators':[1, 3, 5, 7, 10, 13, 16, 19],
    'max_depth':[1, 3, 5, 7, 10, 13, 16, 19],
    'random_state':[0, 2, 4, 6, 8, 10, 12, 14]}
grid_rfc = GridSearchCV(rfc, param_rfc, cv=3, scoring='accuracy')
grid_rfc.fit(X_train, y_train)
```

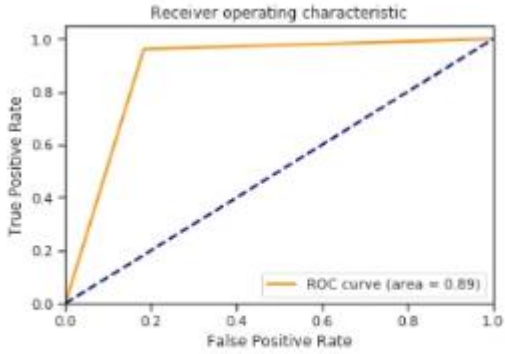
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'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]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

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

```
{'max_depth': 5, 'n_estimators': 19, 'random_state': 4}
```

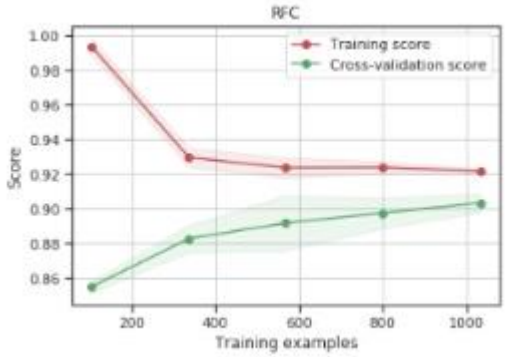
Результат:

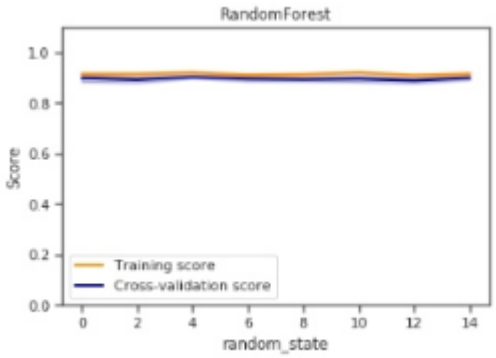





```
: 0.884318766066838
```

Кривые обучения и валидации:





B. Метод ближайших соседей

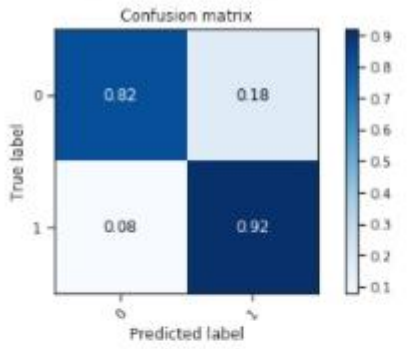
Исторически является одним из наиболее известных и простых методов классификации. Значение целевого признака определяется на основе значений целевых признаков ближайших объектов.

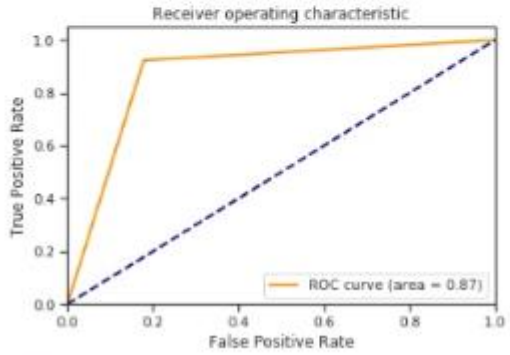
n_neighbors – число соседей;

Обучим модель:

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

Оценим результаты работы нашей модели:






```
0.8688946015424165
```

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

```
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])}]
```

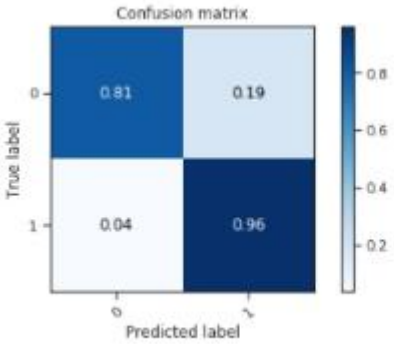
```
grid_knn = GridSearchCV(knn, tuned_parameters, cv=3, scoring='accuracy')
grid_knn.fit(X_train, y_train)
```

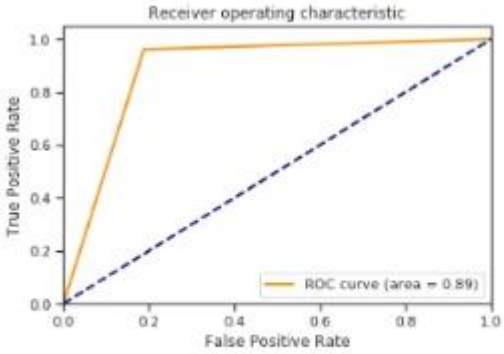
```
GridSearchCV(cv=3, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'n_neighbors': 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='warn',
       scoring='accuracy', verbose=0)
```

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

```
{'n_neighbors': 21}
```

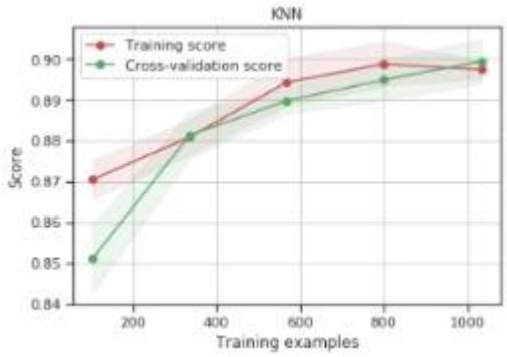
Результат:

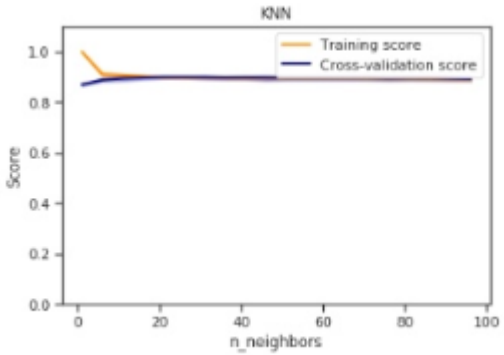





```
: 0.8817480719794345
```

Кривые обучения и валидации:





Г. Метод опорных векторов

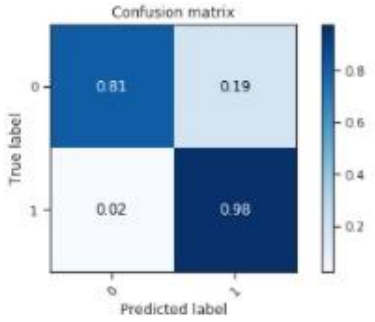
Метод Опорных Векторов или SVM (от англ. Support Vector Machines) — это линейный алгоритм, используемый в задачах классификации и регрессии. Данный алгоритм имеет широкое применение на практике и может решать как линейные, так и нелинейные задачи. Суть работы “Машин” Опорных Векторов проста: алгоритм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

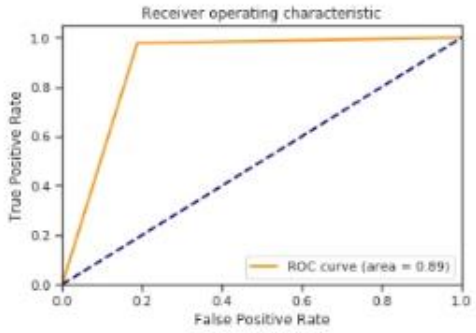
C – помогает отрегулировать ту тонкую грань между “гладкостью” и точностью классификации объектов обучающей выборки, gamma - определяет, насколько далеко каждый из элементов в наборе данных имеет влияние при определении “идеальной линии”, kernel – ядро классификатора;

Обучим модель:

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

Оценим результаты работы нашей модели:






```
0.8894601542416453
```

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

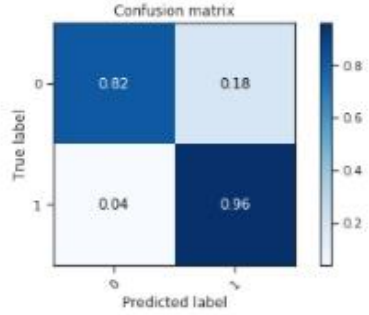
```
#Поиск оптимальных параметров для модели SVC
param = {
    'C': [0.1,0.8,0.9,1,1.1,1.2,1.3,1.4],
    'kernel':['linear', 'rbf'],
    'gamma' :[0.1,0.8,0.9,1,1.1,1.2,1.3,1.4]
}
grid_svc = GridSearchCV(svc, param_grid=param, scoring='accuracy', cv=3)
grid_svc.fit(X_train, y_train)
```

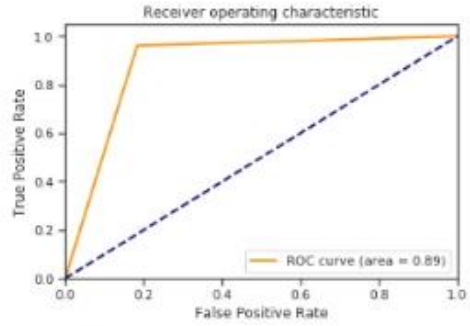
```
GridSearchCV(cv=3, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'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]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

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

```
{'C': 0.1, 'gamma': 0.1, 'kernel': 'linear'}
```

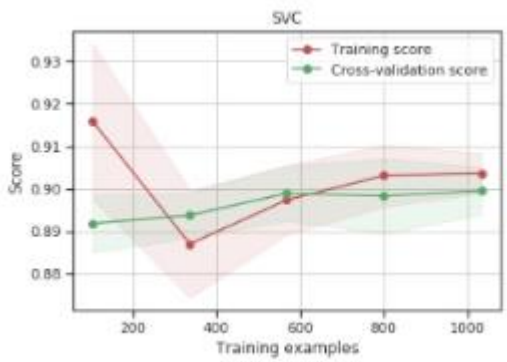
Результат:

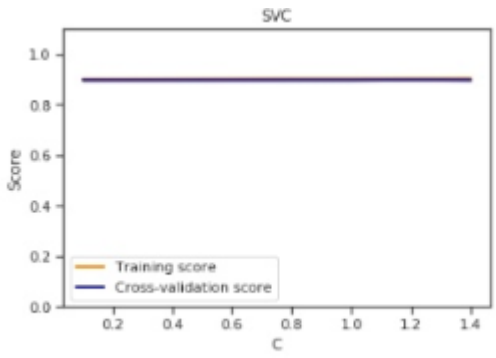





```
: 0.884318766066838
```

Кривые обучения и валидации:





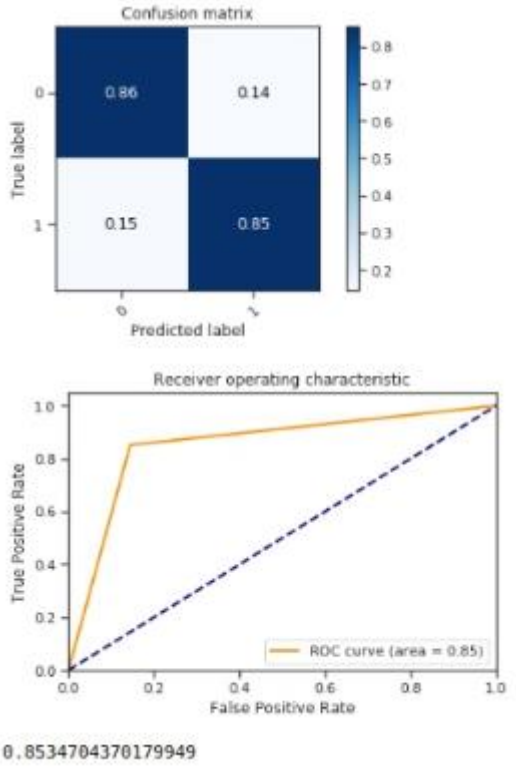
Д. Градиентный бустинг

Ансамблевый метод. 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:




```
0.8534704370179949
```

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

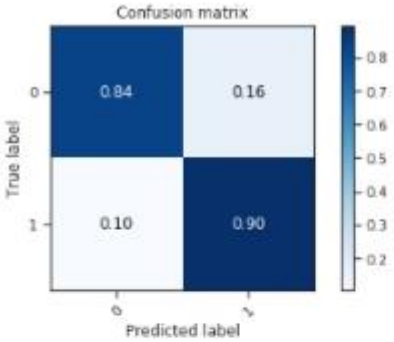
```
param_gbs = {'n_estimators':[1, 3, 5, 7, 10, 13, 16],
                           'max_depth':[1, 3, 5, 7, 10, 13, 16],
                           'learning_rate':[0.01, 0.05, 0.1, 0.5, 2, 3, 4, 5]}
grid_gbs = GridSearchCV(gbs, param_gbs, scoring='accuracy', cv=3)
grid_gbs.fit(X_train, y_train)
```

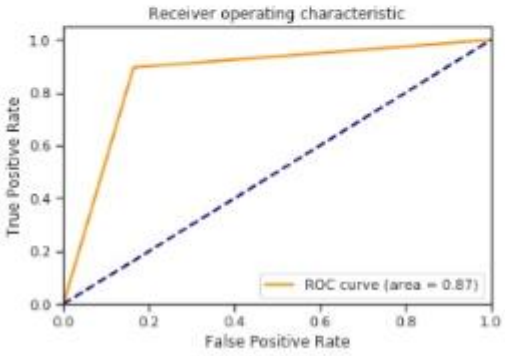
```
GridSearchCV(cv=3, error_score='raise',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,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

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

```
{'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 16}
```

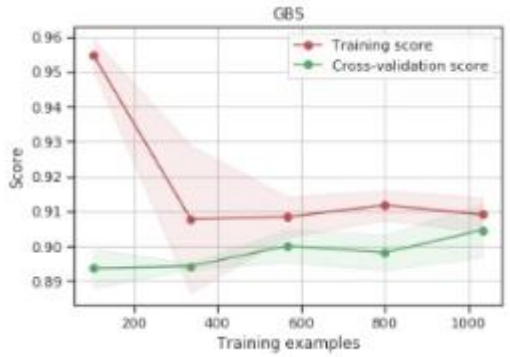
Результат:

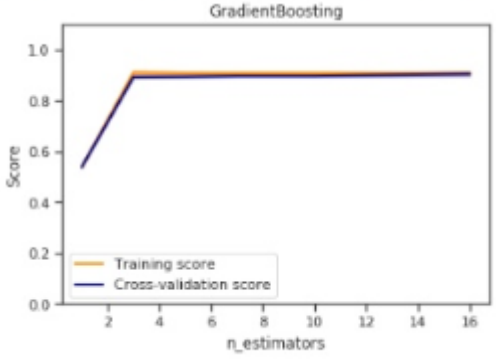





```
: 0.8637532133676092
```

Кривые обучения и валидации:





Функции для построения ROC-кривой, матрицы ошибок, кривых обучения и валидации:

```
# 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
```

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):

    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt
```

```
def plot_validation_curve(estimator, title, X, y,
                          param_name, param_range, cv,
                          scoring="accuracy"):

    train_scores, test_scores = validation_curve(
        estimator, X, y, param_name=param_name, param_range=param_range,
        cv=cv, scoring=scoring, n_jobs=1)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.title(title)
    plt.xlabel(param_name)
    plt.ylabel("Score")
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.plot(param_range, train_scores_mean, label="Training score",
                 color="darkorange", lw=lw)
    plt.fill_between(param_range, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.2,
                     color="darkorange", lw=lw)
    plt.plot(param_range, test_scores_mean, label="Cross-validation score",
                 color="navy", lw=lw)
    plt.fill_between(param_range, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.2,
                     color="navy", lw=lw)
    plt.legend(loc="best")
    return plt
```

Выводы по проделанной работе

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 (SVC). Для оценки качества использовались три метрики: ROC-кривая, confusion matrix и balanced_accuracy. Для наглядности были построены кривые обучения и валидации.

После подбора гиперпараметров наилучшую точность показал стохастический градиентный спуск, при этом метод опорных векторов без подбора гиперпараметров смог показать аналогичное качество (подбор параметров наоборот немного ухудшил качество модели).

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/