Московский государственный технический университет им. Н.Э.Баумана

Утверждаю:

Гапанюк Ю.Е. _____

"__"_____2019 г.

Курсовая работа по курсу Технологии машинного обучения

пояснительная записка
(наименование документа)

19
(количество листов)

Исполнитель:

студентка группы ИУ5-61 Абросимова Н.Г.

"__"_____2019 г.

Москва - 2019

Оглавление

1 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.

2 Введение.

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

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

3 Основная часть, содержащая описание постановки задачи и последовательности действий студента по решению поставленной задачи.

3.1. Описание набора данных

Для исследований был выбран следующий датасет:
https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results
Он содержит набор данных о современных Олимпийских играх, включая все игры от Афин 1896 до Рио 2016.
Файл athlete_events.csv содержит 271116 строк и 15 столбцов. Каждый ряд соответствует индивидуальному спортсмену, участвующему в индивидуальном Олимпийском соревновании
В него включены следующие столбцы:
ID – уникальный номер каждого спортсмена
Name - имя
Sex – пол (M or F)
Age – возраст (Integer)
Height – рост в сантиметрах
Weight – вес в килограммах
Team – имя команды
NOC - Национальный олимпийский комитет (3-буквенный код)
Games - год и сезон
Year - год
Season – лето или зима
City - принимающий город
Sport – вид спорта
Event - событие
Medal – полученная медаль
Будем решать задачу классификации – определение наличия какой-либо медали или её отсутствие.

3.2. Ход работы

Импортируем необходимые для работы библиотеки:

```
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
%matplotlib inline
sns.set(style="ticks")
```

Считываем набор данных:

```
data=pd.read_csv('athlete_events.csv', sep=",")
```

Размер датасета:

```
data.shape
```

```
(271116, 15)
```

Типы столбцов:

```
data.dtypes
```

```
Out[4]:
ID          int64
Name       object
Sex        object
Age       float64
Height    float64
Weight    float64
Team       object
NOC        object
Games      object
Year        int64
Season     object
City       object
Sport      object
Event      object
Medal      object
dtype: object
```

Проверим на наличие пропусков:

```
data.isnull().sum()
```

```
Out[5]:
ID             0
Name           0
Sex            0
Age         9474
Height     60171
Weight     62875
Team           0
NOC            0
Games          0
Year           0
Season         0
City           0
Sport          0
Event          0
Medal     231333
dtype: int64
```

Выведем первые 5 строк:

Out[6]:

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN

Основные статистические характеристики набора данных:

Out[7]:

	ID	Age	Height	Weight	Year
count	271116.000000	261642.000000	210945.000000	208241.000000	271116.000000
mean	68248.954396	25.556898	175.338970	70.702393	1978.378480
std	39022.286345	6.393561	10.518462	14.348020	29.877632
min	1.000000	10.000000	127.000000	25.000000	1896.000000
25%	34643.000000	21.000000	168.000000	60.000000	1960.000000
50%	68205.000000	24.000000	175.000000	70.000000	1988.000000
75%	102097.250000	28.000000	183.000000	79.000000	2002.000000
max	135571.000000	97.000000	226.000000	214.000000	2016.000000

Пропуски были обнаружены в 4 столбцах: Age, Height, Weight и Medal. В первых трёх заполним пропуски средними значениями, в последнем – конкретным значением, обозначающим, что медалей нет.

```
data.loc[:, 'Medal']=data.loc[:, 'Medal'].fillna('Net')
data.loc[:, 'Age']=data.loc[:, 'Age'].fillna(data['Age'].mean())
data.loc[:, 'Height']=data.loc[:, 'Height'].fillna(data['Height'].mean())
data.loc[:, 'Weight']=data.loc[:, 'Weight'].fillna(data['Weight'].mean())
```

Проверим:

```
data.isnull().sum()
```

Out[15]:

```
ID         0
Name       0
Sex        0
Age        0
Height     0
Weight     0
Team       0
NOC        0
Games      0
Year       0
Season     0
City       0
Sport      0
Event      0
Medal      0
dtype: int64
```

Кодирование категориальных признаков:

```
le=LabelEncoder()
le.fit(data.Sex)
data['Sex']=le.transform(data.Sex)
le.fit(data.Team)
data['Team']=le.transform(data.Team)
le.fit(data.Season)
data['Season']=le.transform(data.Season)
le.fit(data.Name)
data['Name']=le.transform(data.Name)
le.fit(data.NOC)
data['NOC']=le.transform(data.NOC)
le=LabelEncoder()
le.fit(data.Games)
data['Games']=le.transform(data.Games)
le=LabelEncoder()
le.fit(data.City)
data['City']=le.transform(data.City)
le=LabelEncoder()
le.fit(data.Sport)
data['Sport']=le.transform(data.Sport)
le.fit(data.Event)
data['Event']=le.transform(data.Event)
le.fit(data.Medal)
data['Medal']=le.transform(data.Medal)
```

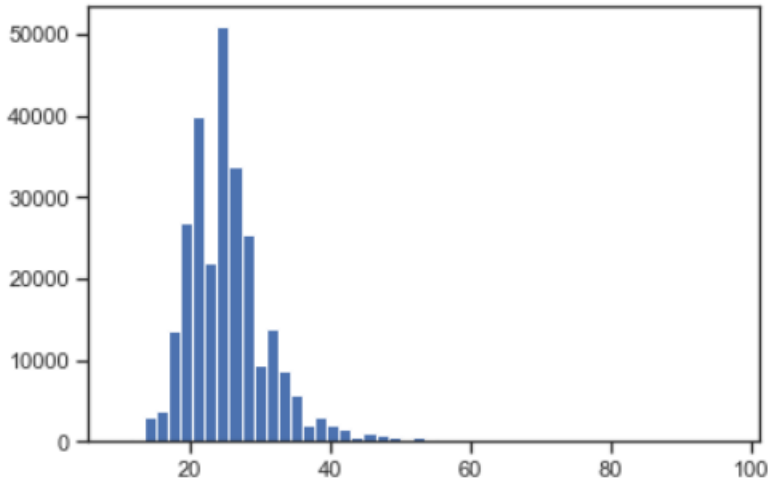
Итог:

Масштабирование данных:

```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[['Age']])
plt.hist(data['Age'], 50)
plt.show()
```

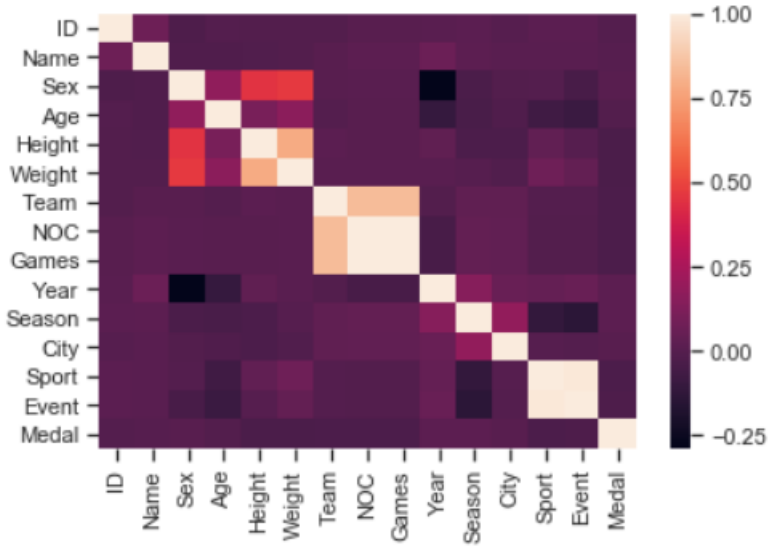
Out[26]:

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	7	1	24.0	180.00000	80.000000	198	41	41	1992	0	5	8	159	2
1	2	8	1	23.0	170.00000	60.000000	198	41	41	2012	0	17	32	397	2
2	3	44094	1	24.0	175.33897	70.702393	273	55	55	1920	0	2	24	348	2
3	4	29258	1	34.0	175.33897	70.702393	278	55	55	1900	0	26	61	709	1
4	5	21425	0	21.0	185.00000	82.000000	704	145	145	1988	1	8	53	622	2
5	5	21425	0	21.0	185.00000	82.000000	704	145	145	1988	1	8	53	618	2
6	5	21425	0	25.0	185.00000	82.000000	704	145	145	1992	1	0	53	622	2
7	5	21425	0	25.0	185.00000	82.000000	704	145	145	1992	1	0	53	618	2
8	5	21425	0	27.0	185.00000	82.000000	704	145	145	1994	1	16	53	622	2
9	5	21425	0	27.0	185.00000	82.000000	704	145	145	1994	1	16	53	618	2
10	6	99749	1	31.0	188.00000	75.000000	1095	216	216	1992	1	0	17	227	2
11	6	99749	1	31.0	188.00000	75.000000	1095	216	216	1992	1	0	17	235	2



Корреляционная матрица после выполненных операций:

```
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1d7dfeecb00>
```



Удалим сильно коррелирующие между собой столбцы (рост и вес).
Сформируем тестовую и обучающую выборку:

```
X_train, X_test, y_train, y_test = train_test_split(
    data, data['Medal'], test_size= 0.4, random_state= 1)
```

```
# Размер обучающей выборки
X_train.shape, y_train.shape
```

```
Out[32]:
```

```
((162669, 15), (162669,))
# Размер тестовой выборки
X_test.shape, y_test.shape
Out[33]:
((108447, 15), (108447,))
```

В качестве метрик будем использовать accuracy, precision и recall.
Accuracy вычисляет процент (долю в диапазоне от 0 до 1) правильно определенных классов.
Precision показывает долю верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.
Recall определяет долю верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.
Эти метрики позволяют наглядно понять, насколько хорошо проходит классификация.

1. **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)

```
#обучение линейной модели
sgd = SGDClassifier().fit(X_train, y_train)
```

Предсказание:

```
target_sgd = sgd.predict(X_test)
```

Найдём оптимальное значение альфы.

```
scores_sgd = cross_val_score(SGDClassifier(),
                             X_train, y_train, cv=2)
scores_sgd
```

```
parameters = {'alpha':[0.5,0.4,0.3,0.2,0.1]}
clf_gs_sgd = GridSearchCV(SGDClassifier(), parameters, cv=2, scoring='accu
racy')
clf_gs_sgd.fit(X_train, y_train)
```

```
GridSearchCV(cv=2, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=N
one,
       early_stopping=False, 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_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', 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='warn',
       scoring='accuracy', verbose=0)
```

```
#оптимальный гиперпараметр
clf_gs_sgd.best_params_
Out[38]:
{'alpha': 0.5}
#для 0.5:
sgd2 = SGDClassifier(alpha=0.5).fit(X_train, y_train)
```

```
target_sgd2= sgd2.predict(X_test)
```

Сравним значения метрик при разных значениях альфы:

```
accuracy_score(y_test, target_sgd),\
accuracy_score(y_test, target_sgd2)
```

```
Out[45]:
 (0.8475845343808497, 0.8534768135586969)
```

```
precision_score(y_test, target_sgd, average='micro') ,\
precision_score(y_test, target_sgd2, average='micro')
```

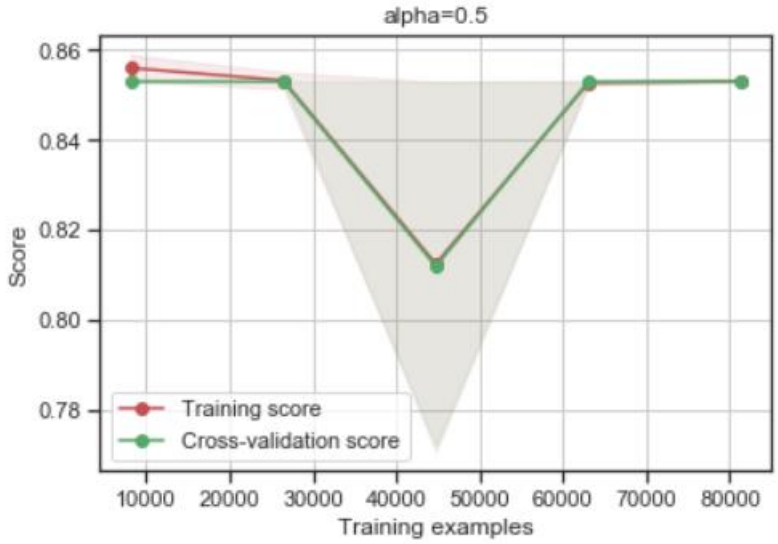
```
Out[46]:
(0.8475845343808497, 0.8534768135586969)
```

```
recall_score(y_test, target_sgd, average='micro'),\
recall_score(y_test, target_sgd2, average='micro')
```

```
Out[47]:
(0.8475845343808497, 0.8534768135586969)
```

Кривые обучения:

```
plot_learning_curve(SGDClassifier(alpha=0.5), 'alpha=0.5', X_train,
y_train, cv=2)
```



Кривые валидации:

```
plot_validation_curve(SGDClassifier(alpha=0.5), 'SGD',
                      X_train, y_train,
                      param_name='alpha', param_range=n_range,
                      cv=2, scoring="accuracy")
```

2. Дерево решений

При помощи дерева решений решаются задачи классификации и прогнозирования. Дерево решений – это схематическое представление проблемы принятия решений. Ветви дерева решений представляют собой различные события (решения), а его вершины – ключевые состояния, в которых возникает необходимость выбора.
max_depth – максимальная глубина дерева, min_impurity_decrease – минимальное сокращение

Обучение:

```
tree = DecisionTreeClassifier(random_state=1, max_depth=0.75).fit(X_train, y_train)
```

Предсказание:

```
target_tree = tree.predict(X_test)
```

Поиск оптимальных параметров:

```
scores_decision_tree = cross_val_score(DecisionTreeClassifier(),
                          X_train, y_train, cv=2)
scores_decision_tree
parameters = {'min_impurity_decrease':[0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.2,0.1]}
clf_gs_decision_tree = GridSearchCV(DecisionTreeClassifier(), parameters, cv=2, scoring='accuracy')
clf_gs_decision_tree.fit(X_train, y_train)
```

```
clf_gs_decision_tree.best_params_
```

```
Out[53]:
```

```
{'min_impurity_decrease': 0.1}
```

При оптимальных параметрах:

```
tree2 = DecisionTreeClassifier(random_state=1, min_impurity_decrease=0.1, max_depth=0.75).fit(X_train, y_train)
target_tree2 = tree2.predict(X_test)
```

Сравнение:

```
accuracy_score(y_test, target_tree), \
accuracy_score(y_test, target_tree2)
```

```
Out[56]:
```

```
(0.8536612354421975, 0.8536612354421975)
```

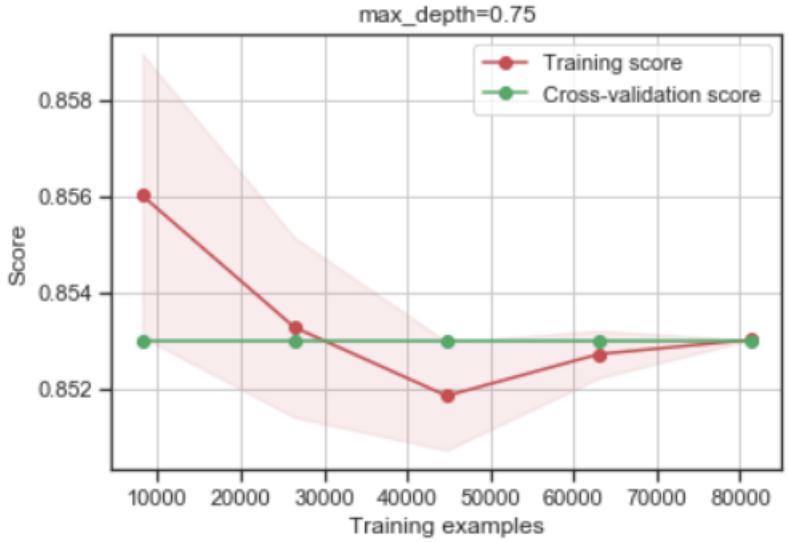
```
precision_score(y_test, target_tree, average='micro') ,\
precision_score(y_test, target_tree2, average='micro')
```

```
Out[57]:
```

```
(0.8536612354421975, 0.8536612354421975)
recall_score(y_test, target_tree, average='micro'),\
recall_score(y_test, target_tree2, average='micro')
Out[58]:
(0.8536612354421975, 0.8536612354421975)
```

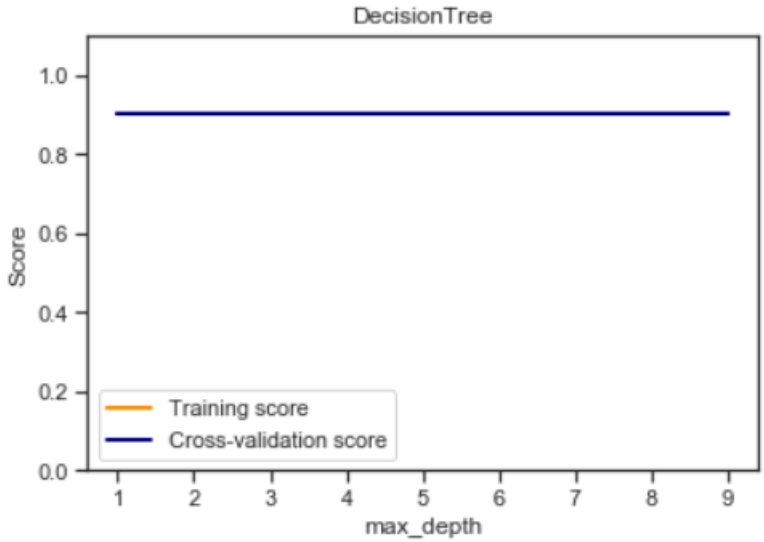
Кривые обучения:

```
plot_learning_curve(DecisionTreeClassifier(random_state=1, min_impurity_de
crease=0.1, max_depth=0.75), 'max_depth=0.75',
                    X_train, y_train, cv=2)
```



Кривые валидации:

```
plot_validation_curve(DecisionTreeClassifier(random_state=1, min_impurity_
decrease=0.1, max_depth=0.75), 'DecisionTree',
                      X_train, y_train,
                      param_name="max_depth", param_range=n_range,
                      cv=2, scoring="accuracy")
```



3. **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

Обучение:

```
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
#случайный лес
randomforest = RandomForestClassifier(n_estimators=5, max_depth=1, random_
state=0).fit(X_train, y_train)
```

Предсказание:

```
target_randomforest = randomforest.predict(X_test)
```

Поиск оптимальных параметров:

```
parameters_random_forest = {'n_estimators':[1, 3, 5, 7, 10],
                            'max_depth':[1, 3, 5, 7, 10],
                            'random_state':[0, 2, 4, 6, 8, 10]}
best_random_forest = GridSearchCV(RandomForestClassifier(), parameters_ran
dom_forest, cv=3, scoring='accuracy')
best_random_forest.fit(X_train, y_train)
```

```
#оптимальные гиперпараметры
best_random_forest.best_params_
```

```
Out[64]:
```

```
{'max_depth': 10, 'n_estimators': 10, 'random_state': 0}
```

Для найденных значений:

```
randomforest2 = RandomForestClassifier(n_estimators=10, max_depth=10, rand
om_state=0).fit(X_train, y_train)
target_randomforest2 = randomforest2.predict(X_test)
```

Сравнение:

```
accuracy_score(y_test, target_randomforest),\
accuracy_score(y_test, target_randomforest2)
```

```
Out[67]:
```

```
(0.8536612354421975, 0.9914243824172176)
```

```
precision_score(y_test, target_randomforest, average='micro') ,\
precision_score(y_test, target_randomforest2, average='micro')
```

```
Out[68]:
```

```
(0.8536612354421975, 0.9914243824172176)
```

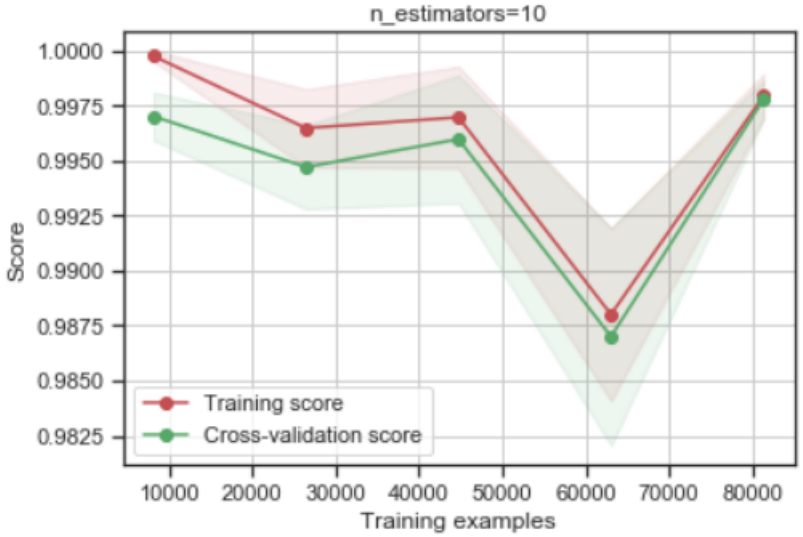
```
recall_score(y_test, target_randomforest, average='micro') ,\
recall_score(y_test, target_randomforest2, average='micro')
```

```
Out[69]:
```

```
(0.8536612354421975, 0.9914243824172176)
```

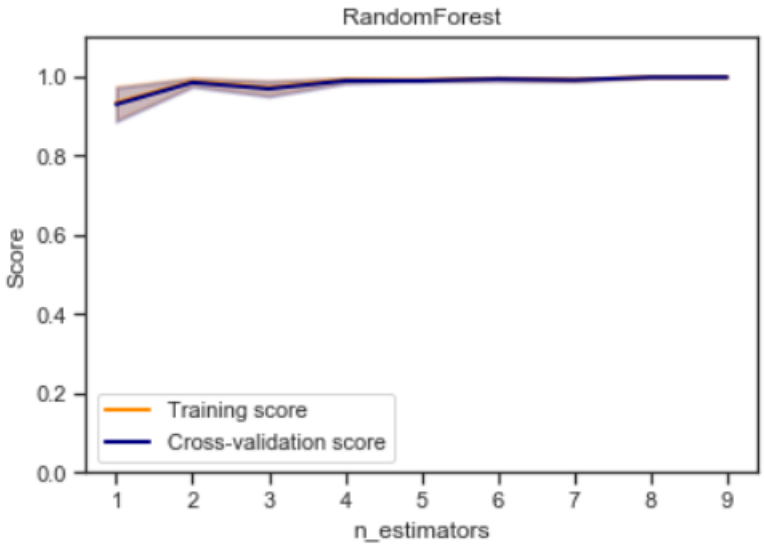
Кривые обучения:

```
plot_learning_curve(RandomForestClassifier(n_estimators=10, max_depth=10,
random_state=0), 'n_estimators=10', X_train, y_train, cv=2)
```



Кривые валидации:

```
plot_validation_curve(RandomForestClassifier(n_estimators=10, max_depth=10
, random_state=0), 'RandomForest',
                      X_train, y_train,
                      param_name='n_estimators', param_range=n_range,
                      cv=2, scoring="accuracy")
```



4. Градиентный бустинг

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

```
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingR
egressor
```

Обучение:

```
#градиентный бустинг
gradient_boosting = GradientBoostingClassifier(n_estimators=5, max_depth=1
, learning_rate=0.01).fit(X_train, y_train)
```

Предсказание:

```
target_gradient_boosting = gradient_boosting.predict(X_test)
```

Поиск оптимальных гиперпараметров:

```
parameters_gradient_boosting = {'n_estimators':[1, 3, 5, 7, 10],
                                'max_depth':[1, 3, 5, 7, 10],
                                'learning_rate':[0.001, 0.0025, 0.005, 0.0075,
0.01, 0.025]}
```

```
best_gradient_boosting = GridSearchCV(GradientBoostingClassifier(), parame
ters_gradient_boosting, cv=3, scoring='accuracy')
best_gradient_boosting.fit(X_train, y_train)
```

```
#оптимальные гиперпараметры
best_gradient_boosting.best_params_
```

```
Out[74]:
{'learning_rate': 0.025, 'max_depth': 3, 'n_estimators': 10}
```

Для найденных значений:

```
gradient_boosting2 = GradientBoostingClassifier(n_estimators=10, max_depth
=3, learning_rate=0.025).fit(X_train, y_train)
target_gradient_boosting2 = gradient_boosting2.predict(X_test)
```

Сравнение:

```
accuracy_score(y_test, target_gradient_boosting),\
accuracy_score(y_test, target_gradient_boosting2)
```

```
Out[77]:
(0.8536612354421975, 1.0)
```

```
precision_score(y_test, target_gradient_boosting, average='micro') ,\
precision_score(y_test, target_gradient_boosting2, average='micro')
```

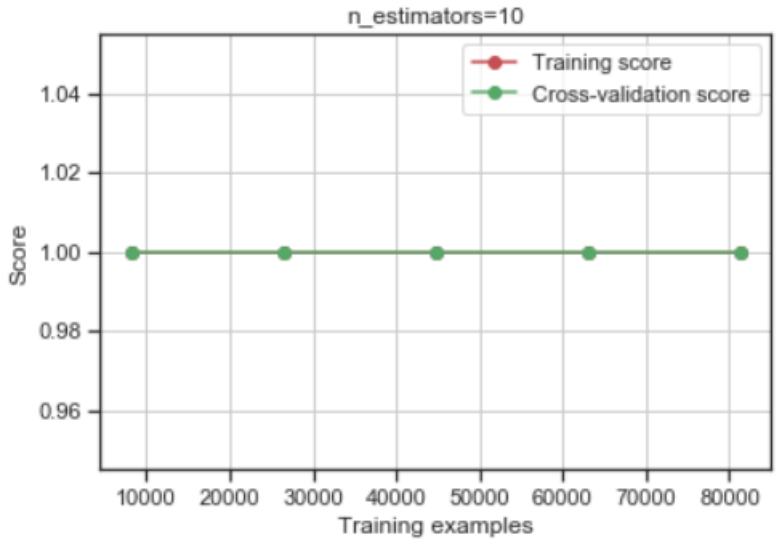
```
Out[78]:
(0.8536612354421975, 1.0)
```

```
recall_score(y_test, target_gradient_boosting, average='micro') ,\
recall_score(y_test, target_gradient_boosting2, average='micro')
```

```
Out[79]:
(0.8536612354421975, 1.0)
```

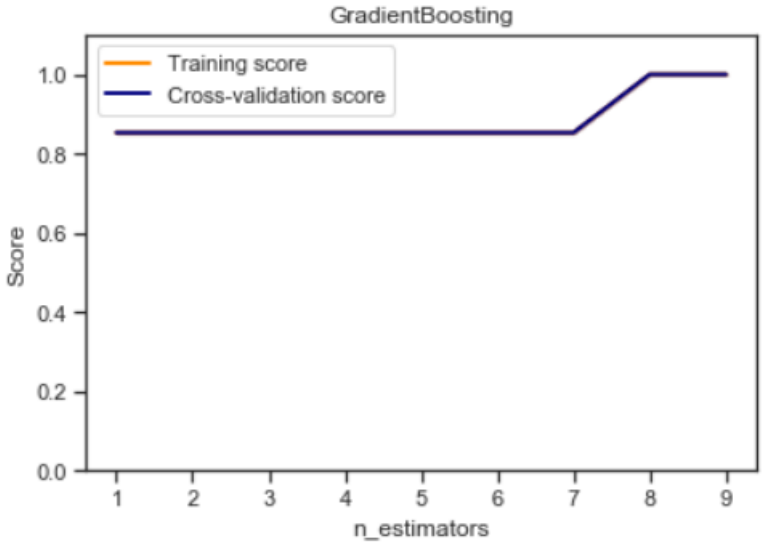
Кривые обучения:

```
plot_learning_curve(GradientBoostingClassifier(n_estimators=10, max_depth=
3, learning_rate=0.025), 'n_estimators=10',
                    X_train, y_train, cv=2)
```



Кривые валидации:

```
plot_validation_curve(GradientBoostingClassifier(n_estimators=10, max_dept
h=3, learning_rate=0.025), 'GradientBoosting',
                      X_train, y_train,
                      param_name='n_estimators', param_range=n_range,
                      cv=2, scoring="accuracy")
```



5. 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
Обучение:

```
simple_knn = KNeighborsClassifier(n_neighbors=2)
simple_knn.fit(X_train, y_train)
```

Предсказание:

```
target_1 = simple_knn.predict(X_test)
```

Поиск оптимального значения:

```
from sklearn.model_selection import StratifiedKFold
n_range = np.array(range(1,10,1))
tuned_parameters = [{'n_neighbors': n_range}]
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters,
                      cv=StratifiedKFold(n_splits=5), scoring='accuracy')
clf_gs.fit(X_train, y_train)
```

```
clf_gs.best_params_
```

```
Out[40]:
```

```
{'n_neighbors': 9}
```

Для найденного значения:

```
simple_knn2 = KNeighborsClassifier(n_neighbors=9)
simple_knn2.fit(X_train, y_train)
target_2 = simple_knn2.predict(X_test)
```

Сравнение:

```
accuracy_score(y_test, target_1),\
accuracy_score(y_test, target_2)
```

```
Out[44]:
```

```
(0.7428697889291543, 0.8504522946692854)
```

```
precision_score(y_test, target_1, average='micro') ,\
precision_score(y_test, target_2, average='micro')
```

```
Out[45]:
```

```
(0.7428697889291543, 0.8504522946692854)
```

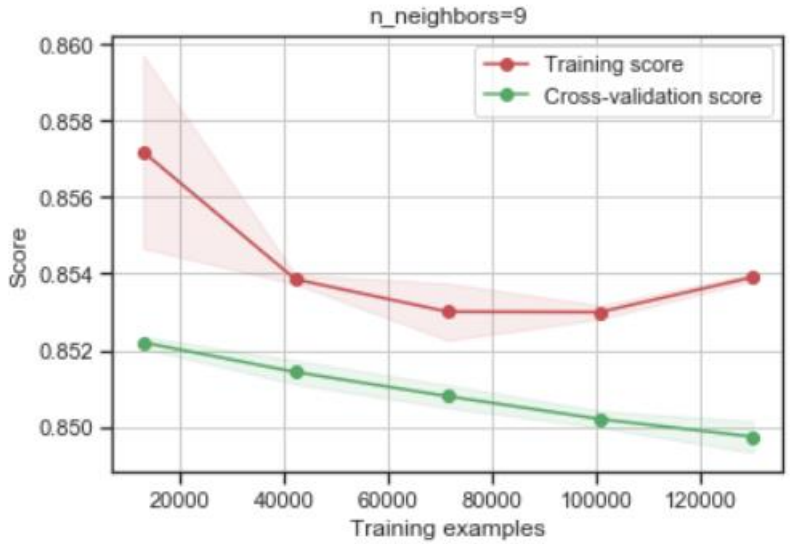
```
recall_score(y_test, target_1, average='micro') ,\
recall_score(y_test, target_2, average='micro')
```

Out[46]:

```
(0.7428697889291543, 0.8504522946692854)
```

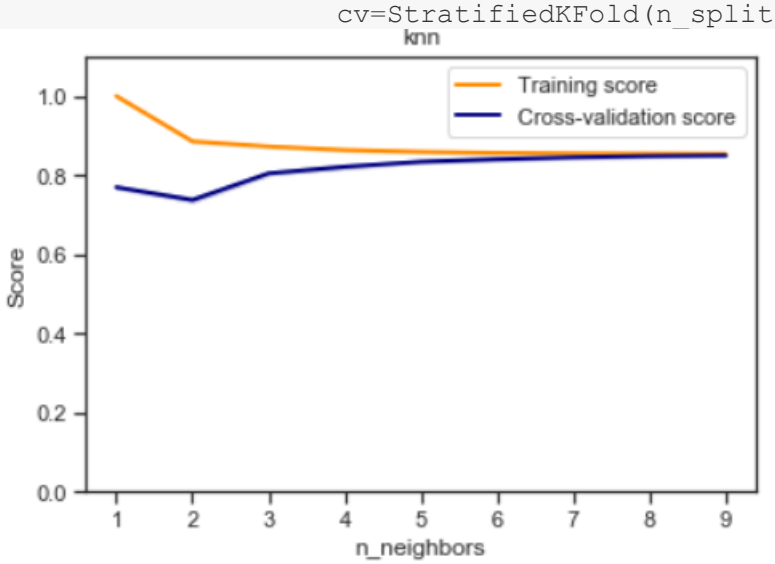
Кривые обучения:

```
plot_learning_curve(KNeighborsClassifier(n_neighbors=9), 'n_neighbors=9',
                    X_train, y_train, cv=StratifiedKFold(n_splits=5))
```



Кривые валидации:

```
plot_validation_curve(KNeighborsClassifier(), 'knn',
                      X_train, y_train,
                      param_name='n_neighbors', param_range=n_range,
                      cv=StratifiedKFold(n_splits=5), scoring="accuracy")
```



Функции для построения кривых обучения и валидации:

```
from sklearn.model_selection import learning_curve, validation_curve
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
```

4. Заключение (формулировка выводов по выполненной работе).

В ходе курсовой работы были закреплены полученные в течение курса знания и навыки.
Для исследования использовались следующие модели: стохастический градиентный спуск, дерево решений, случайный лес, градиентный бустинг, метод k-ближайших соседей. Для оценки качества использовались три метрики:

accuracy, precision и recall. Для наглядности были построены кривые обучения и валидации.
После подбора гиперпараметров модели показали следующие значения метрик:

- Стохастический градиентный спуск
  accuracy_score 0.8534768135586969
  precision_score 0.8534768135586969
  recall_score 0.8534768135586969
- Дерево решений
  accuracy_score 0.8536612354421975
  precision_score 0.8536612354421975
  recall_score 0.8536612354421975
- Метод ближайших соседей
  accuracy_score 0.8504522946692854
  precision_score 0.8504522946692854
  recall_score 0.8504522946692854
- Случайный лес
  accuracy_score 0.9914243824172176
  precision_score 0.9914243824172176
  recall_score 0.9914243824172176
- Градиентный бустинг
  accuracy_score 1.0

  precision_score 1.0
  recall_score 1.0

Наибольшую точность показал градиентный бустинг.

5. Список использованных источников информации (бумажных и электронных).

1. Конспект лекций по дисциплине “Технологии машинного обучения”. М.: МГТУ им. Н.Э.Баумана. 2019. (электронный ресурс)
2. Метрики в задачах машинного обучения: https://habr.com/ru/company/ods/blog/328372/ (электронный ресурс)
3. Ансамбли в машинном обучении https://dyakonov.org/2019/04/19/ансамбли-в-машинном-обучении/ (электронный ресурс)
4. Документация scikit-learn https://scikit-learn.org/stable/index.html (электронный ресурс)