

---

**Московский государственный технический университет им. Н.Э.Баумана**

---

Утверждаю:

Гапанюк Ю.Е.

"\_\_" 2019 г.

**Курсовая работа по курсу  
“Технологии машинного обучения”  
“Бинарная классификация”**

Вариант № 2

Пояснительная записка  
(вид документа)

писчая бумага  
(вид носителя)

х3  
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-63  
Желанкина А.С.

"\_\_" 2019 г.

---

Москва – 2019

---

## Содержание

1. Задание.....	3
2. Введение.....	4
3. Основная часть.....	5
а. Постановка задачи.....	5
б. Описание выбранного датасета (листинг).....	5
в. Решение задачи бинарной классификации (листинг).....	12
г. Графическая реализация (листинг).....	16
4. Заключение.....	21
5. Список литературы.....	22

## Задание

В данной курсовой работе необходимо предпринять следующие шаги:

1. 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.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. 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.

5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (*baseline*) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. 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.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством *baseline*-моделей.
11. 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 гиперпараметров на качество моделей и т.д.

Приведенная схема исследования является рекомендуемой. Возможно выполнение курсовой работы на нестандартную тему, которая должна быть предварительно согласована с ответственным за прием курсовой работы.

## **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.

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

## **Основная часть**

### ***Постановка задачи***

В данный момент времени всё больше людей заболевает раком. Существует множество методов диагностики, но большинство из них не дают точной информации о том, является ли опухоль доброкачественной или нет. Для того, чтобы определить её вид необходимо хирургическое вмешательство. Но даже проведение биопсии не всегда даёт точное решение. Для увеличения точности постановки диагноза и, соответственно, выбора наилучшего пути лечения можно применять различные методы машинного обучения.

Был взят датасет, содержащий информацию об опухолях молочных желёз. Необходимо предсказать, является ли опухоль доброкачественной или нет. Признаки, которые в нём описаны, были сформированы благодаря вычислениям из оцифрованного изображения аспирата тонкой иглы (FNA) массы груди. Они описывают характеристики ядер клеток, присутствующих на изображении. Разделительная плоскость, описанная выше, была получена с использованием Multisurface Method-Tree (MSM-T) [К. П. Беннетт, "Построение дерева решений с помощью линейного программирования". Труды 4-го Среднего Запада Общества искусственного интеллекта и когнитивной науки, стр. 97-101, 1992], метод классификации, который использует линейное программирование для построения дерева решений. Соответствующие элементы были выбраны с использованием исчерпывающего поиска в пространстве 1-4 элементов и 1-3 разделительных плоскостей. Фактическая линейная программа, используемая для получения разделяющей плоскости в трехмерном пространстве, описана в: [К. П. Беннетт и О. Л. Мангасарян: «Надежное распознавание линейного программирования двух линейно неразделимых множеств», Методы оптимизации и программное обеспечение 1, 1992, 23-34].

### ***Описание выбранного датасета***

Информация об атрибутах:

- 1) идентификационный номер 2) диагноз (M = злокачественный, B = доброкачественный)

Десять действительных признаков вычисляются для каждого ядра клетки:

- a) радиус (среднее расстояние от центра до точек по периметру) b) текстура (стандартное отклонение значений серой шкалы) c) периметр d) площадь e) гладкость (локальное изменение длины радиуса) f) компактность (периметр  $\wedge 2 / \text{площадь} - 1,0$ ) g) вогнутость (серьезность вогнутых частей контура) h) вогнутые точки (количество вогнутых частей контура) i) симметрия j) фрактальная размерность («приближение береговой линии» - 1)

Среднее значение, стандартная ошибка и «наихудшее» или наибольшее (среднее из трех самых больших значений) этих признаков были рассчитаны для каждого

изображения, в результате чего было получено 30 признаков. Например, поле 3 - средний радиус, поле 13 - радиус SE, поле 23 - наибольший радиус.

Импорт необходимых для работы библиотек

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5
6 from sklearn.impute import SimpleImputer
7
8 from sklearn.preprocessing import LabelEncoder
9 from sklearn.preprocessing import MinMaxScaler
10
11 from sklearn.model_selection import train_test_split
12 from sklearn.model_selection import cross_val_score
13 from sklearn.model_selection import LeaveOneOut
14 from sklearn.model_selection import GridSearchCV
15 from sklearn.neighbors import KNeighborsClassifier
16 from sklearn.svm import SVC
17 from sklearn.linear_model import LogisticRegression
18 from sklearn.ensemble import RandomForestClassifier
19 from sklearn.ensemble import GradientBoostingClassifier
20 from sklearn.naive_bayes import GaussianNB
21
22 from sklearn.metrics import accuracy_score
23 from sklearn.metrics import precision_score, recall_score
24
25 from sklearn.model_selection import learning_curve, validation_curve
26
27 %matplotlib inline
28 sns.set(style="ticks")
```

Загрузка данных "Breast Cancer" с kaggle. Этот набор данных содержит описание опухолей.

Целевая переменная «Диагноз» означает: считать ли эту опухоль за злокачественную? Да или нет.

```
In [2]: 1 data = pd.read_csv('wisconsin.csv')
2 data.head(5)
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003

5 rows × 33 columns

```
In [60]: 1 data.shape
```

Out[60]: (569, 33)

## Разведочный анализ данных, обработка датасета

```
In [4]: 1 data.describe()
```

Out[4]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
--	----	-------------	--------------	----------------	-----------	-----------------

<b>count</b>	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000
<b>mean</b>	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360
<b>std</b>	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064
<b>min</b>	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630
<b>25%</b>	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370
<b>50%</b>	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870
<b>75%</b>	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300
<b>max</b>	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400

8 rows × 32 columns

```
In [5]: 1 data.dtypes
```

```
out[5]: id                      int64
diagnosis                  object
radius_mean                 float64
texture_mean                float64
perimeter_mean              float64
area_mean                   float64
smoothness_mean             float64
compactness_mean            float64
concavity_mean              float64
concave points_mean         float64
symmetry_mean               float64
fractal_dimension_mean     float64
radius_se                    float64
texture_se                   float64
perimeter_se                float64
area_se                      float64
smoothness_se                float64
compactness_se               float64
concavity_se                 float64
concave points_se            float64
symmetry_se                  float64
fractal_dimension_se         float64
radius_worst                 float64
texture_worst                float64
perimeter_worst              float64
area_worst                   float64
smoothness_worst             float64
compactness_worst            float64
concavity_worst              float64
concave points_worst          float64
symmetry_worst                float64
fractal_dimension_worst      float64
```

```
Unnamed: 32          float64
dtype: object
```

Корреляция

```
In [6]: 1 data.corr()
```

```
out[6]:
```

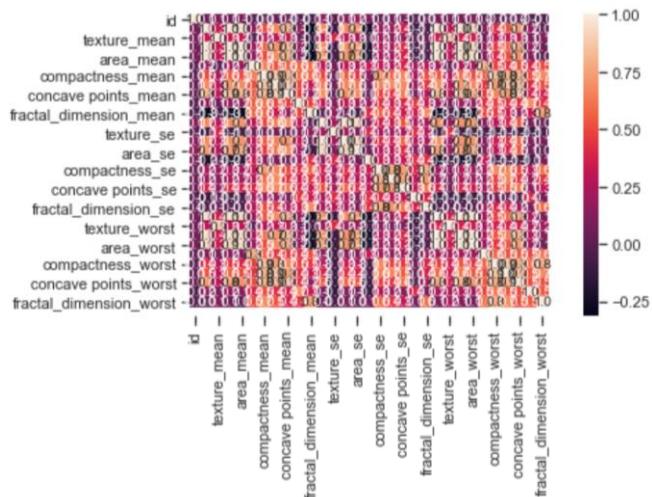
	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
<b>id</b>	1.000000	0.074626	0.099770	0.073159	0.096893	
<b>radius_mean</b>	0.074626	1.000000	0.323782	0.997855	0.987357	
<b>texture_mean</b>	0.099770	0.323782	1.000000	0.329533	0.321086	
<b>perimeter_mean</b>	0.073159	0.997855	0.329533	1.000000	0.986507	
<b>area_mean</b>	0.096893	0.987357	0.321086	0.986507	1.000000	
<b>smoothness_mean</b>	-0.012968	0.170581	-0.023389	0.207278	0.177028	
<b>compactness_mean</b>	0.000096	0.506124	0.236702	0.556936	0.498502	
<b>concavity_mean</b>	0.050080	0.676764	0.302418	0.716136	0.685983	
<b>concave points_mean</b>	0.044158	0.822529	0.293464	0.850977	0.823269	
<b>symmetry_mean</b>	-0.022114	0.147741	0.071401	0.183027	0.151293	
<b>fractal_dimension_mean</b>	-0.052511	-0.311631	-0.076437	-0.261477	-0.283110	
<b>radius_se</b>	0.143048	0.679090	0.275869	0.691765	0.732562	
<b>texture_se</b>	-0.007526	-0.097317	0.386358	-0.086761	-0.066280	
<b>perimeter_se</b>	0.137331	0.674172	0.281673	0.693135	0.726628	
<b>area_se</b>	0.177742	0.735864	0.259845	0.744983	0.800086	
<b>smoothness_se</b>	0.096781	-0.222600	0.006614	-0.202694	-0.166777	
<b>compactness_se</b>	0.033961	0.206000	0.191975	0.250744	0.212583	
<b>concavity_se</b>	0.055239	0.194204	0.143293	0.228082	0.207660	
<b>concave points_se</b>	0.078768	0.376169	0.163851	0.407217	0.372320	
<b>symmetry_se</b>	-0.017306	-0.104321	0.009127	-0.081629	-0.072497	
<b>fractal_dimension_se</b>	0.025725	-0.042641	0.054458	-0.005523	-0.019887	
<b>radius_worst</b>	0.082405	0.969539	0.352573	0.969476	0.962746	
<b>texture_worst</b>	0.064720	0.297008	0.912045	0.303038	0.287489	
<b>perimeter_worst</b>	0.079986	0.965137	0.358040	0.970387	0.959120	
<b>area_worst</b>	0.107187	0.941082	0.343546	0.941550	0.959213	
<b>smoothness_worst</b>	0.010338	0.119616	0.077503	0.150549	0.123523	
<b>compactness_worst</b>	-0.002968	0.413463	0.277830	0.455774	0.390410	
<b>concavity_worst</b>	0.023203	0.526911	0.301025	0.563879	0.512606	
<b>concave points_worst</b>	0.035174	0.744214	0.295316	0.771241	0.722017	
<b>symmetry_worst</b>	-0.044224	0.163953	0.105008	0.189115	0.143570	
<b>fractal_dimension_worst</b>	-0.029866	0.007066	0.119205	0.051019	0.003738	
Unnamed: 32	NaN	NaN	NaN	NaN	NaN	NaN

32 rows × 32 columns

Вывод значений в ячейках в тепловой карте

```
In [7]: 1 sns.heatmap(data.corr(), annot=True, fmt='.1f')
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x248832bdeb8>
```

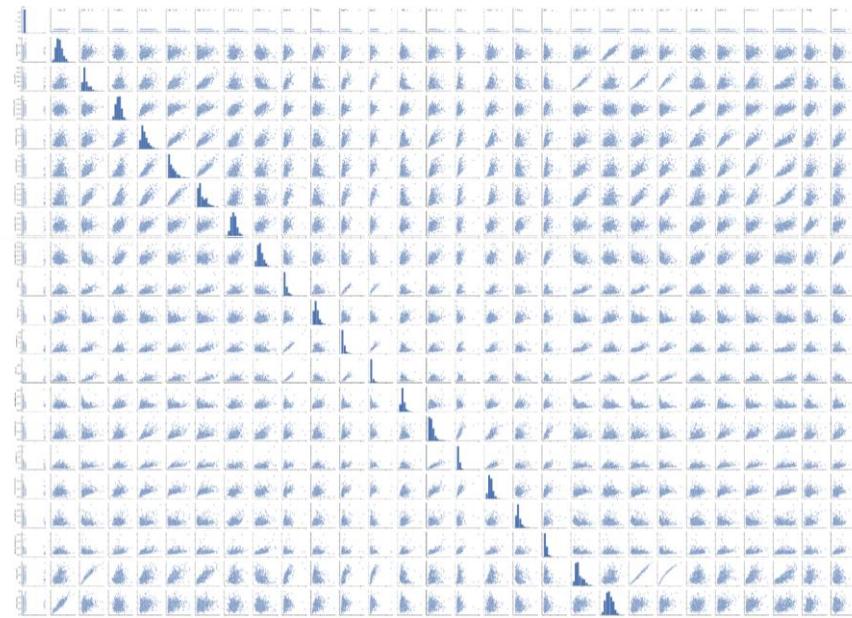


Удаление сильно коррелирующих между собой и мусорных данных

```
In [3]: 1 data = data.drop('Unnamed: 32', 1)
2 data = data.drop('radius_mean', 1)
3 data = data.drop('perimeter_mean', 1)
```

Парные диаграммы

```
In [9]: 1 sns.pairplot(data)
out[9]: <seaborn.axisgrid.PairGrid at 0x248832d47f0>
```



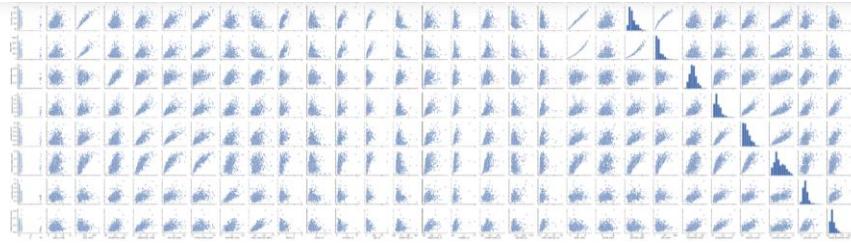
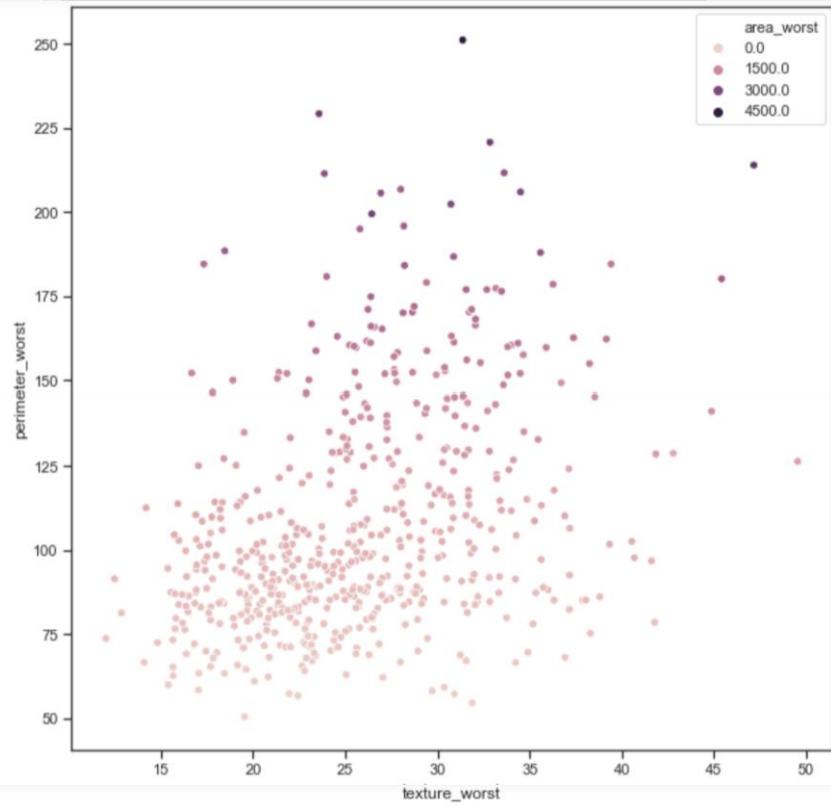


Диаграмма рассеяния площади

```
In [10]: 1 fig, ax = plt.subplots(figsize=(10,10))
2 sns.scatterplot(ax=ax, x='texture_worst', y='perimeter_worst', data=data, hu
```



Поиск пустых значений

```
In [9]: 1 def search_isnull():
2     total_count = data.shape[0]
3     num_cols = []
4     for col in data.columns:
5         # Количество пустых значений
6         temp_null_count = data[data[col].isnull()].shape[0]
7         dt = str(data[col].dtype)
8         if temp_null_count>0:
9             num_cols.append(col)
10            temp_perc = round((temp_null_count / total_count) * 100.0, 2)
11            print('Колонка {}. Тип данных {}. Количество пустых значений {},
```

```
In [10]: 1 search_isnull()
```

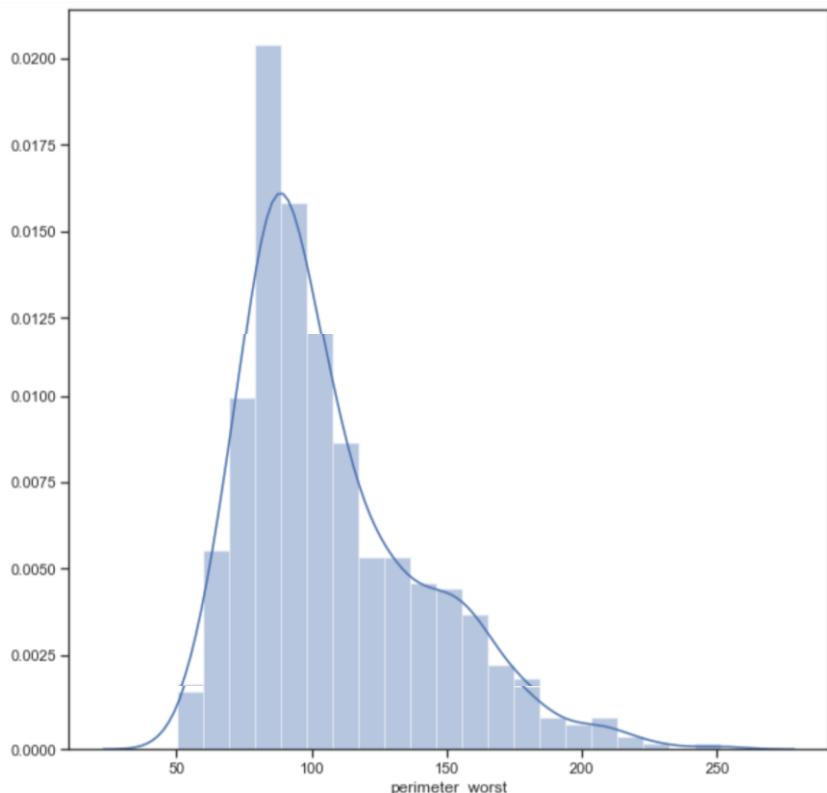
Гистограмма наихудшего периметра

```
In [13]: 1 fig, ax = plt.subplots(figsize=(10,10))
2 sns.distplot(data['perimeter_worst'])
```

C:\Anaconda\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple (seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.

```
    return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x248a290b8d0>
```



Кодирование категориальных признаков

```
In [4]: 1 label_enc = LabelEncoder()
2 obj_columns = ['diagnosis']
3
4 for obj in obj_columns:
5     data[obj] = label_enc.fit_transform(data[obj])
```

```
In [5]: 1 data.head(5)
```

```
Out[5]:
```

	id	diagnosis	texture_mean	area_mean	smoothness_mean	compactness_mean	concav
0	842302	1	10.38	1001.0	0.11840	0.27760	
1	842517	1	17.77	1326.0	0.08474	0.07864	
2	84300903	1	21.25	1203.0	0.10960	0.15990	
3	84348301	1	20.38	386.1	0.14250	0.28390	
4	84358402	1	14.34	1297.0	0.10030	0.13280	

5 rows × 30 columns

# Решение задачи бинарной классификации

## Выбор моделей для классификации и метрик качества

Разбиение выборки на тестовую и обучающую

```
In [6]: 1 target = data['diagnosis']
2 data = data.drop('diagnosis', axis = 1)
```

```
In [7]: 1 X_train, X_test, y_train, y_test = train_test_split(
2         data, target, test_size=0.3, random_state=1)
```

Масштабирование данных

```
In [8]: 1 min_max_sc = MinMaxScaler()
2
3 X_train = min_max_sc.fit_transform(X_train)
4 X_test = min_max_sc.transform(X_test)
```

```
C:\Anaconda\lib\site-packages\sklearn\preprocessing\data.py:323: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by
MinMaxScaler.
    return self.partial_fit(X, y)
```

Будут использоваться такие модели, как k-ближайших соседей, опорных векторов, логистическая регрессия, случайный лес, наивный Байес и градиентный бустинг

Обучение моделей

```
In [26]: 1 k_neighbors = KNeighborsClassifier(n_neighbors=300, algorithm='ball_tree', )
2 SVM = SVC(tol=500).fit(X_train, y_train)
3 logistic_regression = LogisticRegression(tol=25, max_iter=3).fit(X_train, y_
4 random_forest = RandomForestClassifier(n_estimators=3).fit(X_train, y_train)
5 naive_bayes = GaussianNB(var_smoothing=0.1).fit(X_train, y_train)
6 gradient_boosting = GradientBoostingClassifier(learning_rate=0.00001, n_esti
```

C:\Anaconda\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default value of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
"avoid this warning.", FutureWarning)
C:\Anaconda\lib\site-packages\sklearn\linear\_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)

Предсказание

```
In [27]: 1 target_k_neighbors = k_neighbors.predict(X_test)
2 target_SVM = SVM.predict(X_test)
3 target_logistic_regression = logistic_regression.predict(X_test)
4 target_random_forest = random_forest.predict(X_test)
5 target_naive_bayes = naive_bayes.predict(X_test)
6 target_gradient_boosting = gradient_boosting.predict(X_test)
```

Для оценки качества моделей будут использоваться метрики:

1. accuracy - доля верно предсказанных классификатором положительных и отрицательных объектов
2. precision - доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил

лил как положительные  
3. recall - доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов

Эти метрики позволяют оптимально понять, насколько хорошо происходит бинарная классификация

```
In [11]: 1 def print_accuracy(target):  
2     acc = accuracy_score(y_test, target)  
3     pr = precision_score(y_test, target)  
4     re = recall_score(y_test, target)  
5     print('accuracy = {}, precision = {}, recall = {}'.format(acc, pr, re))
```

Оценка качества k-ближайших соседей

```
In [102]: 1 print_accuracy(target_k_neighbors)  
  
accuracy = 0.631578947368421, precision = 0.0, recall = 0.0  
  
C:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1143: Undefined  
MetricWarning: Precision is ill-defined and being set to 0.0 due to no predicte  
d samples.  
    'precision', 'predicted', average, warn_for)
```

Оценка качества опорных векторов

```
In [103]: 1 print_accuracy(target_SVM)  
  
accuracy = 0.3684210526315789, precision = 0.3684210526315789, recall = 1.0
```

Оценка качества логистической регрессии

```
In [104]: 1 print_accuracy(target_logistic_regression)  
  
accuracy = 0.631578947368421, precision = 0.0, recall = 0.0  
  
C:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1143: Undefined  
MetricWarning: Precision is ill-defined and being set to 0.0 due to no predicte  
d samples.  
    'precision', 'predicted', average, warn_for)
```

Оценка качества случайного леса

```
In [105]: 1 print_accuracy(target_random_forest)  
  
accuracy = 0.935672514619883, precision = 0.9193548387096774, recall = 0.904761  
9047619048
```

Оценка качества градиентного бустинга

```
In [106]: 1 print_accuracy(target_gradient_boosting)  
  
accuracy = 0.631578947368421, precision = 0.0, recall = 0.0  
  
C:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1143: Undefined  
MetricWarning: Precision is ill-defined and being set to 0.0 due to no predicte  
d samples.  
    'precision', 'predicted', average, warn_for)
```

Оценка качества наивного Байеса

```
In [28]: 1 print_accuracy(target_naive_bayes)  
  
accuracy = 0.9181286549707602, precision = 0.9152542372881356, recall = 0.85714  
28571428571
```

Наиболее хорошие результаты показали модели k-ближайших соседей, опорных векторов и случайный лес, поэтому подбору гиперпараметров для них будет уделено больше времени

Подбор параметров логистической регрессии, наивного Байеса и градиентного бустинга с помощью решётчатого поиска

```
In [77]: 1 parameters_for_log_reg = {'tol':[0.00008,0.00009,0.0001,0.0002,0.0003,0.0004,
2                               'max_iter':np.array(range(5,100,5))} 
3 grs_log_reg = GridSearchCV(LogisticRegression(), parameters_for_log_reg, cv=5)
4 grs_log_reg.fit(X_train, y_train)
5 grs_log_reg.best_params_
6
7 warnings: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
8 FutureWarning)
C:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
9 FutureWarning)
C:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
10 FutureWarning)
C:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
11 FutureWarning)
C:\Anaconda\lib\site-packages\sklearn\svm\base.py:922: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
12   "the number of iterations.", ConvergenceWarning)
```

Out[77]: {'max\_iter': 5, 'tol': 8e-05}

```
In [29]: 1 parameters_for_nav_bay = {'var_smoothing':[0.0000001, 0.0000001, 0.000001,
2                               'grs_nav_bay = GridSearchCV(GaussianNB(), parameters_for_nav_bay, cv=2, scoring='accuracy')
3                               grs_nav_bay.fit(X_train, y_train)
4                               grs_nav_bay.best_params_
```

Out[29]: {'var\_smoothing': 1e-08}

```
In [78]: 1 parameters_for_grd_boost = {'learning_rate':[0.08, 0.09, 0.1, 0.2, 0.3],
2                               'n_estimators':np.array(range(3,75,3))}
3 grs_grd_boost = GridSearchCV(GradientBoostingClassifier(), parameters_for_grd_boost, cv=5)
4 grs_grd_boost.fit(X_train, y_train)
5 grs_grd_boost.best_params_
```

Out[78]: {'learning\_rate': 0.3, 'n\_estimators': 9}

Подбор параметров для остальных моделей с помощью решётчатого поиска и кросс-валидации

```
In [79]: 1 parameters_for_knn = {'n_neighbors':np.array(range(2,10,1)),
2                               'leaf_size':np.array(range(20,40,2))}
3 grs_knn = GridSearchCV(KNeighborsClassifier(algorithm='ball_tree'), parameters_for_knn, cv=5)
4 grs_knn.fit(X_train, y_train)
5 grs_knn.best_params_
```

Out[79]: {'leaf\_size': 20, 'n\_neighbors': 4}

```
In [80]: 1 parameters_for_svm = {'tol':[0.00001, 0.00005, 0.0001, 0.0005, 0.001]}
2 grs_svm = GridSearchCV(SVC(), parameters_for_svm, cv=LeaveOneOut(), scoring='accuracy')
3 grs_svm.fit(X_train, y_train)
4 grs_svm.best_params_
```

Out[80]: {'tol': 1e-05}

```
In [81]: 1 parameters_for_rand_for = {'n_estimators':np.array(range(2,15,1))}
2 grs_rand_for = GridSearchCV(RandomForestClassifier(), parameters_for_rand_for, cv=5)
3 grs_rand_for.fit(X_train, y_train)
4 grs_rand_for.best_params_
```

Out[81]: {'n\_estimators': 14}

Обучение моделей с подобранными гиперпараметрами и предсказание

```
In [30]: 1 new_k_neighbors = KNeighborsClassifier(n_neighbors=4, algorithm='ball_tree',
2 new_SVM = SVC(tol=0.00001).fit(X_train, y_train)
```

```
3 new_logistic_regression = LogisticRegression(tol=0.0005, max_iter=5).fit(X_t
4 new_random_forest = RandomForestClassifier(n_estimators=14).fit(X_train, y_t
5 new_naive_bayes = GaussianNB(var_smoothing=0.0000001).fit(X_train, y_train)
6 new_gradient_boosting = GradientBoostingClassifier(learning_rate=0.3, n_esti
```

```
In [31]: 1 new_target_k_neighbors = new_k_neighbors.predict(X_test)
2 new_target_SVM = new_SVM.predict(X_test)
3 new_target_logistic_regression = new_logistic_regression.predict(X_test)
4 new_target_random_forest = new_random_forest.predict(X_test)
5 new_target_naive_bayes = new_naive_bayes.predict(X_test)
6 new_target_gradient_boosting = new_gradient_boosting.predict(X_test)
```

Оценка качества моделей

Оценка качества k-ближайших соседей

```
In [85]: 1 print_accuracy(new_target_k_neighbors)
```

```
accuracy = 0.9590643274853801, precision = 0.9827586206896551, recall = 0.90476
19047619048
```

Оценка качества опорных векторов

```
In [86]: 1 print_accuracy(new_target_SVM)
```

```
accuracy = 0.9415204678362573, precision = 0.9818181818181818, recall = 0.85714
28571428571
```

Оценка качества логистической регрессии

```
In [87]: 1 print_accuracy(new_target_logistic_regression)
```

```
accuracy = 0.9473684210526315, precision = 0.9821428571428571, recall = 0.87301
5873015873
```

Оценка качества случайного леса

```
In [88]: 1 print_accuracy(new_target_random_forest)
```

```
accuracy = 0.9473684210526315, precision = 0.9354838709677419, recall = 0.92063
49206349206
```

Оценка качества градиентного бустинга

```
In [89]: 1 print_accuracy(new_target_gradient_boosting)
```

```
accuracy = 0.9532163742690059, precision = 0.9508196721311475, recall = 0.92063
49206349206
```

Оценка качества наивного Байеса

```
In [32]: 1 print_accuracy(new_target_naive_bayes)
```

```
accuracy = 0.935672514619883, precision = 0.90625, recall = 0.9206349206349206
```

```
In [33]: 1 print('Сравнение')
2 print('К-ближайших соседей')
3 print_accuracy(target_k_neighbors)
4 print_accuracy(new_target_k_neighbors)
5 print('Опорные векторы')
6 print_accuracy(target_SVM)
7 print_accuracy(new_target_SVM)
8 print('Логистическая регрессия')
9 print_accuracy(target_logistic_regression)
10 print_accuracy(new_target_logistic_regression)
11 print('Случайный лес')
12 print_accuracy(target_random_forest)
```

```

13 print_accuracy(new_target_random_forest)
14 print('Градиентный бустинг')
15 print_accuracy(target_gradient_boosting)
16 print_accuracy(new_target_gradient_boosting)
17 print('Наивный Байес')
18 print_accuracy(target_naive_bayes)
19 print_accuracy(new_target_naive_bayes)

Сравнение
К-ближайших соседей
accuracy = 0.631578947368421, precision = 0.0, recall = 0.0
accuracy = 0.9590643274853801, precision = 0.9827586206896551, recall = 0.90476
19047619048
Опорные вектора
accuracy = 0.3684210526315789, precision = 0.3684210526315789, recall = 1.0
accuracy = 0.9415204678362573, precision = 0.9818181818181818, recall = 0.85714
28571428571
Логистическая регрессия
accuracy = 0.631578947368421, precision = 0.0, recall = 0.0
accuracy = 0.9473684210526315, precision = 0.9821428571428571, recall = 0.87301
5873015873
Случайный лес
accuracy = 0.9239766081871345, precision = 0.9310344827586207, recall = 0.85714
28571428571
accuracy = 0.935672514619883, precision = 0.9193548387096774, recall = 0.904761
9047619048
Градиентный бустинг
accuracy = 0.631578947368421, precision = 0.0, recall = 0.0
accuracy = 0.9590643274853801, precision = 0.9666666666666667, recall = 0.92063
49206349206
Наивный Байес
accuracy = 0.9181286549707602, precision = 0.9152542372881356, recall = 0.85714
28571428571
accuracy = 0.935672514619883, precision = 0.90625, recall = 0.9206349206349206

```

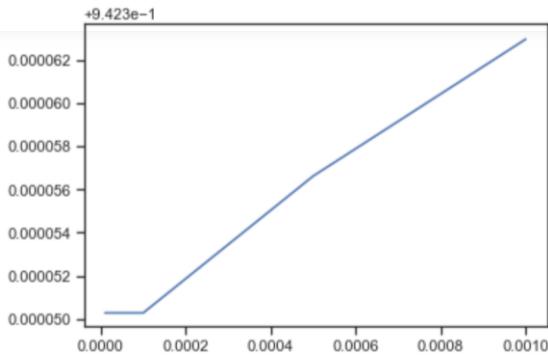
## *Графическая реализация*

### **Графики изменения качества на обучающей и тестовой выборках**

Опорные вектора

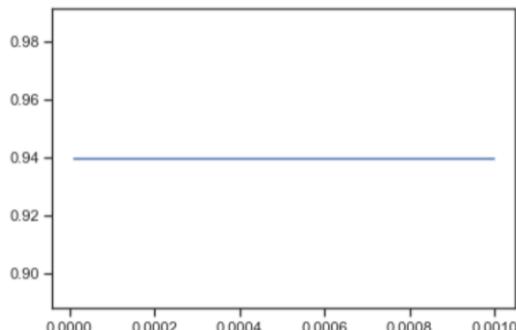
```
In [110]: 1 n_range = [0.00001, 0.00005, 0.0001, 0.0005, 0.001]
2 plt.plot(n_range, grs_svm.cv_results_['mean_train_score'])

Out[110]: [<matplotlib.lines.Line2D at 0x248ac624240>]
```



```
In [111]: 1 plt.plot(n_range, grs_svm.cv_results_['mean_test_score'])
```

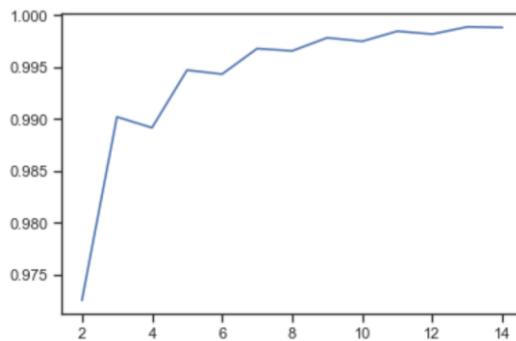
```
Out[111]: [<matplotlib.lines.Line2D at 0x2489aab2da0>]
```



Случайный лес

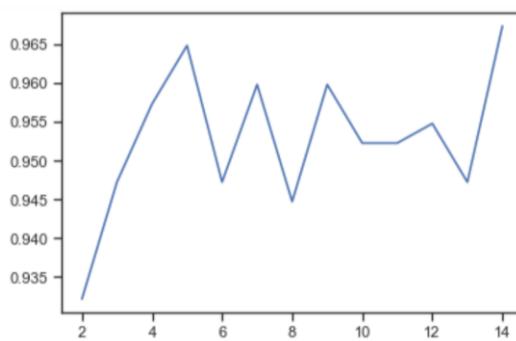
```
In [112]: 1 n_range = np.array(range(2,15,1))
2 plt.plot(n_range, grs_rand_for.cv_results_['mean_train_score'])
```

```
Out[112]: [<matplotlib.lines.Line2D at 0x248a28d6860>]
```



```
In [113]: 1 plt.plot(n_range, grs_rand_for.cv_results_['mean_test_score'])
```

```
Out[113]: [<matplotlib.lines.Line2D at 0x248abef2b00>]
```



## Построение кривых обучения

```
In [35]: 1 def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
2                           n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
3     """
4         Generate a simple plot of the test and training learning curve.
5
6     Parameters
7     -----
8         estimator : object type that implements the "fit" and "predict" methods
9                 An object of that type which is cloned for each validation.
10
11         title : string
12                 Title for the chart.
```

```

14     X : array-like, shape (n_samples, n_features)
15         Training vector, where n_samples is the number of samples and
16         n_features is the number of features.
17
18     y : array-like, shape (n_samples) or (n_samples, n_features), optional
19         Target relative to X for classification or regression;
20         None for unsupervised learning.
21
22     ylim : tuple, shape (ymin, ymax), optional
23         Defines minimum and maximum yvalues plotted.
24
25     cv : int, cross-validation generator or an iterable, optional
26         Determines the cross-validation splitting strategy.
27         Possible inputs for cv are:
28             - None, to use the default 3-fold cross-validation,
29             - integer, to specify the number of folds.
30             - :term:`CV splitter`,
31             - An iterable yielding (train, test) splits as arrays of indices.
32
33     For integer/None inputs, if ``y`` is binary or multiclass,
34     :class:`StratifiedKFold` used. If the estimator is not a classifier
35     or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.
36
37     Refer :ref:`User Guide <cross_validation>` for the various
38     cross-validation strategies that can be used here.
39
40     n_jobs : int or None, optional (default=None)
41         Number of jobs to run in parallel.
42         ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
43         ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
44         for more details.
45
46     train_sizes : array-like, shape (n_ticks,), dtype float or int
47         Relative or absolute numbers of training examples that will be used
48         to generate the learning curve. If the dtype is float, it is regarded as
49         a fraction of the maximum size of the training set (that is determined
50         by the selected validation method), i.e. it has to be within (0, 1].
51         Otherwise it is interpreted as absolute sizes of the training sets.
52         Note that for classification the number of samples usually have to
53         be big enough to contain at least one sample from each class.
54         (default: np.linspace(0.1, 1.0, 5))
55     """
56     plt.figure()
57     plt.title(title)
58     if ylim is not None:
59         plt.ylim(*ylim)
60     plt.xlabel("Training examples")
61     plt.ylabel("Score")
62     train_sizes, train_scores, test_scores = learning_curve(
63         estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
64     train_scores_mean = np.mean(train_scores, axis=1)
65     train_scores_std = np.std(train_scores, axis=1)
66     test_scores_mean = np.mean(test_scores, axis=1)
67     test_scores_std = np.std(test_scores, axis=1)
68     plt.grid()
69
70     plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
71                      train_scores_mean + train_scores_std, alpha=0.1,
72                      color="r")
73     plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
74                      test_scores_mean + test_scores_std, alpha=0.1, color="g")
75     plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
76              label="Training score")
77     plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
78              label="Cross-validation score")
79
80     plt.legend(loc="best")
81     return plt

```

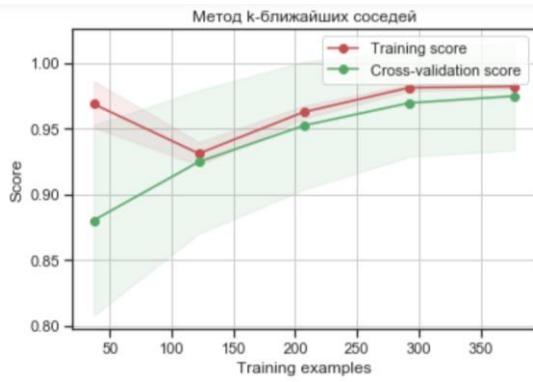
In [36]:

```

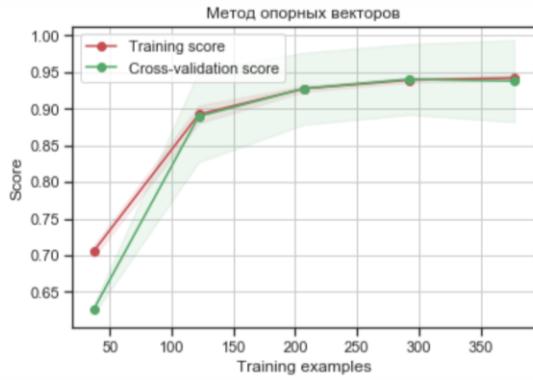
1 plot_learning_curve(KNeighborsClassifier(n_neighbors=4, algorithm='ball_tree',
2                         X_train, y_train, cv=20)

```

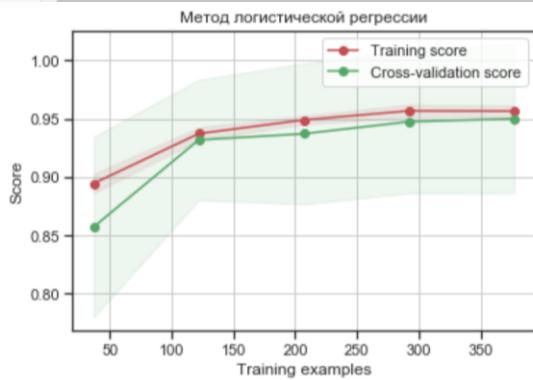
Out[36]: <module 'matplotlib.pyplot' from 'C:\\\\Anaconda\\\\lib\\\\site-packages\\\\matplotlib\\\\pyplot.py'>



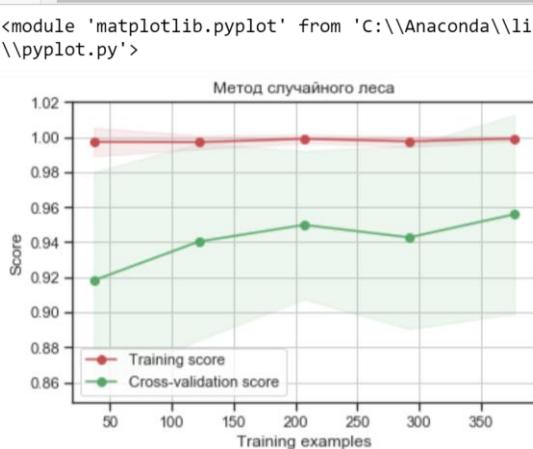
```
In [20]: 1 plot_learning_curve(SVC(tol=0.00001), 'Метод опорных векторов',  
2 X_train, y_train, cv=20)
```



```
In [21]: 1 plot_learning_curve(LogisticRegression(tol=0.0005, max_iter=5), 'Метод логистической регрессии',  
2 X_train, y_train, cv=20)
```

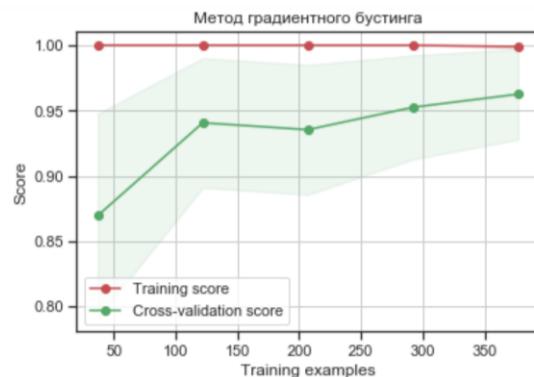


```
In [22]: 1 plot_learning_curve(RandomForestClassifier(n_estimators=14), 'Метод случайного леса',  
2 X_train, y_train, cv=20)
```



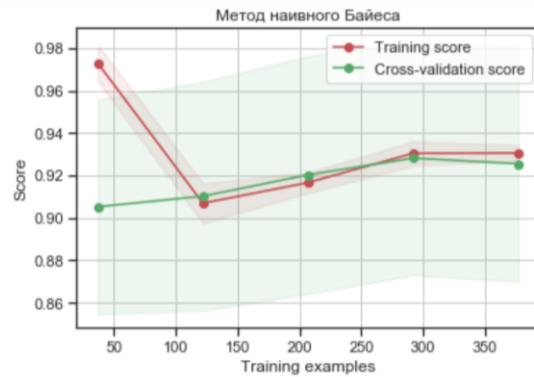
```
In [37]: 1 plot_learning_curve(GradientBoostingClassifier(learning_rate=0.3, n_estimators=100),  
2 X_train, y_train, cv=20)
```

```
Out[37]: <module 'matplotlib.pyplot' from 'C:\\\\Anaconda\\\\lib\\\\site-packages\\\\matplotlib\\\\pyplot.py'>
```



```
In [38]: 1 plot_learning_curve(GaussianNB(var_smoothing=0.0000001), 'Метод наивного Байеса',  
2 X_train, y_train, cv=20)
```

```
Out[38]: <module 'matplotlib.pyplot' from 'C:\\\\Anaconda\\\\lib\\\\site-packages\\\\matplotlib\\\\pyplot.py'>
```



## **Заключение**

Таким образом, внедрение технологий машинного обучения может помочь решении задач, которые возникают в всех сферах жизни человека, в том числе и медицине. Использование таких методов делает диагностику точнее, а лечение действеннее. На этом датасете самые хорошие результаты по всем метрикам качества показали методы К-ближайших соседей, наивный Байес и градиентный бустинг.

## **Список литературы**

1. Лекции Гапанюка Ю.Е. [Электронный ресурс]. – Электрон. дан. - URL: [https://github.com/ugapanyuk/ml\\_course/wiki/COURSE\\_TMO](https://github.com/ugapanyuk/ml_course/wiki/COURSE_TMO) (дата обращения: 10.05.2019)
2. Breast Cancer Wisconsin (Diagnostic) Data Set [Электронный ресурс]. – Электрон. дан. - URL: <https://www.kaggle.com/uciml/breast-cancer-wisconsin-data> (дата обращения: 10.05.2019)
3. Открытый курс машинного обучения. Тема 4. Линейные модели классификации и регрессии [Электронный ресурс]. – Электрон. дан. - URL: <https://habr.com/ru/company/ods/blog/323890/> (дата обращения: 10.05.2019)
4. Руководство для начинающих [Электронный ресурс]. – Электрон. дан. - URL: <https://mlbootcamp.ru/article/tutorial/> (дата обращения: 10.05.2019)