# Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите 1) одну из линейных моделей, 2) SVM и 3) дерево решений. Оцените качество моделей с помощью трех подходящих для задачи метрик. Сравните качество полученных моделей.
5. Произведите для каждой модели подбор одного гиперпараметра с использованием GridSearchCV и кросс-валидации.
6. Повторите пункт 4 для найденных оптимальных значений гиперпараметров. Сравните качество полученных моделей с качеством моделей, полученных в пункте 4.

Датасет: wine

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import learning_curve, validation_curve
from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut, LeavePOut, Shuffle
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import roc_curve,confusion_matrix, roc_auc_score, accuracy_score, ba

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LinearRegression

import warnings

warnings.filterwarnings('ignore')
plt.style.use('ggplot')
```

```
# Считывание данных
data = pd.read_csv('Wine.csv', sep=";")
data.head()
```

| | Class | Alcohol | Malic acid | Ash | Alcalinity of ash | Magnesium | Total phenols | Flavanoids | Nonfla pl |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 14.23 | 1.71 | 2.43 | 15.6 | 127 | 2.80 | 3.06 | |
| **1** | 1 | 13.20 | 1.78 | 2.14 | 11.2 | 100 | 2.65 | 2.76 | |
| **2** | 1 | 13.16 | 2.36 | 2.67 | 18.6 | 101 | 2.80 | 3.24 | |
| **3** | 1 | 14.37 | 1.95 | 2.50 | 16.8 | 113 | 3.85 | 3.49 | |

```
# Типы данных
data.dtypes
```

```
Class                           int64
Alcohol                       float64
Malic acid                    float64
Ash                           float64
Alcalinity of ash             float64
Magnesium                       int64
Total phenols                 float64
Flavanoids                    float64
Nonflavanoid phenols          float64
Proanthocyanins               float64
Color intensity               float64
Hue                           float64
OD280/OD315 of diluted wines  float64
Proline                         int64
dtype: object
```

```
# Проверка на пустые значения
for col in data.columns:
    print('{} - {}'.format(col, data[data[col].isnull()].shape[0]))
```

```
Class - 0
Alcohol - 0
Malic acid - 0
Ash - 0
Alcalinity of ash - 0
Magnesium - 0
Total phenols - 0
Flavanoids - 0
Nonflavanoid phenols - 0
Proanthocyanins - 0
Color intensity - 0
Hue - 0
OD280/OD315 of diluted wines - 0
Proline - 0
```

```
# Размерность данных
data.shape
```

```
(178, 14)
```

```
CLASS = 'Class'
RANDOM_STATE = 17
TEST_SIZE = 0.3

X = data.drop(CLASS, axis=1).values
Y = data[CLASS].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=TEST_SIZE, random_sta
print('X_train: {}'.format(X_train.shape))
print('X_test: {}'.format(X_test.shape))
```

```
X_train: (124, 13)
X_test: (54, 13)
```

# Обучите 1) одну из линейных моделей, 2) SVM и 3) дерево решений. Оцените качество

# моделей с помощью трех подходящих для задачи метрик. Сравните качество полученных моделей.

SVM

```
# SVM
clf = SVC(gamma='auto')
clf.fit(X_train, Y_train)
clf.score(X_test, Y_test)
```

```
0.3888888888888889
```

```
Y_pred = clf.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        18
           2       0.39      1.00      0.56        21
           3       0.00      0.00      0.00        15

   micro avg       0.39      0.39      0.39        54
   macro avg       0.13      0.33      0.19        54
weighted avg       0.15      0.39      0.22        54
```

DTREE

```
# Decision tree
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, Y_train)
tree.score(X_test, Y_test)
```

```
0.9814814814814815
```

```
Y_pred = tree.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           1       0.95      1.00      0.97        18
           2       1.00      0.95      0.98        21
           3       1.00      1.00      1.00        15

   micro avg       0.98      0.98      0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54
```

LINEAR REGRESSION

```
lin = LinearRegression()
lin.fit(X_train, Y_train)
lin.score(X_test, Y_test)
```

```
0.8820501536198686
```

# Произведите для каждой модели подбор одного гиперпараметра с использованием GridSearchCV и кросс-валидации.

SVM

Основная идея метода — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

```
CROSS_VALIDATOR_GENERATOR = 5
PARAMETER_TAG = 'C'
PARAMETER_MAX_VALUE = 3

param_grid = {PARAMETER_TAG : np.arange(0.01, PARAMETER_MAX_VALUE, 0.01)}
clf = SVC(gamma='auto')

clf_cv = GridSearchCV(clf, param_grid, cv = CROSS_VALIDATOR_GENERATOR)
clf_cv.fit(X_train,Y_train)
clf_cv.best_score_
```

```
0.47580645161290325
```

```
clf_cv.best_params_
```

```
{'C': 1.21}
```

```
clf = SVC(gamma='auto', C = clf_cv.best_params_[PARAMETER_TAG])
clf.fit(X_train, Y_train)
clf.score(X_test, Y_test)
```

```
0.4074074074074074
```

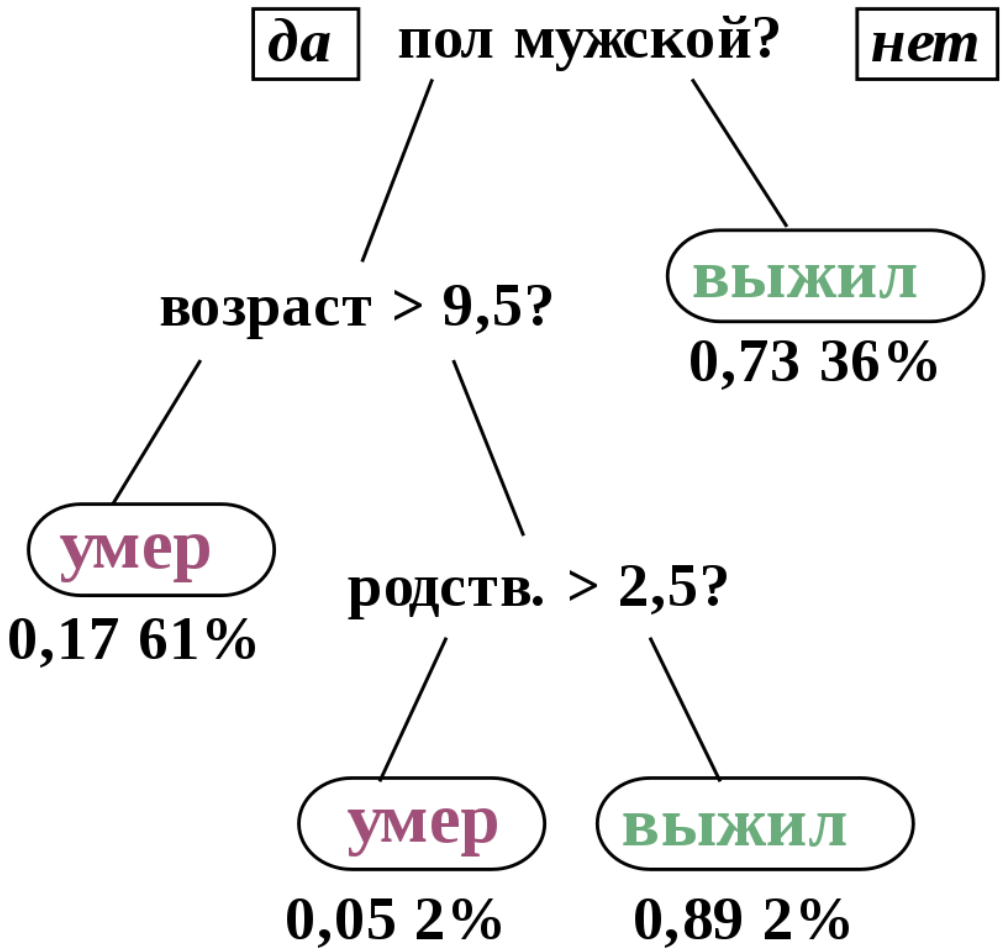
```
Y_pred = clf.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           1       1.00      0.06      0.11        18
           2       0.40      1.00      0.57        21
```

DTREE

Структура дерева представляет собой «листья» и «ветки». На рёбрах («ветках») дерева решения записаны атрибуты, от которых зависит целевая функция, в «листьях» записаны значения целевой функции, а в остальных узлах — атрибуты, по которым различаются случаи. Чтобы классифицировать новый случай, надо спуститься по дереву до листа и выдать соответствующее значение




```
PARAMETER_TAG = 'min_impurity_decrease'

param_grid = {PARAMETER_TAG : np.arange(0.01, PARAMETER_MAX_VALUE, 0.01)}
tree = DecisionTreeClassifier(random_state=0)


tree_cv = GridSearchCV(tree, param_grid, cv = CROSS_VALIDATOR_GENERATOR)
tree_cv.fit(X_train,Y_train)
tree_cv.best_score_
```

```
0.9193548387096774
```

```
tree_cv.best_params_
```

```
{'min_impurity_decrease': 0.05}
```

```
# Decision tree
tree = DecisionTreeClassifier(random_state=0, min_impurity_decrease = tree_cv.best_params
tree.fit(X_train, Y_train)
tree.score(X_test, Y_test)
```

```
0.9259259259259259
```

```
Y_pred = tree.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           1       0.82      1.00      0.90        18
           2       1.00      0.81      0.89        21
           3       1.00      1.00      1.00        15

   micro avg       0.93      0.93      0.93        54
   macro avg       0.94      0.94      0.93        54
weighted avg       0.94      0.93      0.93        54
```

LINEAR REGRESSION

егрессио́нный анализ — статистический метод исследования влияния одной или нескольких независимых переменных {\displaystyle X_{1},X_{2},...,X_{p}} X_{1},X_{2},...,X_{p} на зависимую переменную {\displaystyle Y} Y

```
PARAMETER_TAG = 'n_jobs'

param_grid = {PARAMETER_TAG : np.arange(1, 100)}
lin = LinearRegression()
```

```
lin_cv = GridSearchCV(lin, param_grid, cv = CROSS_VALIDATOR_GENERATOR)
lin_cv.fit(X_train, Y_train)
lin_cv.best_score_
```

```
0.8673662670575766
```

CODE TEXT

```
lin_cv.best_params_
```

```
{'n_jobs': 1}
```

```
lin = LinearRegression(n_jobs = lin_cv.best_params_[PARAMETER_TAG])
lin.fit(X_train, Y_train)
lin.score(X_test, Y_test)
```

```
0.8820501536198686
```