Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Решение задачи машинного обучения

Студент группы ИУ5-64Б	_____	Харчевников А.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.

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/aungpyaeap/fish-market

Он содержит набор данных о параметрах разных видов рыб. Файл fish.csv содержит 156 строк и 7 стобцов. Для целей курсовой работы датасет был расширен до 646 строк.

В него включены следующие столбцы:

1. Species - вид/название рыбы
2. Weight - вес рыбы в граммах
3. Length1 - вертикальная длина в см
4. Length2 - диагональная длина в см
5. Length3 - перекрестная длина в см
6. Height - толщина рыбы
7. Width - ширина рыбы

Будем решать задачу классификации – определение принадлежности рыбы с определенными характеристиками к какому-либо виду.

3.2 Ход работы

Импортируем необходимые для работы библиотеки:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split, RandomizedSearchCV, learni
ng_curve, validation_curve
```

```
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classificat
ion_report, mean_absolute_error
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_square
d_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSV
R
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_g
raphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegress
or
from sklearn import svm
%matplotlib inline
sns.set(style="ticks")
```

Загружаем данные:

In [2]:

```
fish = pd.read_csv('fish2.csv', sep=",")
```

3.3Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

In [3]:

```
fish.head()
```

Out[3]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

Оценим размер базы данных:

In [4]:

```
fish.shape
```

Out[4]:

```
(646, 7)
```

Датасет достаточно маленький, из-за чего может страдать точность работы алгоритма классификации.

In [5]:

```
fish.columns
```

Out[5]:

```
Index(['Species', 'Weight', 'Length1', 'Length2', 'Length3', 'Height',
       'Width'],
      dtype='object')
```

Посмотрим, какие типы данных присутствуют в датасете:

In [6]:

```
fish.dtypes
```

Out[6]:

```
Species     object
Weight     float64
Length1    float64
Length2    float64
Length3    float64
Height     float64
Width      float64
dtype: object
```

Проверим датасет на наличие пустых значений:

In [7]:

```
fish.isnull().sum()
```

Out[7]:

```
Species    0
Weight     0
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

Пустые значения отсутствуют. Заполнение не требуется. Так как обучающая и тестовая выборк будут представлены в виде частей данного датасета, там так же будут отсутствовать пропуски в данных.

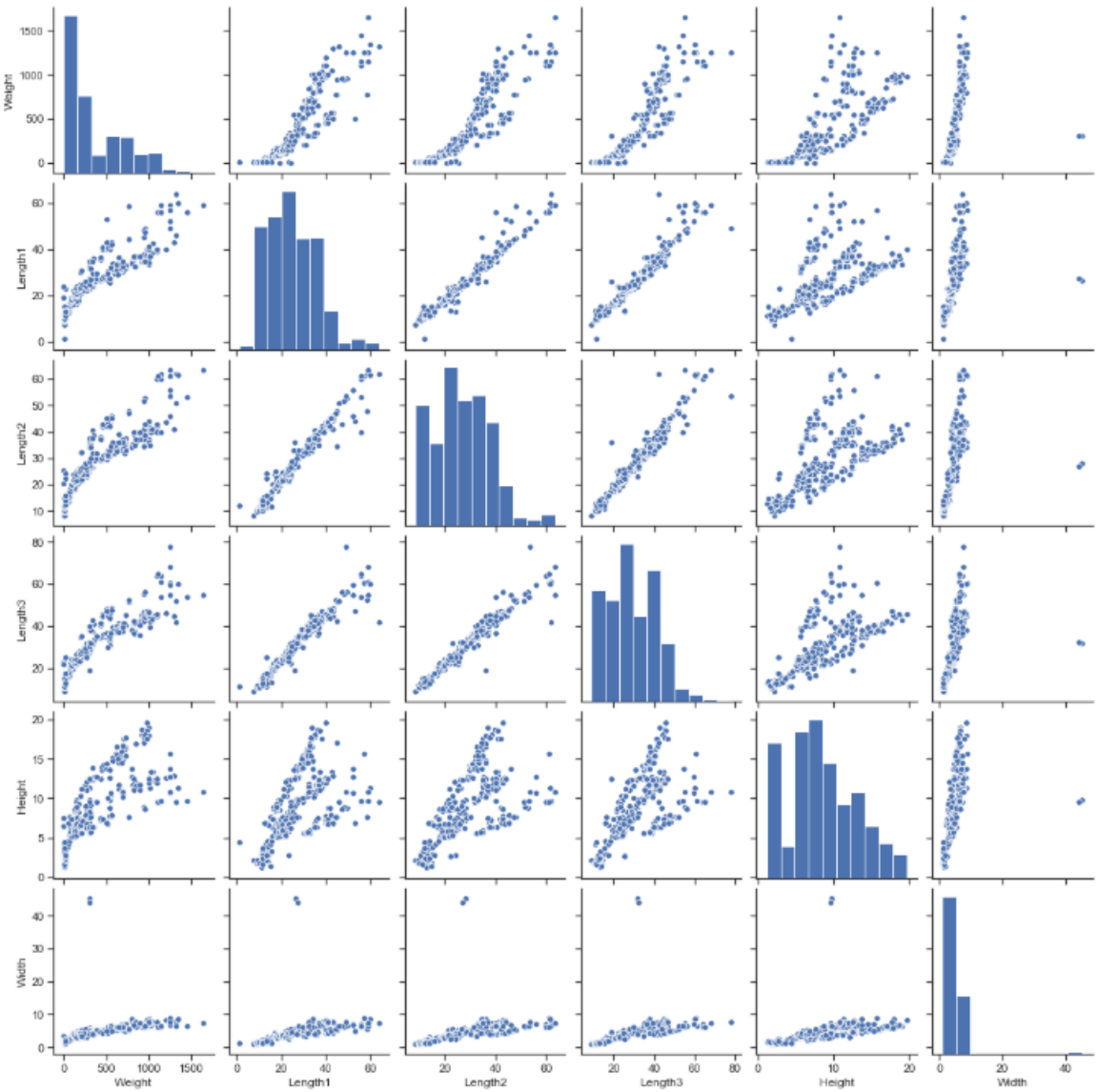
Оценим структуру представленных данных:

In [8]:

```
sns.pairplot(fish)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0xdf47750>
```

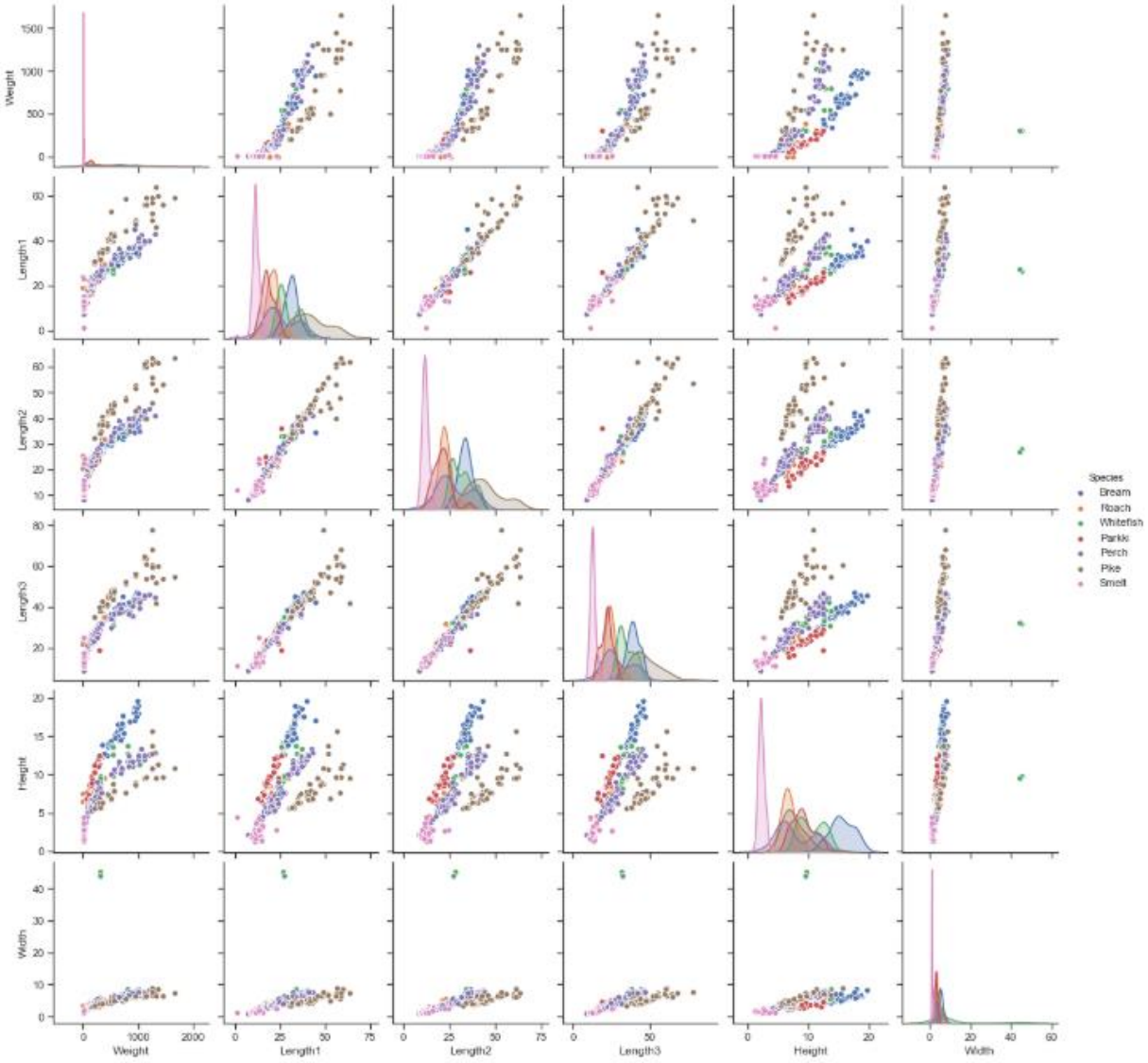


In [10]:

```
sns.pairplot(fish, hue = "Species")
```

Out[10]:

```
<seaborn.axisgrid.PairGrid at 0xa739
```



Уточним количество уникальных значений целевого признака:

In [11]:

```
fish['Species'].unique()
```

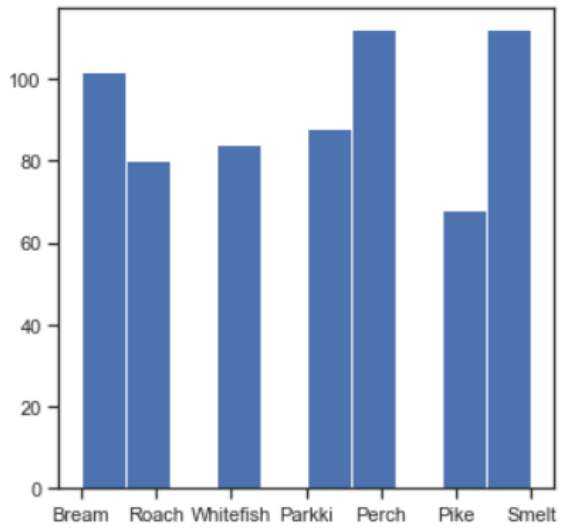
Out[11]:

```
array(['Bream', 'Roach', 'Whitefish', 'Parkki', 'Perch', 'Pike', 'Smelt'],
      dtype=object)
```

Имеем 7 видов рыб.

In [12]:

```
# Оценим дисбаланс классов для Species
fig, ax = plt.subplots(figsize=(5,5))
plt.hist(fish['Species'])
plt.show()
```



In [13]:

```
fish['Species'].value_counts()
```

Out[13]:

```
Perch        112
Smelt        112
Bream        102
Parkki        88
Whitefish     84
Roach         80
Pike          68
Name: Species, dtype: int64
```

In [14]:

```
#посчитаем дисбаланс классов
total = fish.shape[0]
class_0, class_1, class_2, class_3, class_4, class_5, class_6 = fish['Species
'].value_counts()
print('Класс 0 составляет {}%, \nкласс 1 составляет {}%, \nкласс 2 составляет
{}%, \nкласс 3 составляет {}%, \nКласс 4 составляет {}%, \nкласс 5 составляет
{}%, \nкласс 6 составляет {}%, '
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100, r
ound(class_2 / total, 4)*100, round(class_3 / total, 4)*100,  round(class_4 /
total, 4)*100,  round(class_5 / total, 4)*100,
               round(class_6 / total, 4)*100,))
```

```
Класс 0 составляет 17.34%,
класс 1 составляет 17.34%,
класс 2 составляет 15.790000000000001%,
класс 3 составляет 13.62%,
Класс 4 составляет 13.0%,
класс 5 составляет 12.379999999999999%,
класс 6 составляет 10.530000000000001%,
```

Дисбаланс классов присутствует, но является приемлемым.

3.4 Выбор признаков, подходящих для построения моделей. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

In [16]:
```
fish.dtypes
```

Out[16]:
```
Species     object
Weight     float64
Length1    float64
Length2    float64
Length3    float64
Height     float64
Width      float64
dtype: object
```

Для построения моделей будем использовать все признаки.

Категориальные признаки отсутствуют, их кодирования не требуется.

Вспомогательные признаки для улучшения качества моделей в данном примере мы строить не будем.

Выполним масштабирование данных.

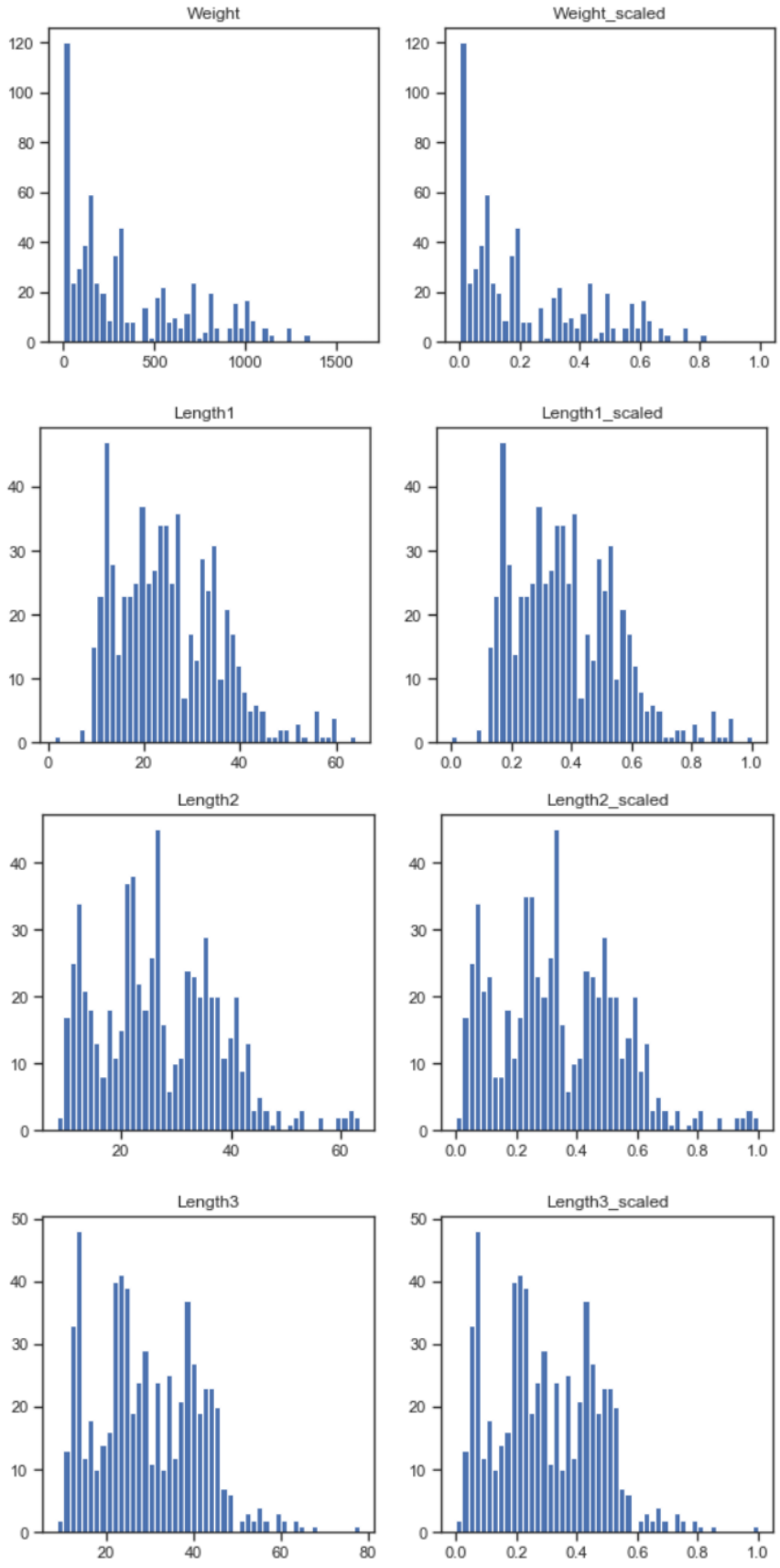
In [17]:
```
# Числовые колонки для масштабирования
scale_cols = ['Weight', 'Length1', 'Length2', 'Length3', 'Height',
       'Width']
```

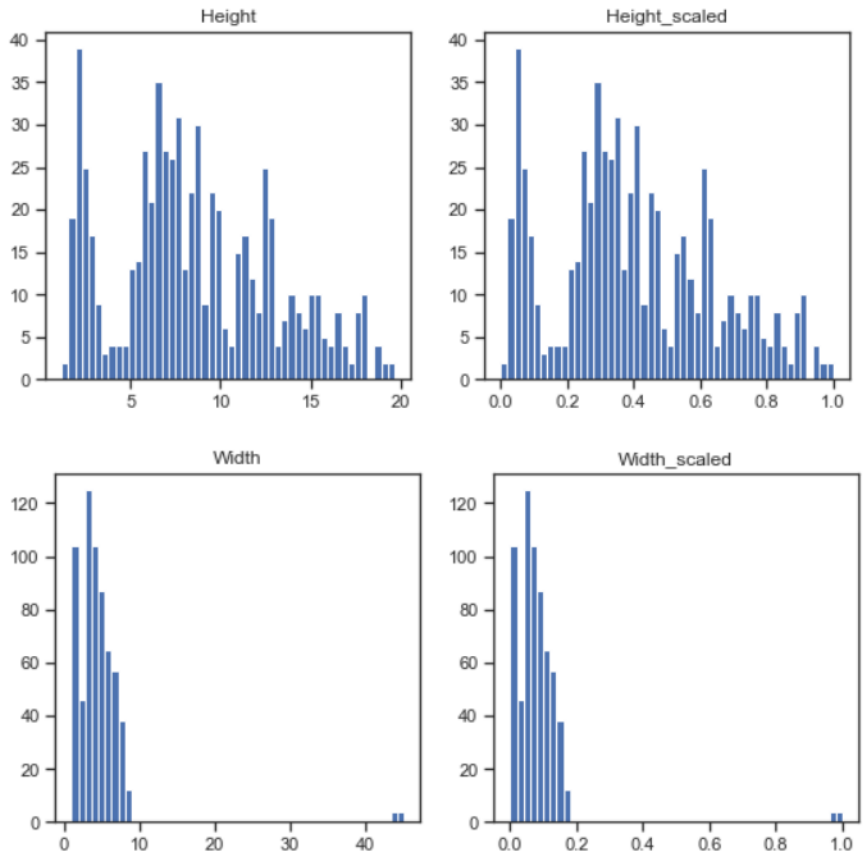
In [18]:
```
# Создадим дубликат базы, на случай, если что-то пойдет не так
fish2 = fish
```

In [19]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(fish2[scale_cols])
```

In [20]:
```
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    fish2[new_col_name] = sc1_data[:,i]
```

In [21]:
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(9,4))
    ax[0].hist(fish2[col], 50)
    ax[1].hist(fish2[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```


Weight
Weight_scaled
Length1
Length1_scaled
Length2
Length2_scaled
Length3
Length3_scaled



3.5 Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

In [22]:

```
corr_cols_1 = scale_cols + ['Species']
corr_cols_1
```

Out[22]:

```
['Weight', 'Length1', 'Length2', 'Length3', 'Height', 'Width', 'Species']
```

In [23]:

```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['Species']
corr_cols_2
```

Out[23]:

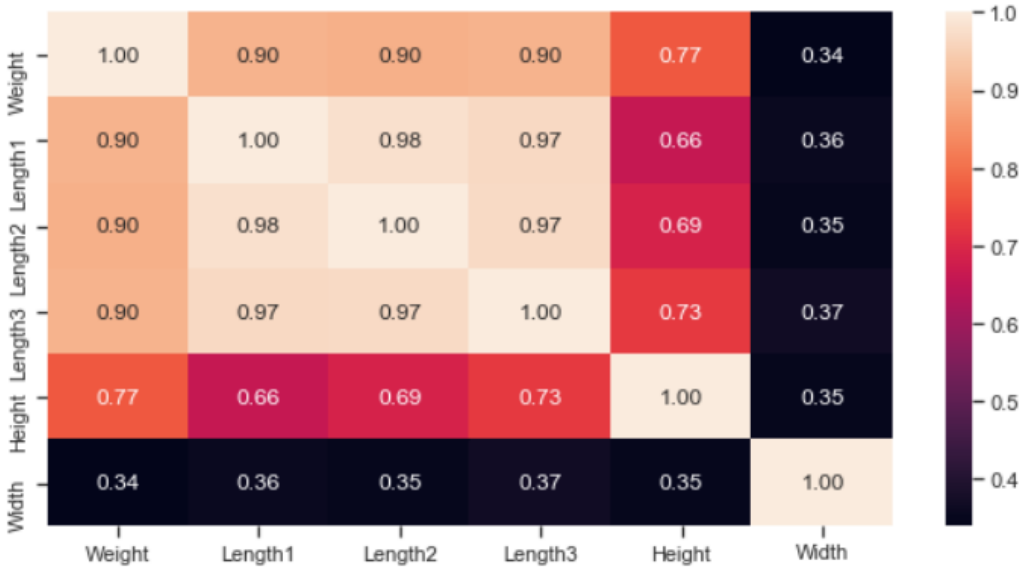
```
['Weight_scaled',
 'Length1_scaled',
 'Length2_scaled',
 'Length3_scaled',
 'Height_scaled',
 'Width_scaled',
 'Species']
```

In [24]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(fish[corr_cols_1].corr(), annot=True, fmt='.2f')
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x12101cb0>
```

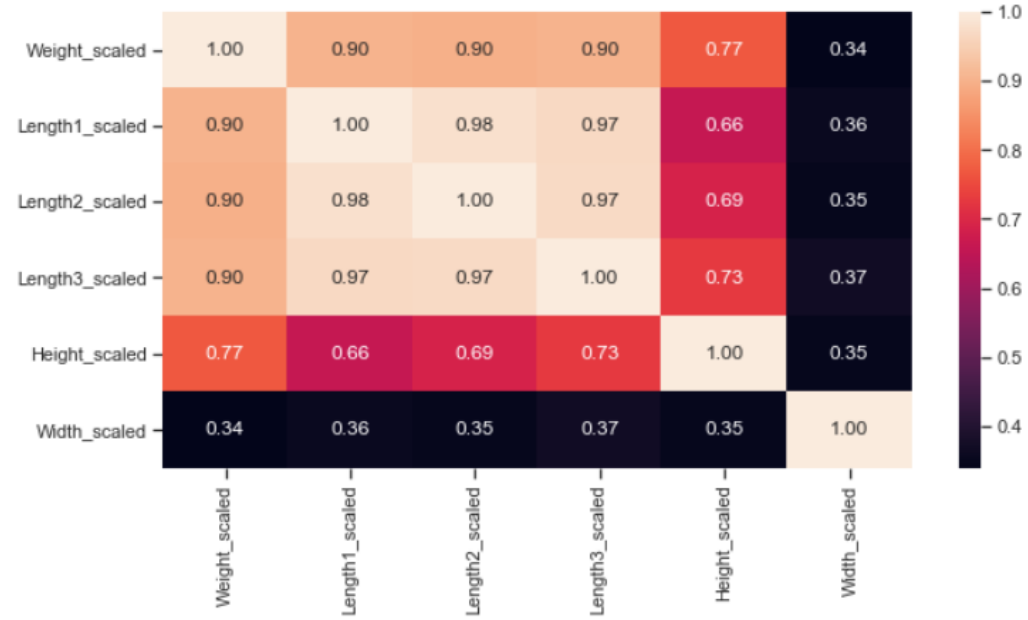


In [25]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(fish2[corr_cols_2].corr(), annot=True, fmt='.2f')
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1059d870>
```



На основе корреляционной матрицы можно сделать следующие выводы:

1. Корреляционные матрицы для исходных и масштабируемых данных совпадают
2. Наиболее сильно коррелируют значения: Length1 и Length2, Length1 и Length3

3.6 Выбор метрик для последующей оценки качества моделей

1. Метрика accuracy
2. Метрика confusion matrix

3. Метрика classification report

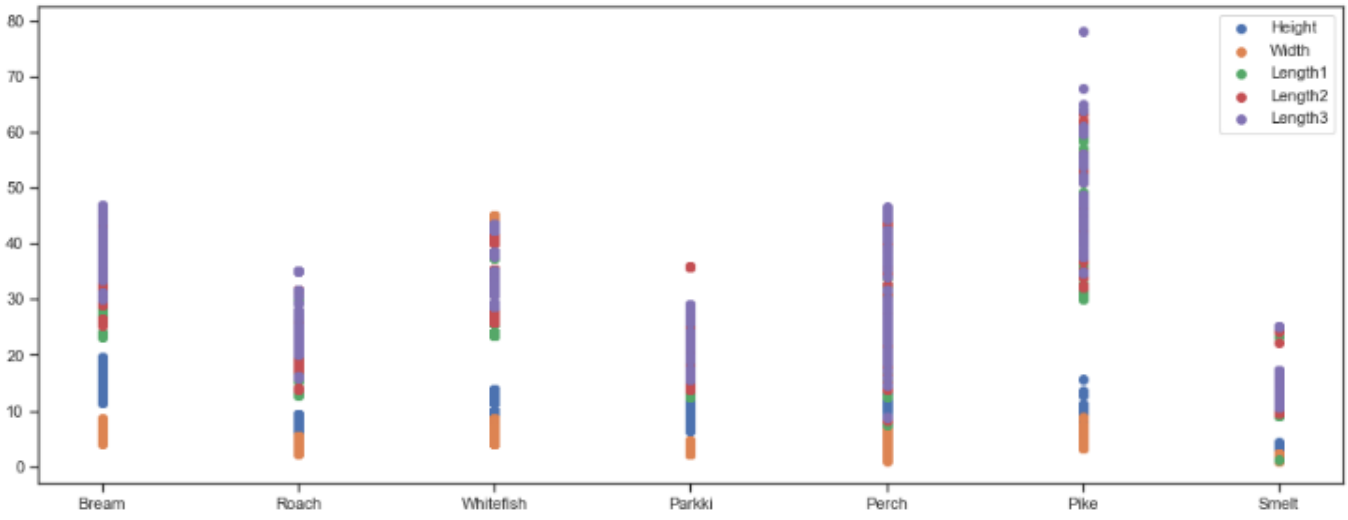
3.7 Выбор наиболее подходящих моделей для решения задачи классификации

1. Метод опорных векторов
2. Логистическая регрессия
3. Метод k-ближайших соседей
4. Дерево решений
5. Метод случайного леса (ансамблевый)
6. Классификатор усиления градиента (ансамблевый)

3.8 Формирование обучающей и тестовой выборок на основе исходного набора данных

In [26]:

```
X = fish2['Species']
y = fish2['Height']
z = fish2['Width']
q = fish2['Length1']
w = fish2['Length2']
r = fish2['Length3']

plt.figure(figsize=(16, 6))

plt.scatter(X,y,label='Height')
plt.scatter(X,z,label = 'Width')
plt.scatter(X,q,label = 'Length1')
plt.scatter(X,w,label = 'Length2')
plt.scatter(X,r,label = 'Length3')
plt.legend()
plt.show()
 fish.Species.unique(
```



Out[26]:

```
array(['Bream', 'Roach', 'Whitefish', 'Parkki', 'Perch', 'Pike', 'Smelt'],
      dtype=object)
```

In [27]:

```
X = fish.drop(columns=['Species'])
y = fish.Species
```

In [28]:

```
from sklearn.naive_bayes import *
```

In [29]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, ran
dom_state=42)
```

3.9 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.

In [30]:

```
SVC_model = svm.SVC()
LOG_model = LogisticRegression()
# В KNN-модели нужно указать параметр n_neighbors
# Это число точек, на которое будет смотреть
# классификатор, чтобы определить, к какому классу принадлежит новая точка
KNN_model = KNeighborsClassifier(n_neighbors=5)
DTC_model = DecisionTreeClassifier()
RandFor_model = RandomForestClassifier()
GBC_model = GradientBoostingClassifier()
```

In [31]:

```
SVC_model.fit(X_train, y_train)
LOG_model.fit(X_train,y_train)
KNN_model.fit(X_train, y_train)
DTC_model.fit(X_train, y_train)
RandFor_model.fit(X_train, y_train)
GBC_model.fit(X_train, y_train)
```

Out[31]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None
,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None
,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [32]:

```
SVC_prediction = SVC_model.predict(X_test)
LOG_prediction = LOG_model.predict(X_test)
KNN_prediction = KNN_model.predict(X_test)
```

```
DTC_prediction = DTC_model.predict(X_test)
RandFor_prediction = RandFor_model.predict(X_test)
GBC_prediction = GBC_model.predict(X_test)
```

In [33]:

```
# Оценка точности — простейший вариант оценки работы классификатора
print('SVC_accuracy: ',accuracy_score(SVC_prediction, y_test))
print('LOG_accuracy: ',accuracy_score(LOG_prediction, y_test))
print('KNN_accuracy: ',accuracy_score(KNN_prediction, y_test))
print('DTC_accuracy: ',accuracy_score(DTC_prediction, y_test))
print('RandFor_accuracy: ',accuracy_score(RandFor_prediction, y_test))
print('GBC_accuracy: ',accuracy_score(GBC_prediction, y_test))

# Но матрица неточности и отчёт о классификации дадут больше информации о про
изводительности
print(confusion_matrix(SVC_prediction, y_test))
print(confusion_matrix(LOG_prediction, y_test))
print(confusion_matrix(KNN_prediction, y_test))
print(confusion_matrix(DTC_prediction, y_test))
print(confusion_matrix(RandFor_prediction, y_test))
print(confusion_matrix(GBC_prediction, y_test))

print(classification_report(KNN_prediction, y_test))
print(classification_report(SVC_prediction, y_test))
print(classification_report(LOG_prediction, y_test))
print(classification_report(DTC_prediction, y_test))
print(classification_report(RandFor_prediction, y_test))
print(classification_report(GBC_prediction, y_test))
```

```
SVC_accuracy:  0.4153846153846154
LOG_accuracy:  0.7692307692307693
KNN_accuracy:  0.7769230769230769
DTC_accuracy:  0.9538461538461539
RandFor_accuracy:  0.9923076923076923
GBC_accuracy:  0.9923076923076923
[[16  0  6  2  0  0  8]
 [ 0  7  7  1 13  0  0]
 [ 0  0  0  0  0  0  0]
 [ 1  0  1  4  0  0  0]
 [ 0  0  0  0  0  0  0]
 [ 0  6  3  0  3 22  0]
 [ 7  4  9  2  3  0  5]]
[[24  0  0  0  0  0  2]
 [ 0 15  0  0  2  0  0]
 [ 0  2 12  0  7  0  1]
 [ 0  0  0  9  0  0  0]
 [ 0  0  7  0  8  0  0]
 [ 0  0  1  0  2 22  0]
 [ 0  0  6  0  0  0 10]]
[[24  0  3  0  2  0  0]
 [ 0 16  2  0  2  0  0]
```

```
 [ 0  1  8  0  3  0  0]
 [ 0  0  1  9  0  0  0]
 [ 0  0  5  0  9  0  0]
 [ 0  0  1  0  2 22  0]
 [ 0  0  6  0  1  0 13]]
[[24  0  0  0  0  0  0]
 [ 0 17  0  0  0  0  0]
 [ 0  0 22  0  2  0  0]
 [ 0  0  0  9  0  0  0]
 [ 0  0  3  0 17  0  0]
 [ 0  0  0  0  0 22  0]
 [ 0  0  1  0  0  0 13]]
[[24  0  0  0  0  0  0]
 [ 0 17  0  0  0  0  0]
 [ 0  0 25  0  0  0  0]
 [ 0  0  0  9  0  0  0]
 [ 0  0  0  0 19  0  0]
 [ 0  0  0  0  0 22  0]
 [ 0  0  1  0  0  0 13]]
[[24  0  0  0  0  0  0]
 [ 0 17  0  0  0  0  0]
 [ 0  0 26  1  0  0  0]
 [ 0  0  0  8  0  0  0]
 [ 0  0  0  0 19  0  0]
 [ 0  0  0  0  0 22  0]
 [ 0  0  0  0  0  0 13]]
              precision    recall  f1-score   support

       Bream       1.00      0.83      0.91        29
      Parkki       0.94      0.80      0.86        20
       Perch       0.31      0.67      0.42        12
        Pike       1.00      0.90      0.95        10
       Roach       0.47      0.64      0.55        14
       Smelt       1.00      0.88      0.94        25
  Whitefish        1.00      0.65      0.79        20

    accuracy                           0.78       130
   macro avg       0.82      0.77      0.77       130
weighted avg       0.87      0.78      0.81       130

              precision    recall  f1-score   support

       Bream       0.67      0.50      0.57        32
      Parkki       0.41      0.25      0.31        28
       Perch       0.00      0.00      0.00         0
        Pike       0.44      0.67      0.53         6
       Roach       0.00      0.00      0.00         0
       Smelt       1.00      0.65      0.79        34
  Whitefish        0.38      0.17      0.23        30
```

accuracy			0.42	130
macro avg	0.42	0.32	0.35	130
weighted avg	0.62	0.42	0.49	130

	precision	recall	f1-score	support
Bream	1.00	0.92	0.96	26
Parkki	0.88	0.88	0.88	17
Perch	0.46	0.55	0.50	22
Pike	1.00	1.00	1.00	9
Roach	0.42	0.53	0.47	15
Smelt	1.00	0.88	0.94	25
Whitefish	0.77	0.62	0.69	16
accuracy			0.77	130
macro avg	0.79	0.77	0.78	130
weighted avg	0.80	0.77	0.78	130

	precision	recall	f1-score	support
Bream	1.00	1.00	1.00	24
Parkki	1.00	1.00	1.00	17
Perch	0.85	0.92	0.88	24
Pike	1.00	1.00	1.00	9
Roach	0.89	0.85	0.87	20
Smelt	1.00	1.00	1.00	22
Whitefish	1.00	0.93	0.96	14
accuracy			0.95	130
macro avg	0.96	0.96	0.96	130
weighted avg	0.96	0.95	0.95	130

	precision	recall	f1-score	support
Bream	1.00	1.00	1.00	24
Parkki	1.00	1.00	1.00	17
Perch	0.96	1.00	0.98	25
Pike	1.00	1.00	1.00	9
Roach	1.00	1.00	1.00	19
Smelt	1.00	1.00	1.00	22
Whitefish	1.00	0.93	0.96	14
accuracy			0.99	130
macro avg	0.99	0.99	0.99	130
weighted avg	0.99	0.99	0.99	130

	precision	recall	f1-score	support

```
       Bream       1.00      1.00      1.00        24
      Parkki       1.00      1.00      1.00        17
       Perch       1.00      0.96      0.98        27
        Pike       0.89      1.00      0.94         8
       Roach       1.00      1.00      1.00        19
       Smelt       1.00      1.00      1.00        22
   Whitefish       1.00      1.00      1.00        13

    accuracy                           0.99       130
   macro avg       0.98      0.99      0.99       130
weighted avg       0.99      0.99      0.99       130
```

3.10 Подбор гиперпараметров для выбранных моделей. Использование кросс-валидации (GridSearchCV).

In [36]:

```
n_range = np.array(range(1,30,3))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[36]:

```
[{'n_neighbors': array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28])}]
```

In [37]:

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring
='accuracy')
clf_gs.fit(X_train, y_train)
Wall time: 702 ms
```

Out[37]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 1,  4,  7, 10, 13, 16, 19, 2
2, 25, 28])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [38]:

```
clf_gs.cv_results_
```

Out[38]:

```
{'mean_fit_time': array([0.00439305, 0.00312486, 0.00359054, 0.00431933, 0.00
373011,
        0.00624886, 0.00312448, 0.00312424, 0.00312433, 0.00312433]),
 'std_fit_time': array([0.003605  , 0.00624971, 0.00195407, 0.00579151, 0.006
07588,
        0.00765325, 0.00624895, 0.00624847, 0.00624866, 0.00624866]),
 'mean_score_time': array([0.01723561, 0.01775126, 0.00952873, 0.00971766, 0.
00452185,
        0.00624857, 0.00937304, 0.00312428, 0.00937262, 0.00312448]),
```

```
 'std_score_time': array([0.00566581, 0.00426147, 0.00442688, 0.00646655, 0.0
0616467,
        0.0076529 , 0.00765306, 0.00624857, 0.00765271, 0.00624895]),
 'param_n_neighbors': masked_array(data=[1, 4, 7, 10, 13, 16, 19, 22, 25, 28]
,
              mask=[False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 4},
  {'n_neighbors': 7},
  {'n_neighbors': 10},
  {'n_neighbors': 13},
  {'n_neighbors': 16},
  {'n_neighbors': 19},
  {'n_neighbors': 22},
  {'n_neighbors': 25},
  {'n_neighbors': 28}],
 'split0_test_score': array([0.94230769, 0.77884615, 0.70192308, 0.69230769,
0.61538462,
        0.63461538, 0.60576923, 0.52884615, 0.51923077, 0.46153846]),
 'split1_test_score': array([0.90291262, 0.7961165 , 0.70873786, 0.66990291,
0.59223301,
        0.58252427, 0.5631068 , 0.58252427, 0.54368932, 0.49514563]),
 'split2_test_score': array([0.93203883, 0.81553398, 0.69902913, 0.72815534,
0.70873786,
        0.57281553, 0.57281553, 0.51456311, 0.49514563, 0.45631068]),
 'split3_test_score': array([0.90291262, 0.78640777, 0.69902913, 0.6407767 ,
0.63106796,
        0.63106796, 0.67961165, 0.60194175, 0.55339806, 0.55339806]),
 'split4_test_score': array([0.88349515, 0.78640777, 0.72815534, 0.6407767 ,
0.6407767 ,
        0.60194175, 0.5631068 , 0.53398058, 0.48543689, 0.52427184]),
 'mean_test_score': array([0.91273338, 0.79266243, 0.70737491, 0.67438387, 0.
63764003,
        0.60459298, 0.596882  , 0.55237117, 0.51938013, 0.49813294]),
 'std_test_score': array([0.0214248 , 0.01268203, 0.01097917, 0.0331409 , 0.0
3915363,
        0.02492442, 0.04423551, 0.03372697, 0.02641073, 0.03697204]),
 'rank_test_score': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10], dtype=int
32)}
```

In [39]:

```
# Лучшая модель
clf_gs.best_estimator_
```

Out[39]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [40]:

```
# Лучшее значение метрики
clf_gs.best_score_
```

Out[40]:

```
0.9127333831217326
```

In [41]:

```
# Лучшее значение параметров
clf_gs.best_params_
```

Out[41]:

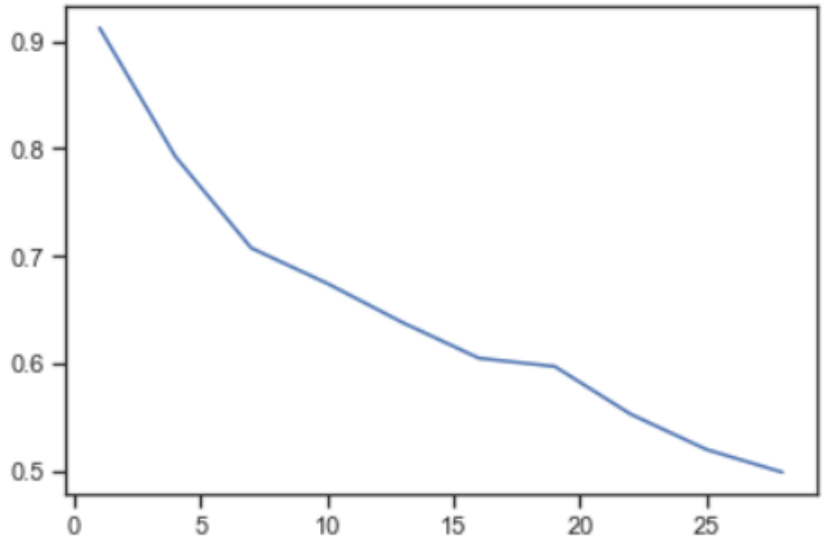
```
{'n_neighbors': 1}
```

In [42]:

```
# Изменение качества на тестовой выборке в зависимости от К-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

Out[42]:

```
[<matplotlib.lines.Line2D at 0x1275e550>]
```



Randomised Search

In [43]:

```
%%time
clf_rs = RandomizedSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, s
coring='accuracy')
clf_rs.fit(X_train, y_train)
Wall time: 870 ms
```

Out[43]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=KNeighborsClassifier(algorithm='auto',
                                                  leaf_size=30,
                                                  metric='minkowski',
                                                  metric_params=None,
                                                  n_jobs=None, n_neighbors=5,
                                                  p=2, weights='uniform'),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions=[{'n_neighbors': array([ 1,  4,  7, 10
, 13, 16, 19, 22, 25, 28])}],
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
```

```
                    return_train_score=False, scoring='accuracy', verbose=0)
```

In [44]:

```
# В данном случае оба способа нашли одинаковое решение
clf_rs.best_score_, clf_rs.best_params_
```

Out[44]:

```
(0.9127333831217326, {'n_neighbors': 1})
```

In [45]:

```
clf_gs.best_score_, clf_gs.best_params_
```

Out[45]:

```
(0.9127333831217326, {'n_neighbors': 1})
```

3.11 Построение кривых обучения

In [46]:

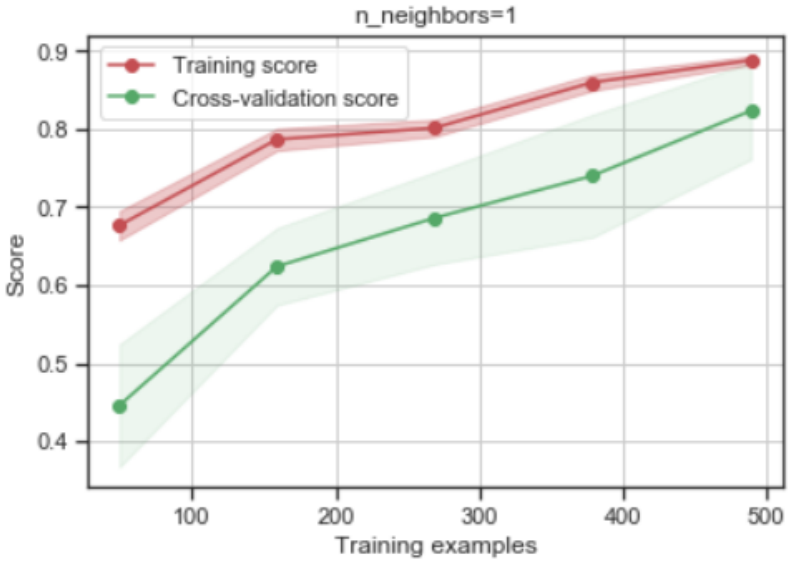
```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, 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=0.3,
                     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
```

In [47]:

```
plot_learning_curve(KNeighborsClassifier(n_neighbors=5), 'n_neighbors=1',
                    X_train, y_train, cv=20)
```

Out[47]:

```
<module 'matplotlib.pyplot' from 'c:\\users\\user\\appdata\\local\\programs\\
python\\python37-32\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



3.12 Построение кривых валидации

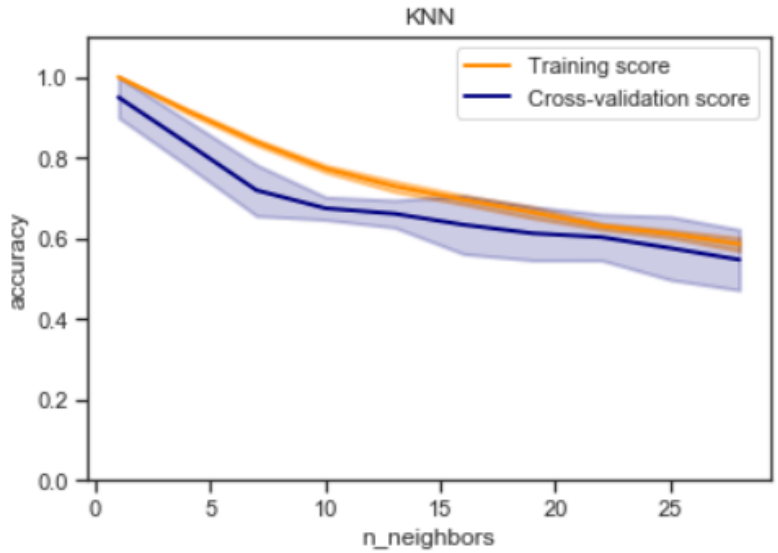
In [48]:

```
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(str(scoring))
    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=0.4,
                     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
```

In [49]:

```
plot_validation_curve(KNeighborsClassifier(), 'KNN',
                      X_train, y_train,
                      param_name='n_neighbors', param_range=n_range,
                      cv=20, scoring="accuracy")
```

Out[49]:



3.13 Повторение пункта 3.9 для найденных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей

In [50]:

```
KNN_model = KNeighborsClassifier(n_neighbors=1)
```

In [51]:

```
KNN_model.fit(X_train, y_train)
```

Out[51]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [52]:

```
KNN_prediction = KNN_model.predict(X_test)
```

In [53]:

```
print('KNN_accuracy: ',accuracy_score(KNN_prediction, y_test))
```

```
KNN_accuracy:  0.9846153846153847
```

3.14 Формирование выводов о качестве построенных моделей на основе выбранных метрик.

In [54]:

```
SVC_model = svm.SVC()
LOG_model = LogisticRegression()
# В KNN-модели нужно указать параметр n_neighbors
# Это число точек, на которое будет смотреть
```

```
# классификатор, чтобы определить, к какому классу принадлежит новая точка
KNN_model = KNeighborsClassifier(n_neighbors=1)
DTC_model = DecisionTreeClassifier()
RandFor_model = RandomForestClassifier()
GBC_model = GradientBoostingClassifier()
```

In [55]:

```
SVC_model.fit(X_train, y_train)
LOG_model.fit(X_train,y_train)
KNN_model.fit(X_train, y_train)
DTC_model.fit(X_train, y_train)
RandFor_model.fit(X_train, y_train)
GBC_model.fit(X_train, y_train)
```

```
c:\users\user\appdata\local\programs\python\python37-32\lib\site-packages\skl
earn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed to conve
rge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[55]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None
,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None
,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [56]:

```
SVC_prediction = SVC_model.predict(X_test)
LOG_prediction = LOG_model.predict(X_test)
KNN_prediction = KNN_model.predict(X_test)
DTC_prediction = DTC_model.predict(X_test)
RandFor_prediction = RandFor_model.predict(X_test)
GBC_prediction = GBC_model.predict(X_test)
```

In [60]:

```
print('SVC_accuracy: ',accuracy_score(SVC_prediction, y_test))
print('LOG_accuracy: ',accuracy_score(LOG_prediction, y_test))
print('KNN_accuracy: ',accuracy_score(KNN_prediction, y_test))
print('DTC_accuracy: ',accuracy_score(DTC_prediction, y_test))
print('RandFor_accuracy: ',accuracy_score(RandFor_prediction, y_test))
```

```
print('GBC_accuracy: ',accuracy_score(GBC_prediction, y_test))

# Но матрица неточности и отчёт о классификации дадут больше информации о про
изводительности
print('\nConfusion matrix для SVC:\n')
print(confusion_matrix(SVC_prediction, y_test))
print('\nConfusion matrix для LOG:\n')
print(confusion_matrix(LOG_prediction, y_test))
print('\nConfusion matrix для KNN:\n')
print(confusion_matrix(KNN_prediction, y_test))
print('\nConfusion matrix для DTC:\n')
print(confusion_matrix(DTC_prediction, y_test))
print('\nConfusion matrix для RandomFor:\n')
print(confusion_matrix(RandFor_prediction, y_test))
print('\nConfusion matrix для GBC:\n')
print(confusion_matrix(GBC_prediction, y_test))

print('\nClassification report для KNN:\n')
print(classification_report(KNN_prediction, y_test))
print('\nClassification report для SVC:\n')
print(classification_report(SVC_prediction, y_test))
print('\nClassification report для LOG:\n')
print(classification_report(LOG_prediction, y_test))
print('\nClassification report для DTC:\n')
print(classification_report(DTC_prediction, y_test))
print('\nClassification report для RandomFor:\n')
print(classification_report(RandFor_prediction, y_test))
print('\nClassification report для GBC:\n')
print(classification_report(GBC_prediction, y_test))
SVC_accuracy:  0.4153846153846154
LOG_accuracy:  0.7692307692307693
KNN_accuracy:  0.9846153846153847
DTC_accuracy:  0.9769230769230769
RandFor_accuracy:  1.0
GBC_accuracy:  0.9923076923076923

Confusion matrix для SVC:

[[16  0  6  2  0  0  8]
 [ 0  7  7  1 13  0  0]
 [ 0  0  0  0  0  0  0]
 [ 1  0  1  4  0  0  0]
 [ 0  0  0  0  0  0  0]
 [ 0  6  3  0  3 22  0]
 [ 7  4  9  2  3  0  5]]

Confusion matrix для LOG:

[[24  0  0  0  0  0  2]
```

```
 [ 0 15  0  0  2  0  0]
 [ 0  2 12  0  7  0  1]
 [ 0  0  0  9  0  0  0]
 [ 0  0  7  0  8  0  0]
 [ 0  0  1  0  2 22  0]
 [ 0  0  6  0  0  0 10]]
```

Confusion matrix для KNN:

```
[[24  0  0  0  0  0  0]
 [ 0 17  0  0  0  0  0]
 [ 0  0 24  0  0  0  0]
 [ 0  0  1  9  0  0  0]
 [ 0  0  1  0 19  0  0]
 [ 0  0  0  0  0 22  0]
 [ 0  0  0  0  0  0 13]]
```

Confusion matrix для DTC:

```
[[24  0  0  0  0  0  0]
 [ 0 17  0  0  0  0  0]
 [ 0  0 25  0  2  0  0]
 [ 0  0  0  9  0  0  0]
 [ 0  0  1  0 17  0  0]
 [ 0  0  0  0  0 22  0]
 [ 0  0  0  0  0  0 13]]
```

Confusion matrix для RandomFor:

```
[[24  0  0  0  0  0  0]
 [ 0 17  0  0  0  0  0]
 [ 0  0 26  0  0  0  0]
 [ 0  0  0  9  0  0  0]
 [ 0  0  0  0 19  0  0]
 [ 0  0  0  0  0 22  0]
 [ 0  0  0  0  0  0 13]]
```

Confusion matrix для GBC:

```
[[24  0  0  0  0  0  0]
 [ 0 17  0  0  0  0  0]
 [ 0  0 26  1  0  0  0]
 [ 0  0  0  8  0  0  0]
 [ 0  0  0  0 19  0  0]
 [ 0  0  0  0  0 22  0]
 [ 0  0  0  0  0  0 13]]
```

Classification report для KNN:

	precision	recall	f1-score	support
Bream	1.00	1.00	1.00	24
Parkki	1.00	1.00	1.00	17
Perch	0.92	1.00	0.96	24
Pike	1.00	0.90	0.95	10
Roach	1.00	0.95	0.97	20
Smelt	1.00	1.00	1.00	22
Whitefish	1.00	1.00	1.00	13
accuracy			0.98	130
macro avg	0.99	0.98	0.98	130
weighted avg	0.99	0.98	0.98	130

Classification report для SVC:

	precision	recall	f1-score	support
Bream	0.67	0.50	0.57	32
Parkki	0.41	0.25	0.31	28
Perch	0.00	0.00	0.00	0
Pike	0.44	0.67	0.53	6
Roach	0.00	0.00	0.00	0
Smelt	1.00	0.65	0.79	34
Whitefish	0.38	0.17	0.23	30
accuracy			0.42	130
macro avg	0.42	0.32	0.35	130
weighted avg	0.62	0.42	0.49	130

Classification report для LOG:

	precision	recall	f1-score	support
Bream	1.00	0.92	0.96	26
Parkki	0.88	0.88	0.88	17
Perch	0.46	0.55	0.50	22
Pike	1.00	1.00	1.00	9
Roach	0.42	0.53	0.47	15
Smelt	1.00	0.88	0.94	25
Whitefish	0.77	0.62	0.69	16
accuracy			0.77	130
macro avg	0.79	0.77	0.78	130
weighted avg	0.80	0.77	0.78	130

Classification report для DTC:

	precision	recall	f1-score	support
Bream	1.00	1.00	1.00	24
Parkki	1.00	1.00	1.00	17
Perch	0.96	0.93	0.94	27
Pike	1.00	1.00	1.00	9
Roach	0.89	0.94	0.92	18
Smelt	1.00	1.00	1.00	22
Whitefish	1.00	1.00	1.00	13
accuracy			0.98	130
macro avg	0.98	0.98	0.98	130
weighted avg	0.98	0.98	0.98	130

Classification report для RandomFor:

	precision	recall	f1-score	support
Bream	1.00	1.00	1.00	24
Parkki	1.00	1.00	1.00	17
Perch	1.00	1.00	1.00	26
Pike	1.00	1.00	1.00	9
Roach	1.00	1.00	1.00	19
Smelt	1.00	1.00	1.00	22
Whitefish	1.00	1.00	1.00	13
accuracy			1.00	130
macro avg	1.00	1.00	1.00	130
weighted avg	1.00	1.00	1.00	130

Classification report для GBC:

	precision	recall	f1-score	support
Bream	1.00	1.00	1.00	24
Parkki	1.00	1.00	1.00	17
Perch	1.00	0.96	0.98	27
Pike	0.89	1.00	0.94	8
Roach	1.00	1.00	1.00	19
Smelt	1.00	1.00	1.00	22
Whitefish	1.00	1.00	1.00	13
accuracy			0.99	130
macro avg	0.98	0.99	0.99	130
weighted avg	0.99	0.99	0.99	130

3.15 Вывод

На основании метрики accuracy можно сделать вывод, что наилучшим методом ддя данного набора данных оказался метод случайного леса.

4. Заключение

В процессе выполнения курсовой работы были закреплены навыки, полученные в течении семестра.

5. Список литературы

1. Конспект лекций по предмету «Технологии машинного обучения» Ю.Е. Гапанюка (2020 год, МГТУ им. Н.Э. Баумана);
2. https://tproger.ru/translations/scikit-learn-in-python/ (Дата обращения: 30.05.2020)
3. https://m.habr.com/ru/post/264241/ (Дата обращения: 30.05.2020)