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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Утверждаю: |  |
| Гапанюк Ю.Е. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. |

**Курсовая работа по дисциплине**

**«Технологии машинного обучения»**

Пояснительная записка

(вид документа)

писчая бумага

(вид носителя)

30

(количество листов)

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студентка группы ИУ5-63б |  |
| Нефёдова И.В. | "\_\_\_"\_\_\_\_\_\_\_\_\_2020 г. |

Москва - 2020

Оглавление

[1. Задание установленного образца. 3](#_Toc42115314)

[2. Введение. 3](#_Toc42115315)

[3. Основная часть, содержащая описание постановки задачи и последовательности действий студента по решению поставленной задачи. 4](#_Toc42115316)

[3.1 Описание набора данных 4](#_Toc42115317)

[3.2 Ход работы 4](#_Toc42115318)

[3.3 Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных. 5](#_Toc42115319)

[3.4 Выбор признаков, подходящих для построения моделей. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей. 10](#_Toc42115320)

[3.5 Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. 12](#_Toc42115321)

[3.6 Выбор метрик для последующей оценки качества моделей 13](#_Toc42115322)

[3.7 Выбор наиболее подходящих моделей для решения задачи классификации 14](#_Toc42115323)

[3.8 Формирование обучающей и тестовой выборок на основе исходного набора данных 14](#_Toc42115324)

[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. 15](#_Toc42115325)

[3.10 Подбор гиперпараметров для выбранных моделей. Использование кросс-валидации (GridSearchCV). 19](#_Toc42115326)

[3.11 Построение кривых обучения 22](#_Toc42115328)

[3.12 Построение кривых валидации 23](#_Toc42115329)

[3.13 Повторение пункта 3.9 для найденных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей 24](#_Toc42115330)

[3.14 Формирование выводов о качестве построенных моделей на основе выбранных метрик. 24](#_Toc42115331)

[3.15 Вывод 30](#_Toc42115332)

[4. Заключение 30](#_Toc42115333)

[5. Список литературы 30](#_Toc42115334)

# Задание установленного образца.

Схема типового исследования, проводимого студентом в рамках курсовой работы, содержит выполнение следующих шагов:

* + Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
  + Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
  + Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
  + Проведение корреляционного анализа данных. Формирование

промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.

* + Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
  + Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из

которых должны быть ансамблевыми.

* + Формирование обучающей и тестовой выборок на основе исходного набора данных.
  + Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производятся обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
  + Подбор гиперпараметров для выбранных моделей. Рекомендуется

использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать

перебор параметров в цикле, или использовать другие методы.

* + Повторение пункта 8 для найденных оптимальных значений

гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

* + Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового

описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

# Введение.

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.

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

# 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.

## 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, learning\_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, classification\_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\_squared\_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, LinearSVR

**from** **sklearn.tree** **import** DecisionTreeClassifier, DecisionTreeRegressor, export\_graphviz

**from** **sklearn.ensemble** **import** RandomForestClassifier, RandomForestRegressor

**from** **sklearn.ensemble** **import** ExtraTreesClassifier, ExtraTreesRegressor

**from** **sklearn.ensemble** **import** GradientBoostingClassifier, GradientBoostingRegressor

**from** **sklearn** **import** svm

%**matplotlib** inline

sns.set(style="ticks")

Загружаем данные:

In [2]:

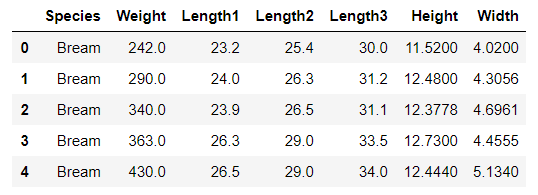
fish = pd.read\_csv('fish2.csv', sep=",")

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

In [3]:

fish.head()

Out[3]:



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.

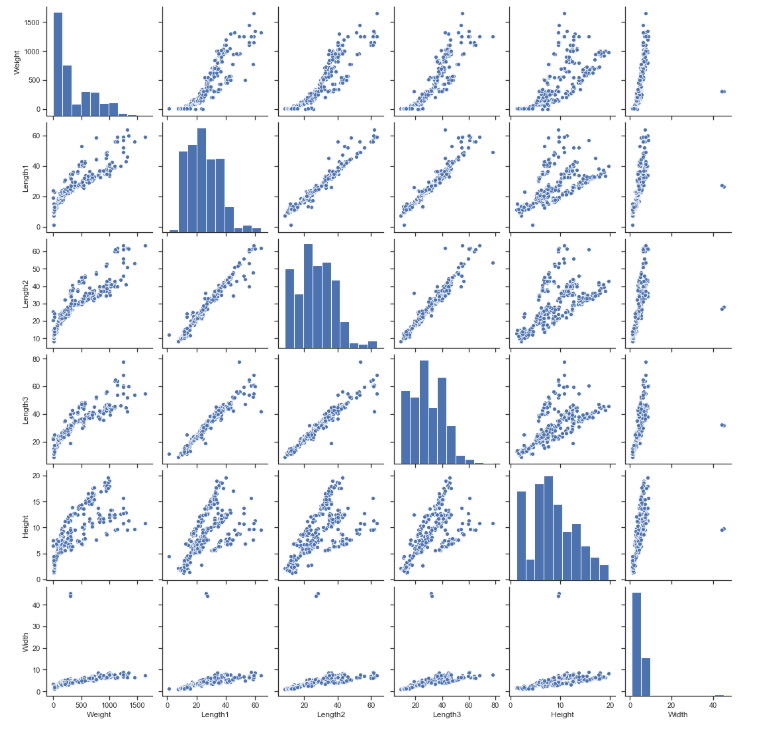
Оценим структуру представленных данных:

In [8]:

sns.pairplot(fish)

Out[8]:

<seaborn.axisgrid.PairGrid at 0xdf47750>

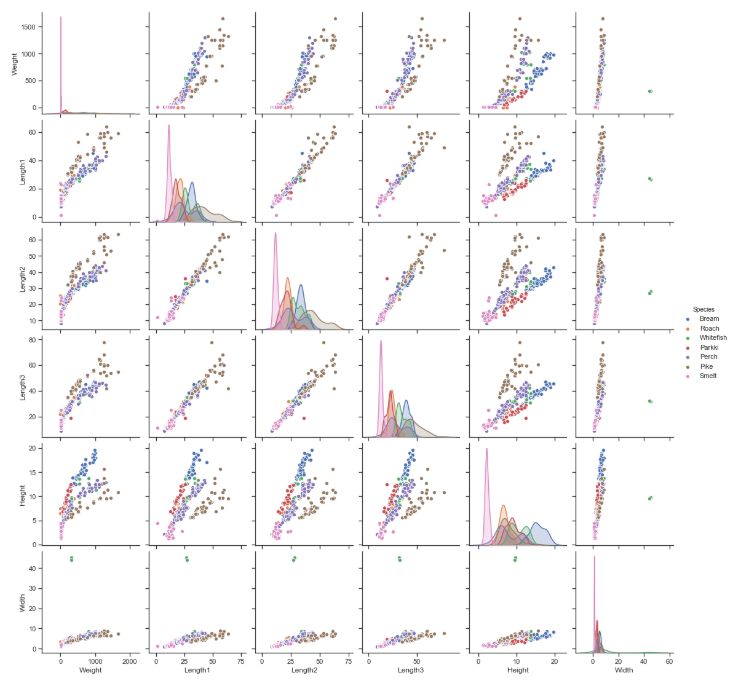


In [10]:

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.

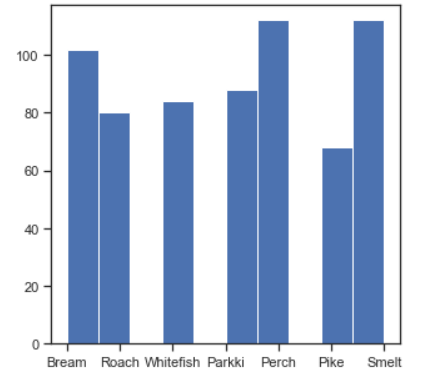
In [12]:

*# 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 **{}**%, **\n**класс 1 составляет **{}**%, **\n**класс 2 составляет **{}**%, **\n**класс 3 составляет **{}**%, **\n**Класс 4 составляет **{}**%, **\n**класс 5 составляет **{}**%, **\n**класс 6 составляет **{}**%, '

.format(round(class\_0 / total, 4)\*100, round(class\_1 / total, 4)\*100, round(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.

## 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))

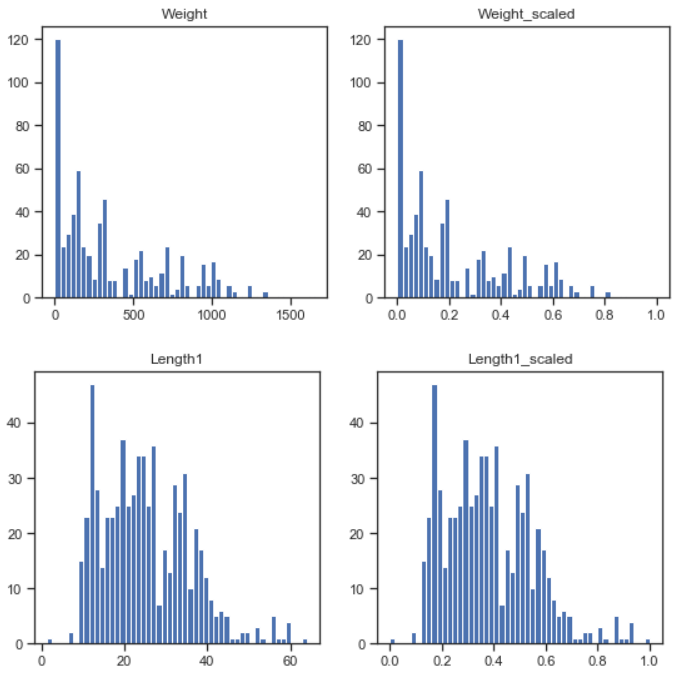
ax[0].hist(fish2[col], 50)

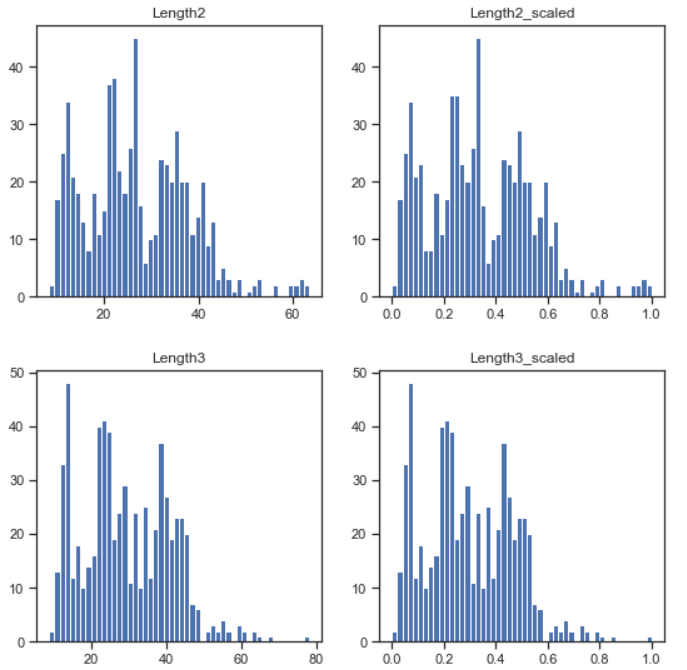
ax[1].hist(fish2[col\_scaled], 50)

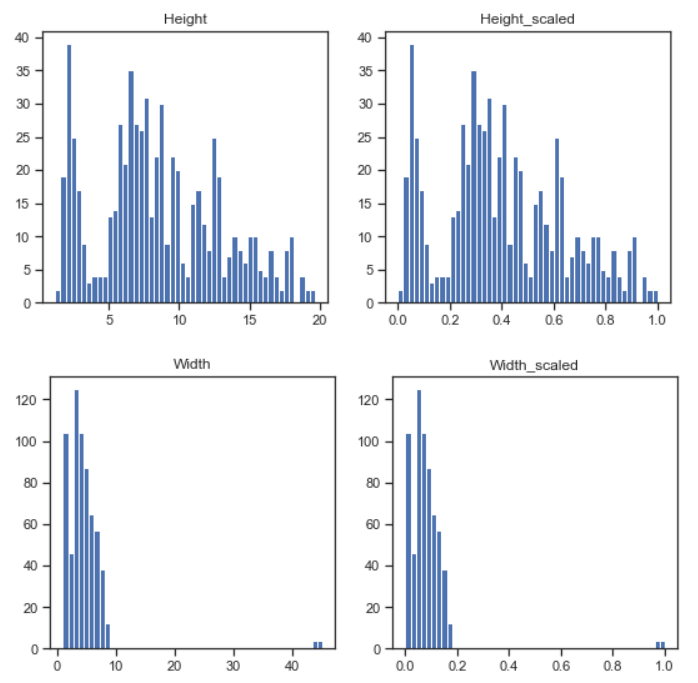
ax[0].title.set\_text(col)

ax[1].title.set\_text(col\_scaled)

plt.show()







## 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']

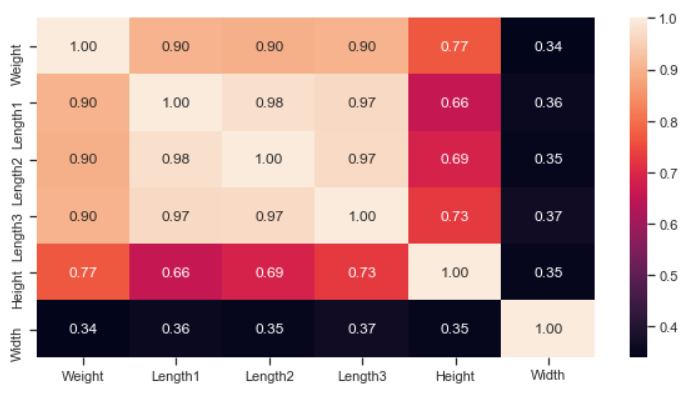
In [24]:

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>



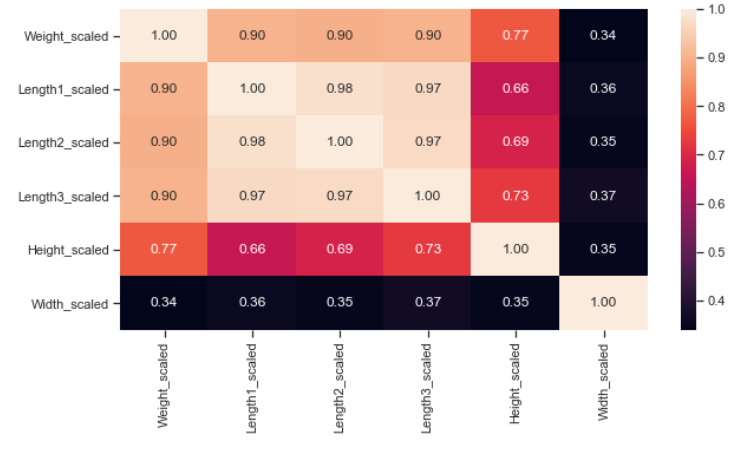
In [25]:

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

## Выбор метрик для последующей оценки качества моделей

1. Метрика accuracy
2. Метрика confusion matrix
3. Метрика classification report

## 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)

## 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')

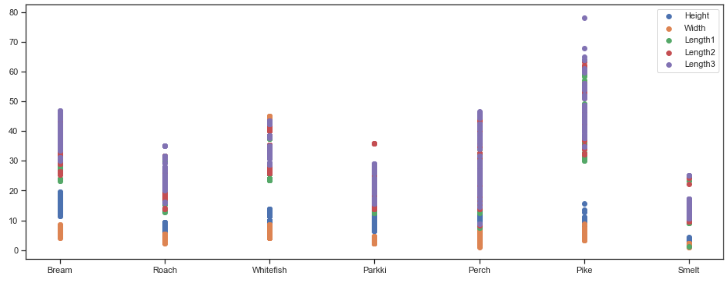
plt.scatter(X,q,label = 'Length1')

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, random\_state=42)

## 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 производительности*

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

## 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, 22, 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, 0.00373011,

0.00624886, 0.00312448, 0.00312424, 0.00312433, 0.00312433]),

'std\_fit\_time': array([0.003605 , 0.00624971, 0.00195407, 0.00579151, 0.00607588,

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, 0.00616467,

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 , 0.03915363,

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=int32)}

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}

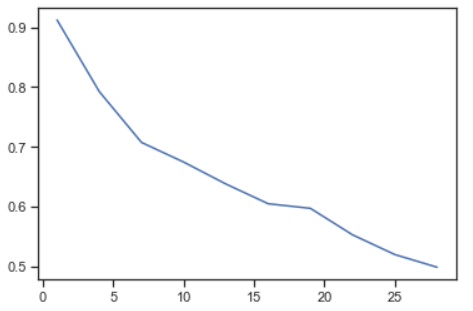
In [42]:

*# 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, scoring='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})

## 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

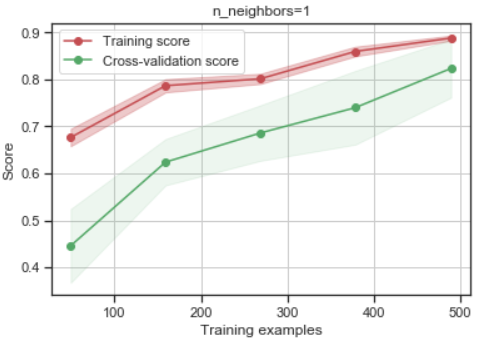
In [47]:

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'>



## 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]:

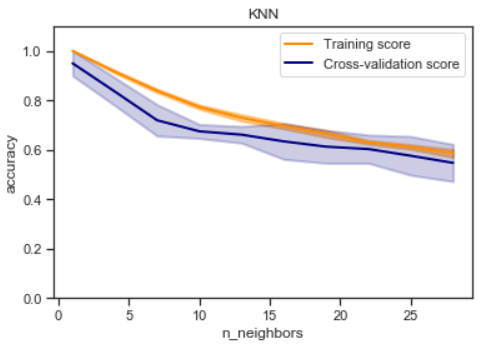
plot\_validation\_curve(KNeighborsClassifier(), 'KNN',

X\_train, y\_train,

param\_name='n\_neighbors', param\_range=n\_range,

cv=20, scoring="accuracy")

Out[49]:



## 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

## 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\sklearn\linear\_model\\_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (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-regression

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 производительности*

print('**\n**Confusion matrix для SVC:**\n**')

print(confusion\_matrix(SVC\_prediction, y\_test))

print('**\n**Confusion matrix для LOG:**\n**')

print(confusion\_matrix(LOG\_prediction, y\_test))

print('**\n**Confusion matrix для KNN:**\n**')

print(confusion\_matrix(KNN\_prediction, y\_test))

print('**\n**Confusion matrix для DTC:**\n**')

print(confusion\_matrix(DTC\_prediction, y\_test))

print('**\n**Confusion matrix для RandomFor:**\n**')

print(confusion\_matrix(RandFor\_prediction, y\_test))

print('**\n**Confusion matrix для GBC:**\n**')

print(confusion\_matrix(GBC\_prediction, y\_test))

print('**\n**Classification report для KNN:**\n**')

print(classification\_report(KNN\_prediction, y\_test))

print('**\n**Classification report для SVC:**\n**')

print(classification\_report(SVC\_prediction, y\_test))

print('**\n**Classification report для LOG:**\n**')

print(classification\_report(LOG\_prediction, y\_test))

print('**\n**Classification report для DTC:**\n**')

print(classification\_report(DTC\_prediction, y\_test))

print('**\n**Classification report для RandomFor:**\n**')

print(classification\_report(RandFor\_prediction, y\_test))

print('**\n**Classification 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

## Вывод

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

# Заключение

В процессе выполнения курсовой работы были закреплены навыки, полученные в течении семестра. Глубже изучен материал, озвученный на лекциях.

# 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)