



Федеральное государственное образовательное бюджетное
учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ
РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)**

**Институт развития профессиональных
компетенций и квалификаций**

ИТОГОВАЯ РАБОТА

Группа обучения	«ИРПО_5_ИД_2023»
Срок обучения	«25.10.2023 г. - 14.12.2023 г.»
«Громов Виталий Каприянович»	
Номер Кейса	«.....»
Название Датасета	«Апельсины или грейпфруты»

Москва 2023 г.

Ссылка на файл в Colaboratory:

<https://colab.research.google.com/drive/1Wyx6VYCIXEY8XTDB2czGegeqWRy3MX26?usp=sharing>

Ссылка на Dataset:

<https://www.kaggle.com/datasets/joshmcadams/oranges-vs-grapefruit>

Описание задачи.

Перед нами стоит задача разделения апельсинов и грейпфрутов по характерным для них признакам, таких как цвет, вес и диаметр. То есть, мы должны решить задачу бинарной классификации. Найти лучшую из трёх моделей обучения, таких как метод логистической регрессии, метод решающих деревьев и метод опорных векторов. Будем действовать последовательно по шагам: загрузка и предобработка данных, обучение трёх упомянутых моделей с дефолтными параметрами, проверка на тестирующей выборке, оценка полученных результатов с помощью семи метрик, промежуточный вывод, затем поиск оптимальных гиперпараметров для получения наилучшей метрики ROC-AUC, обучение с учётом этих гиперпараметров, тестирование, оценка метрики ROC-AUC и вывод.

Решение.

Начнём с импорта необходимых библиотек и методов.

```
import pandas as pd
import numpy as np

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)


import matplotlib.pyplot as plt
%matplotlib inline

# models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# metrics
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
```

Загрузим файл citrus.csv и сохраним полученный DataFrame в переменную data.

```
data = pd.read_csv('/content/gdrive/MyDrive/Diploma/citrus.csv')
data
```



	<i>name</i>	<i>diameter</i>	<i>weight</i>	<i>red</i>	<i>green</i>	<i>blue</i>
0	orange	2.96	86.76	172	85	2
1	orange	3.91	88.05	166	78	3
2	orange	4.42	95.17	156	81	2
3	orange	4.47	95.60	163	81	4
4	orange	4.48	95.76	161	72	9
...
9995	grapefruit	15.35	253.89	149	77	20
9996	grapefruit	15.41	254.67	148	68	7
9997	grapefruit	15.59	256.50	168	82	20
9998	grapefruit	15.92	260.14	142	72	11
9999	grapefruit	16.45	261.51	152	74	2

10000 rows × 6 columns

Изучим загруженные данные.

```
data.shape
```

```
(10000, 6)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
#   Column   Non-Null Count  Dtype
---  ---
0   name     10000 non-null  object
1   diameter 10000 non-null  float64
2   weight   10000 non-null  float64
3   red      10000 non-null  int64
4   green    10000 non-null  int64
5   blue     10000 non-null  int64
dtypes: float64(2), int64(3), object(1)
memory usage: 468.9+ KB
```

Видим 10000 строк, как бы разных фруктов с подробным описанием каждого. А описывают их 6 характеристик (признаков) по названиям столбцов. Первая характеристика - название фрукта, тип object. Это категориальные данные. По условию задачи нам надо научиться отличать фрукты, поэтому эту категориальную характеристику мы позже определим как целевую переменную, естественно, преобразовав её к числовому типу. Остальные признаки - количественные данные. Нам также повезло, что нет пропусков.

Так как мы определили, что перед нами стоит задача бинарной классификации, то есть, фрукт либо апельсин, либо грейпфрут, то давайте посмотрим на сбалансированность нашей выборки.

```
data[data['name'] == 'orange'].shape
```

```
(5000, 6)
```



```
data[data['name'] == 'grapefruit'].shape
```

```
(5000, 6)
```

Да уж, слишком всё идеально. Выборка сбалансирована.
Взглянем на количественные данные.

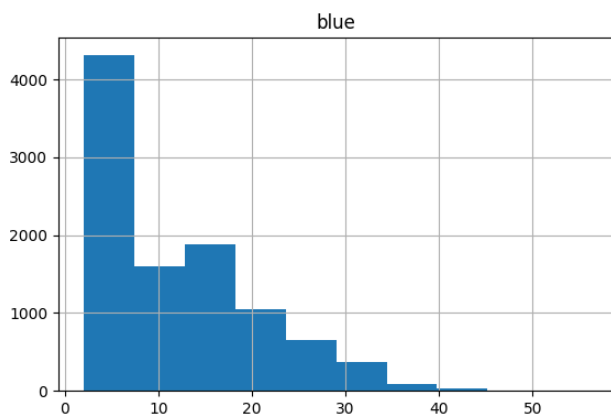
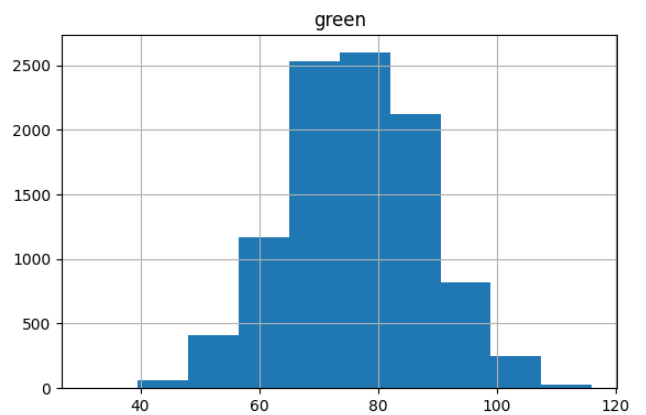
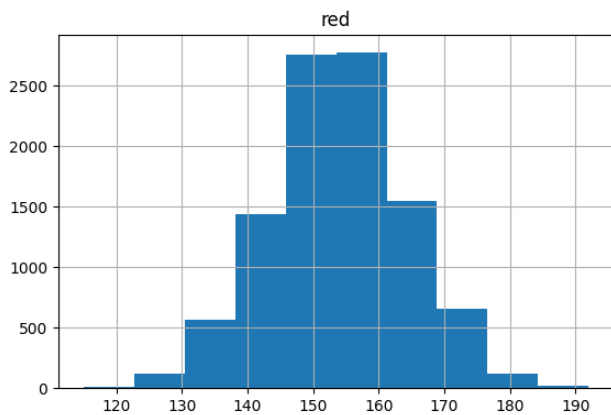
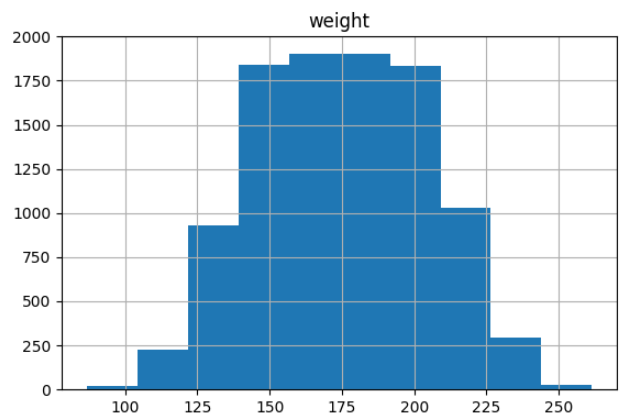
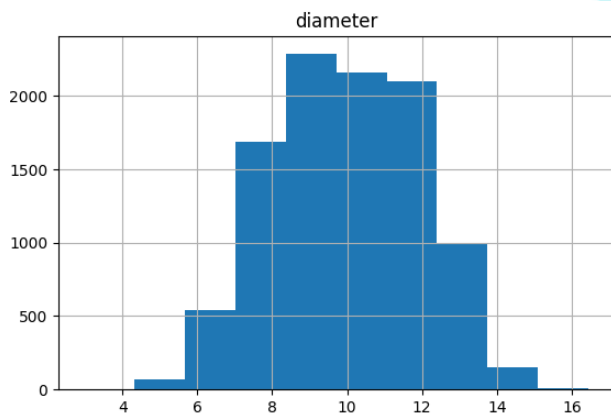
```
data.describe()
```

<i>diameter</i>	<i>weight</i>	<i>red</i>	<i>green</i>	<i>blue</i>	
<i>count</i>	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
<i>mean</i>	9.975685	175.050792	153.847800	76.010600	11.363200
<i>std</i>	1.947844	29.212119	10.432954	11.708433	9.061275
<i>min</i>	2.960000	86.760000	115.000000	31.000000	2.000000
<i>25%</i>	8.460000	152.220000	147.000000	68.000000	2.000000
<i>50%</i>	9.980000	174.985000	154.000000	76.000000	10.000000
<i>75%</i>	11.480000	197.722500	161.000000	84.000000	17.000000
<i>max</i>	16.450000	261.510000	192.000000	116.000000	56.000000

Здесь также нет резких всплесков. Убедимся в этом лишний раз на гистограммах.

```
data.hist(figsize=(15, 15))
```

```
array([[<Axes: title='{center': 'diameter'}>,
        <Axes: title='{center': 'weight'}>],
       [<Axes: title='{center': 'red'}>,
        <Axes: title='{center': 'green'}>],
       [<Axes: title='{center': 'blue'}>, <Axes: >]], dtype=object)
```



Видно, что синего цвета мало, ну да это и понятно.

Закодируем категориальную переменную 'name' в числовой вид с помощью функции LabelEncoder из sklearn.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['name'] = le.fit_transform(data['name'])

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
#   Column   Non-Null Count  Dtype
---  -
```

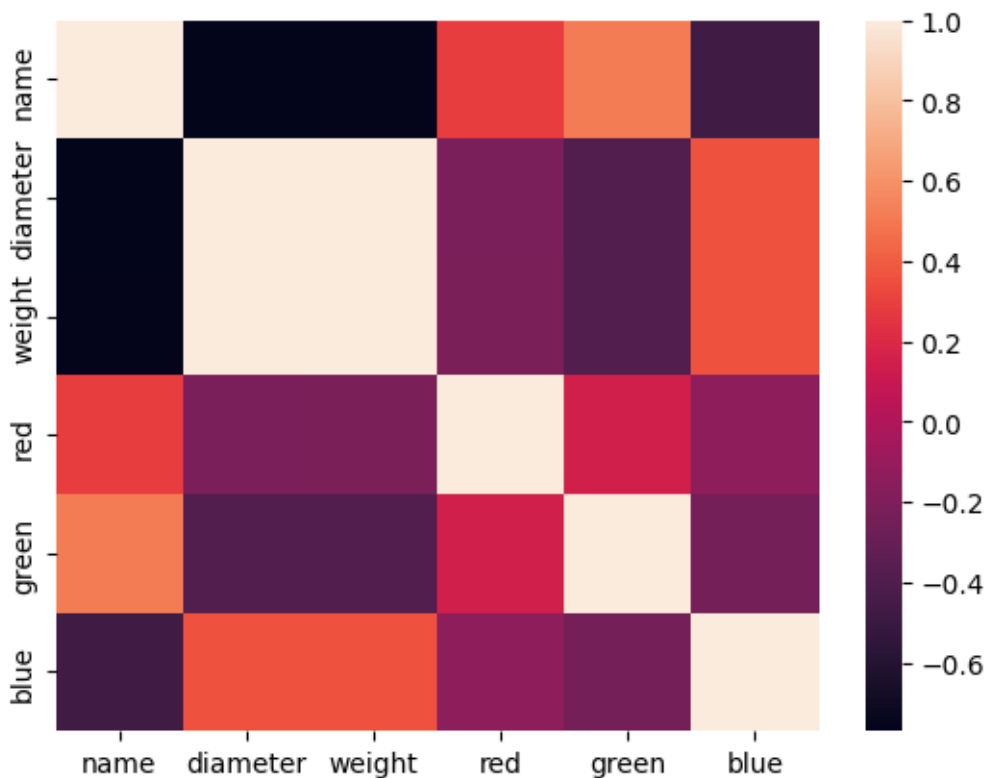


```
0 name 10000 non-null int64
1 diameter 10000 non-null float64
2 weight 10000 non-null float64
3 red 10000 non-null int64
4 green 10000 non-null int64
5 blue 10000 non-null int64
dtypes: float64(2), int64(4)
memory usage: 468.9 KB
```

Посмотрим, как коррелируют между собой все признаки, в том числе категориальный.

```
import seaborn as sns
sns.heatmap(data.corr())
```

<Axes: >



Видим, все так или иначе влияют друг на друга, значит ничего удалять не будем.

Выделяем целевую переменную, а это у нас категориальный признак name. И далее, с помощью функции `train_test_split` из `sklearn` делаем разбиение входной выборки данных на обучающую и тестовую.

```
training_values = data['name']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, target_train, target_test = train_test_split(data, training_values, test_size = 0.2, random_state = 17)
data_train = X_train.drop('name', axis=1)
data_train.head()
```



	<i>diameter</i>	<i>weight</i>	<i>red</i>	<i>green</i>	<i>blue</i>
994	7.43	137.08	155	96	2
76	5.67	112.10	150	85	6
8983	12.51	213.34	147	56	2
4027	9.55	169.30	165	61	2
4087	9.60	170.09	153	75	12

```
data_test = X_test.drop('name', axis=1)
data_test.head()
```

	<i>diameter</i>	<i>weight</i>	<i>red</i>	<i>green</i>	<i>blue</i>
2688	8.60	154.21	171	74	17
233	6.33	121.36	161	85	2
9099	12.61	214.88	151	71	13
8652	12.24	209.07	134	50	19
2842	8.70	155.86	156	86	19

Проводим нормализацию переменных в обучающей и тестовой выборках.

```
from sklearn import preprocessing
data_train_normal = preprocessing.normalize(data_train)
data_train_normal

array([[0.03255419, 0.60060951, 0.67912513, 0.42061944, 0.0087629 ],
       [0.0275491 , 0.54466554, 0.72881205, 0.41299349, 0.02915248],
       [0.04714232, 0.8039443 , 0.55395056, 0.21102878, 0.00753674],
       ...,
       [0.04647244, 0.7881187 , 0.54548441, 0.27628431, 0.0531316 ],
       [0.03179179, 0.599922 , 0.70121984, 0.38377573, 0.00947594],
       [0.0367318 , 0.66289837, 0.6328491 , 0.39829663, 0.00885104]])
```

```
data_test_normal = preprocessing.normalize(data_test)
data_test_normal

array([[0.03544752, 0.63562341, 0.7048285 , 0.3050135 , 0.07007067],
       [0.02891698, 0.55440199, 0.73548715, 0.38830067, 0.00913649],
       [0.04624809, 0.78808799, 0.55380345, 0.26039765, 0.04767844],
       ...,
       [0.04307173, 0.74241769, 0.60517091, 0.28390734, 0.01120687],
       [0.03107486, 0.58562441, 0.68606841, 0.4304743 , 0.00896821],
       [0.04644425, 0.79694282, 0.54457241, 0.24116778, 0.08946547]])
```

Создаём и обучаем наши модели с дефолтными гиперпараметрами.

```
logistic_regression_model = LogisticRegression(random_state = 17)
decision_tree_model = DecisionTreeClassifier(random_state = 17)
svm_model = SVC(random_state = 17)

logistic_regression_model.fit(data_train_normal, target_train)
```



```
decision_tree_model.fit(data_train_normal, target_train)
```

```
svm_model.fit(data_train_normal, target_train)
```

Тестируем обученные модели.

```
predict_logistic_regression = logistic_regression_model.predict(data_test_normal)
```

```
predict_decision_tree = decision_tree_model.predict(data_test_normal)
```

```
predict_svm = svm_model.predict(data_test_normal)
```

Посмотрим, что получилось. Для трёх моделей будем сравнивать метрики precision, recall, roc-auc, accuracy и f1_score.

```
ps_reg = precision_score(target_test, predict_logistic_regression)
```

```
ps_tree = precision_score(target_test, predict_decision_tree)
```

```
ps_svm = precision_score(target_test, predict_svm)
```

```
rs_reg = recall_score(target_test, predict_logistic_regression)
```

```
rs_tree = recall_score(target_test, predict_decision_tree)
```

```
rs_svm = recall_score(target_test, predict_svm)
```

```
ras_reg = roc_auc_score(target_test, predict_logistic_regression)
```

```
ras_tree = roc_auc_score(target_test, predict_decision_tree)
```

```
ras_svm = roc_auc_score(target_test, predict_svm)
```

```
as_reg = accuracy_score(target_test, predict_logistic_regression)
```

```
as_tree = accuracy_score(target_test, predict_decision_tree)
```

```
as_svm = accuracy_score(target_test, predict_svm)
```

```
f1s_reg = f1_score(target_test, predict_logistic_regression)
```

```
f1s_tree = f1_score(target_test, predict_decision_tree)
```

```
f1s_svm = f1_score(target_test, predict_svm)
```

```
print('\t\t\t\t\tlogistic_regression\t\t\t\t\tdecision_tree\t\t\t\t\tsvm')
```

```
print('precision_score\t\t\t', ps_reg, '\t\t\t', ps_tree, '\t\t\t', ps_svm)
```

```
print('recall_score\t\t\t', rs_reg, '\t\t\t', rs_tree, '\t\t\t', rs_svm)
```

```
print('roc_auc_score\t\t\t', ras_reg, '\t\t\t', ras_tree, '\t\t\t', ras_svm)
```

```
print('accuracy_score\t\t\t', as_reg, '\t\t\t\t\t', as_tree, '\t\t\t\t\t', as_svm)
```

```
print('f1_score\t\t\t\t\t', f1s_reg, '\t\t\t\t\t', f1s_tree, '\t\t\t\t\t', f1s_svm)
```

	<i>logistic_regression</i>	<i>decision_tree</i>	<i>svm</i>
<i>precision_score</i>	0.9284274193548387	0.936318407960199	0.92992992992993
<i>recall_score</i>	0.9056047197640118	0.9252704031465093	0.9134709931170109
<i>roc_auc_score</i>	0.9166884229542337	0.9300817936383615	0.921130206629716
<i>accuracy_score</i>	0.9165	0.93	0.921
<i>f1_score</i>	0.9168740666998507	0.930761622156281	0.9216269841269841

Победа за методом решающих деревьев.

Посмотрим также на матрицу ошибок.



```
cm_reg = confusion_matrix(target_test, predict_logistic_regression)
cm_tree = confusion_matrix(target_test, predict_decision_tree)
cm_svm = confusion_matrix(target_test, predict_svm)
```

```
print('confusion_matrix for logistic_regression:')
print(cm_reg)
```

```
confusion_matrix for logistic_regression:
[[912  71]
 [ 96 921]]
```

```
print('confusion_matrix for decision_tree:')
print(cm_tree)
```

```
confusion_matrix for decision_tree:
[[919  64]
 [ 76 941]]
```

```
print('confusion_matrix for svm:')
print(cm_svm)
```

```
confusion_matrix for svm:
[[913  70]
 [ 88 929]]
```

Опять же, главная диагональ лучше у метода решающих деревьев.
Напоследок глянем на pr_auc.

```
prc_reg = precision_recall_curve(target_test, predict_logistic_regression)
prc_tree = precision_recall_curve(target_test, predict_decision_tree)
prc_svm = precision_recall_curve(target_test, predict_svm)
```

```
print('precision_recall_curve')
print(prc_reg)
print(prc_tree)
print(prc_svm)
```

```
precision_recall_curve
(array([0.5085 , 0.92842742, 1.        ], array([1.        , 0.90560472, 0.        ], array([0, 1]))
(array([0.5085 , 0.93631841, 1.        ], array([1.        , 0.9252704 , 0.        ], array([0, 1]))
(array([0.5085 , 0.92992993, 1.        ], array([1.        , 0.91347099, 0.        ], array([0, 1]))
```

Здесь также precision и recall лучше у решающих деревьев.
Итак, что мы имеем.

После предобработки входных данных, мы, используя установки по умолчанию, создали три разные модели для обучения их решению задачи бинарной классификации. Обучили их по методам логистической регрессии, решающих деревьев и опорных векторов. Провели их тестирование на специально отделённой от входных данных выборке. Далее, мы произвели оценку полученных результатов, используя специальные метрики для задачи бинарной классификации. И пришли к следующему выводу. Для гиперпараметров, установленных по умолчанию, по итогам анализа метрик самый точный результат выдаёт метод решающих деревьев. Хотя, в принципе, все методы для этой конкретной выборки хороши.

Лучшей моделью для данной выборки является модель решающих деревьев.



Всё это хорошо, когда знаешь, какую или какие модели использовать, с какими гиперпараметрами. А если нет?

Есть в библиотеке `sklearn.model_selection` функция `GridSearchCV` для перебора моделей с различными настройками (своими для каждой) и выбором лучшей комбинации модель-гиперпараметры. Попробуем эту функцию подбора гиперпараметров для получения наилучшей метрики ROC_AUC с уже знакомыми нам тремя моделями, описанными выше.

Подгружаем эту функцию из `sklearn`. Для логистической регрессии выбираем гиперпараметры: `C` - параметр регуляризации, `penalty` - штраф, `solver` - решатель. Запускаем обучение, используя `GridSearchCV` с нашими гиперпараметрами, выводим результат.

```
from sklearn.model_selection import GridSearchCV
```

```
regression_default = LogisticRegression(random_state = 17)
```

```
param_grid = [  
    {'C': [0.1, 1, 10, 100]},  
    {'penalty': ['none', 'l2']},  
    {'solver': ['lbfgs', 'newton-cg', 'liblinear', 'sag', 'saga']},  
]
```

```
regression_search = GridSearchCV(estimator=regression_default, param_grid=param_grid, scoring="roc_auc",  
cv=3)
```

```
regression_search.fit(data_train_normal, target_train)
```

```
print('Parameters search: ', regression_search.best_params_)
```

```
print('Roc_auc search: ', regression_search.best_score_)
```

```
Parameters search: {'penalty': 'none'}
```

```
Roc_auc search: 0.9867232594564322
```

Но это не совсем то. Давайте пройдем классическим путём обучения. Предобработанные выборки для обучения и тестирования у нас уже готовы. Поэтому сразу приступаем к обучению модели. Но. Если выше мы брали дефолтные значения, то теперь строим модель с учётом рекомендованных функцией `GridSearchCV` гиперпараметров. Обучаем, проверяем на тестирующей выборке, оцениваем ROC_AUC.

```
regression_giper = LogisticRegression(penalty = None, random_state = 17)
```

```
regression_giper.fit(data_train_normal, target_train)
```

```
predict_regression_giper = regression_giper.predict(data_test_normal)
```

```
roc_auc_reg = roc_auc_score(target_test, predict_regression_giper)
```

```
print('Roc_auc giper: ', roc_auc_reg)
```

```
Roc_auc giper: 0.9448835713521208
```

Для метода решающих деревьев выбираем следующие гиперпараметры: `max_depth` - максимальная глубина дерева, `min_samples_split` - минимальное количество экземпляров, которое может содержаться в узле, `criterion` - функция для измерения качества разбиения. Далее делаем всё аналогично сделанному выше для метода логистической регрессии.



```
tree_default = DecisionTreeClassifier(random_state = 17)
```

```
param_grid = [
    {'max_depth': [5, 10, 25, None]},
    {'min_samples_split': [2, 5, 10]},
    {'criterion': ['gini', 'entropy']},
]
```

```
tree_search = GridSearchCV(estimator=tree_default, param_grid=param_grid, scoring="roc_auc", cv=3)
tree_search.fit(data_train_normal, target_train)
print('Parametres search: ', tree_search.best_params_)
print('Roc_auc search: ', tree_search.best_score_)
```

Parametres search: {'max_depth': 5}
Roc_auc search: 0.9750211123283785

```
tree_giper = DecisionTreeClassifier(max_depth = 5, random_state = 17)
tree_giper.fit(data_train_normal, target_train)
predict_tree_giper = tree_giper.predict(data_test_normal)
roc_auc_reg = roc_auc_score(target_test, predict_tree_giper)
print('Roc_auc giper: ', roc_auc_reg)
```

Roc_auc giper: 0.9252163875359979

Для метода опорных векторов предлагаем использовать такие гиперпараметры: C - параметр регуляризации, kernel - ядро. Дальше аналогично предыдущему.

```
svm_default = SVC(random_state = 17)
```

```
param_grid = [
    {'C': [0.1, 1, 10, 100]},
    {'kernel': ['linear']},
    {'kernel': ['poly'], 'degree': [2, 3]},
    {'kernel': ['rbf']},
]
```

```
svm_search = GridSearchCV(estimator=svm_default, param_grid=param_grid, scoring="roc_auc", cv=3)
svm_search.fit(data_train_normal, target_train)
print("Best parametres: ", svm_search.best_params_)
print("Best roc_auc: ", svm_search.best_score_)
```

Best parametres: {'C': 100}
Best roc_auc: 0.9804913050869546

```
svm_giper = SVC(C = 100, random_state = 17)
svm_giper.fit(data_train_normal, target_train)
predict_svm_giper = svm_giper.predict(data_test_normal)
roc_auc_reg = roc_auc_score(target_test, predict_svm_giper)
print('Roc_auc giper: ', roc_auc_reg)
```

Roc_auc giper: 0.9261656618762822



Вывод.

Давайте подытожим описанное выше. Мы решили попробовать научить компьютер отличать апельсины от грейпфрутов. То есть, классифицировать принадлежность определённого фрукта к заданному классу. Подсовываем компьютеру фрукт и спрашиваем: это апельсин, ДА или НЕТ? Если ДА, то этот фрукт – апельсин, если НЕТ, то этот фрукт – грейпфрут, так как у нас во входных данных только апельсины и грейпфруты, других фруктов нет. То есть, мы понимаем, что перед нами задача бинарной классификации. Зная какими моделями решается эта задача, мы выбираем три из них: модель логистической регрессии, модель решающих деревьев и модель опорных векторов. Нам надо выбрать лучшую. На вход нам подаётся набор данных, включающий в себя цвет (разложенный на составляющие – красный, зелёный, синий), вес и диаметр апельсина и грейпфрута. После предобработки входных данных проводим исследования с дефолтными гиперпараметрами для каждой модели. Выясняется преимущество модели решающих деревьев.

Идём дальше. С помощью функции `GridSearchCV` из `sklearn.model_selection` проводим отбор оптимальных гиперпараметров для получения наилучшей метрики `ROC_AUC`, своих для каждой нашей модели. Проводим вторичный эксперимент над нашими данными, но уже на моделях с полученными оптимальными наборами гиперпараметров. Оцениваем результаты для получения наилучшей метрики `ROC-AUC` для трёх моделей. Картина изменилась, теперь лучшей по отношению к нашим двум другим, является модель логистической регрессии.

Отсюда видно какую важную роль, наряду с выбором модели обучения, играет выбор и настройка гиперпараметров для неё.