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

Факультет «Информатика и системы управления» Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»
Отчет по лабораторной работе №3
«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей.»

Выполнил:
Студент(ка) группы ИУ5-65Б
Тазенков Иван Дмитриевич

Подпись:_______________

Дата:________________

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий Евгеньевич

Подпись:______________

Дата:______________

Москва, 2023 г.

**Задание:**

1 Выберите набор данных (датасет) для решения задачи классификации или регрессии.

2 С использованием метода train_test_split разделите выборку на обучающую и тестовую.

3 Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.

4 Произведите подбор гиперпараметра K с использованием GridSearchCV и/или RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Желательно использование нескольких стратегий кросс-валидации.

5 Сравните метрики качества исходной и оптимальной моделей.

In [297]:

```
%matplotlib inline
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from collections import Counter
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")

# metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score,
roc_curve, f1_score
from sklearn.metrics import log_loss
# data
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# ML
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

In [18]:

```
data = pd.read_csv('creditcard.csv')

data.head()
```

Out[18]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 | 0.277838 | 0. |
| **1** | 0.0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | ... | -0.225775 | -0.638672 | 0. |
| **2** | 1.0 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | ... | 0.247998 | 0.771679 | 0. |
| **3** | 1.0 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | ... | -0.108300 | 0.005274 | 0. |
| **4** | 2.0 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | ... | -0.009431 | 0.798278 | 0. |

**5 rows × 31 columns**

In [19]:

```
X = data.drop(columns=['Class'])
y = data['Class']
```

In [20]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
```

```
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [21]:

```
data.describe()
```

Out[21]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 284807.000000 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2 |
| **mean** | 94813.859575 | 1.168375e-15 | 3.416908e-16 | -1.379537e-15 | 2.074095e-15 | 9.604066e-16 | 1.487313e-15 | -5.556467e-16 | 1 |
| **std** | 47488.145955 | 1.958696e+00 | 1.651309e+00 | 1.516255e+00 | 1.415869e+00 | 1.380247e+00 | 1.332271e+00 | 1.237094e+00 | 1 |
| **min** | 0.000000 | -5.640751e+01 | -7.271573e+01 | -4.832559e+01 | -5.683171e+00 | -1.137433e+02 | -2.616051e+01 | -4.355724e+01 | 7 |
| **25%** | 54201.500000 | -9.203734e-01 | -5.985499e-01 | -8.903648e-01 | -8.486401e-01 | -6.915971e-01 | -7.682956e-01 | -5.540759e-01 | |
| **50%** | 84692.000000 | 1.810880e-02 | 6.548556e-02 | 1.798463e-01 | -1.984653e-02 | -5.433583e-02 | -2.741871e-01 | 4.010308e-02 | 2 |
| **75%** | 139320.500000 | 1.315642e+00 | 8.037239e-01 | 1.027196e+00 | 7.433413e-01 | 6.119264e-01 | 3.985649e-01 | 5.704361e-01 | 3 |
| **max** | 172792.000000 | 2.454930e+00 | 2.205773e+01 | 9.382558e+00 | 1.687534e+01 | 3.480167e+01 | 7.330163e+01 | 1.205895e+02 | 2 |

**8 rows × 31 columns**

In [25]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
)
```

In [131]:

```
c = Counter(y_train)
print(f'отношение классов {c[0] / c[1]}')
```

```
отношение классов 559.0112359550562
```

Out[131]:

```
Counter({0: 199008, 1: 356})
```

In [132]:

```
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred_proba = log_reg.predict_proba(X_test)[:, 1]
y_pred = log_reg.predict(X_test)
```

In [41]:

```
print('Линейная регрессия:')
print(log_loss(y_test, y_pred_proba))
```

```
Линейная регрессия
0.004861663045949433
```

In [49]:

```
y_pred_proba_const = np.zeros(len(X_test))
print('Константное предсказание:')
print(log_loss(y_test, const))
```

```
Константное предсказание:
0.054975522742740056
```

In [48]:

```
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
```

Out[48]:

```
KNeighborsClassifier()
```

In [50]:

```
y_pred_proba_knn = knn.predict(X_test)
```

In [53]:

```
print('KNN:')
print(log_loss(y_test, y_pred_proba_knn))
```

```
KNN:
0.05255013203350157
```

## Нормализация

In [98]:

```
scaler = StandardScaler()
scaler.fit(X)
```

Out[98]:

```
StandardScaler()
```

In [99]:

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test);
```

```
/home/fedor/.local/lib/python3.8/site-packages/sklearn/base.py:445: UserWarning: X does n
ot have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
/home/fedor/.local/lib/python3.8/site-packages/sklearn/base.py:445: UserWarning: X does n
ot have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [100]:

In [100]:

```
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
y_pred_proba = log_reg.predict_proba(X_test)[:, 1]
```

In [101]:

```
print('Линейная регрессия:')
print(log_loss(y_test, y_pred_proba))
```

```
Линейная регрессия:
0.0035431214618405356
```

In [87]:

```
knn.fit(X_train, y_train)
y_pred_proba_knn = knn.predict_proba(X_test)
y_pred_knn = knn.predict(X_test)
```

In [94]:

```
print('KNN:')
print(log_loss(y_test, y_pred_proba_knn))
```

```
KNN:
0.008782587386242045
```

In [111]:

```
print('Accuracy:')
print('Константа')
print(accuracy_score(y_test, y_pred_proba_const))
print('Линейная регрессия')
print(accuracy_score(y_test, y_pred))
print('KNN')
print(accuracy_score(y_test, y_pred_knn))
```

```
Accuracy:
Константа
0.9984082955888721
Линейная регрессия
0.9992626663389628
KNN
0.9994148145547324
```

In [115]:

```
print('PRECISION')
print('Константа')
print(precision_score(y_test, y_pred_proba_const))
print('Линейная регрессия')
print(precision_score(y_test, y_pred))
print('KNN')
print(precision_score(y_test, y_pred_knn))
```

```
PRECISION
Константа
0.0
Линейная регрессия
0.8762886597938144
KNN
0.8583333333333333
```

```
/home/fedor/.local/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [116]:

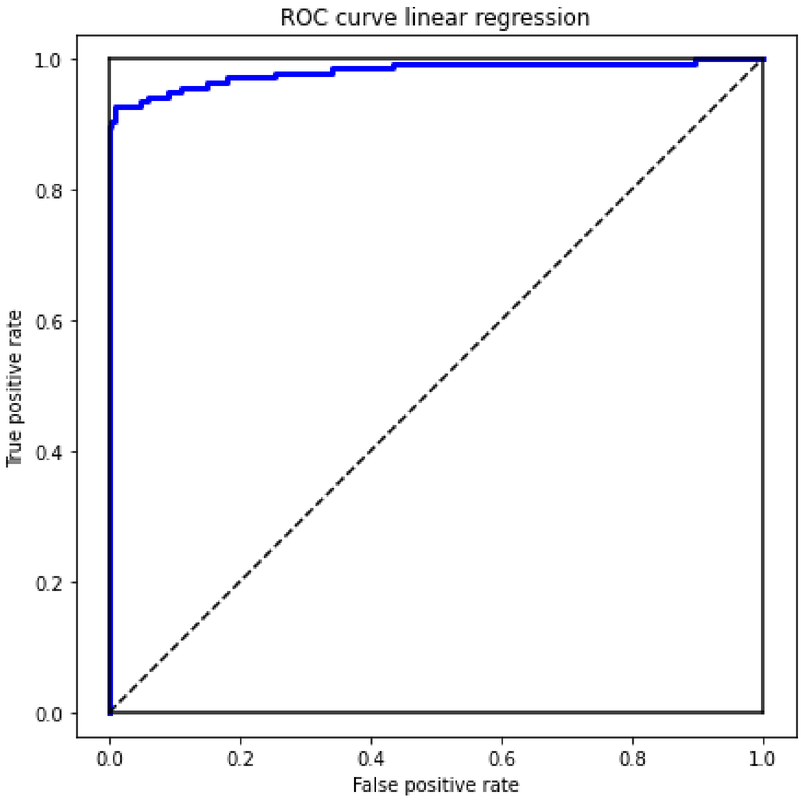
```
print('RECALL')
```

```
print('Константа')
print(recall_score(y_test, y_pred_proba_const))
print('Линейная регрессия')
print(recall_score(y_test, y_pred))
print('KNN')
print(recall_score(y_test, y_pred_knn))
```

```
RECALL
Константа
0.0
Линейная регрессия
0.625
KNN
0.7573529411764706
```

In [121]:

```
plt.figure(figsize=(7, 7))
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.plot(fpr, tpr, 'b', linewidth=3)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot([0, 0], [0, 1], 'k')
plt.plot([1, 1], [0, 1], 'k')
plt.plot([0, 1], [0, 0], 'k')
plt.plot([0, 1], [1, 1], 'k')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.xlim((0, 1))
plt.ylim((0, 1))
plt.axis('equal')
plt.title('ROC curve linear regression')
plt.show()
```




In [124]:

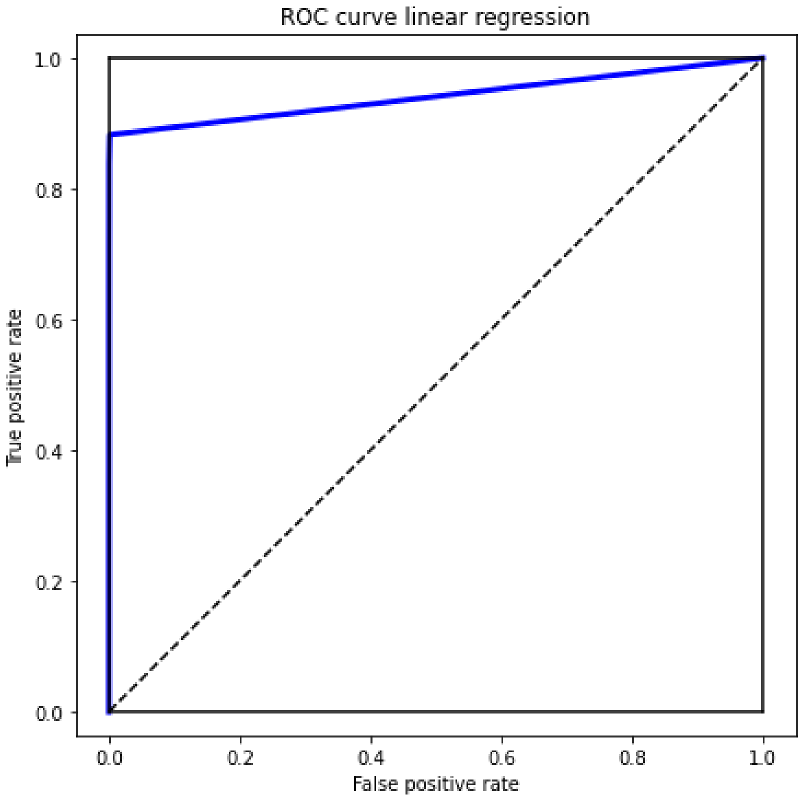
```
print(roc_auc_score(y_test, y_pred_proba))
```

```
0.9807352372296874
```

In [122]:

```
plt.figure(figsize=(7, 7))
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_knn[:, 1])
plt.plot(fpr, tpr, 'b', linewidth=3)
```

```
plt.plot([0, 1], [0, 1], 'k--')
plt.plot([0, 0], [0, 1], 'k')
plt.plot([1, 1], [0, 1], 'k')
plt.plot([0, 1], [0, 0], 'k')
plt.plot([0, 1], [1, 1], 'k')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.xlim((0, 1))
plt.ylim((0, 1))
plt.axis('equal')
plt.title('ROC curve linear regression')
plt.show()
```




In [126]:

```
print(roc_auc_score(y_test, y_pred_proba_knn[:, 1]))
```

0.9410350264339385

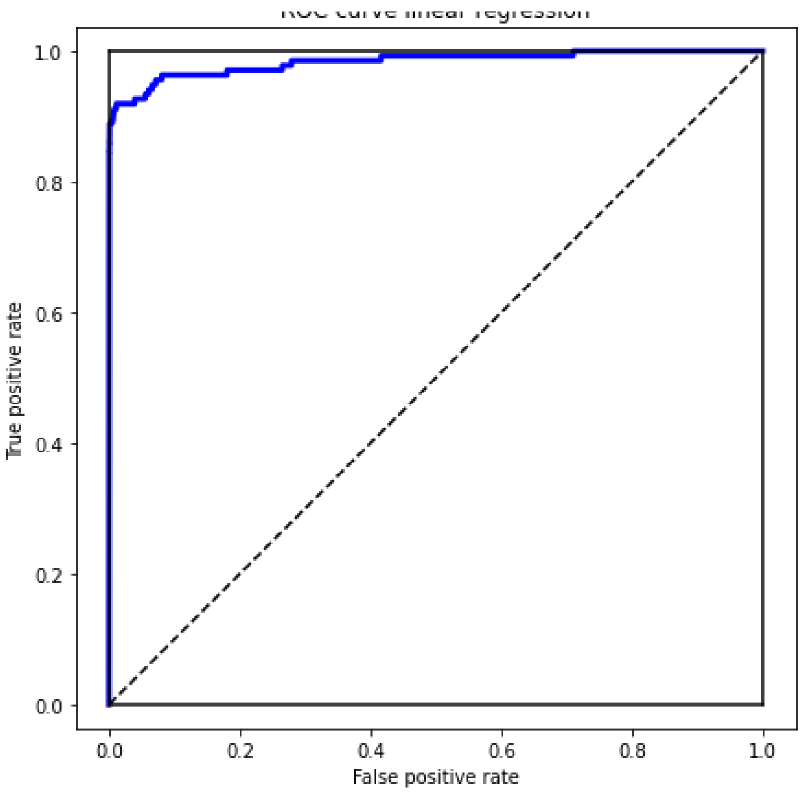
In [197]:

```
log_reg = LogisticRegression(class_weight={0:1, 1:132}, max_iter=1000)
log_reg.fit(X_train, y_train)
log_reg_pred = log_reg.predict(X_test)
log_reg_pred_proba = log_reg.predict_proba(X_test)
```

In [209]:

```
plt.figure(figsize=(7, 7))
fpr, tpr, thresholds = roc_curve(y_test, log_reg_pred_proba[:, 1])
plt.plot(fpr, tpr, 'b', linewidth=3)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot([0, 0], [0, 1], 'k')
plt.plot([1, 1], [0, 1], 'k')
plt.plot([0, 1], [0, 0], 'k')
plt.plot([0, 1], [1, 1], 'k')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.xlim((0, 1))
plt.ylim((0, 1))
plt.axis('equal')
plt.title('ROC curve linear regression')
plt.show()
```

ROC curve linear regression




In [210]:

```
print(roc_auc_score(y_test, log_reg_pred_proba[:, 1]))
```

```
0.9832666652415945
```

In [234]:

```
print(accuracy_score(y_test, log_reg_pred))
```

```
0.9939257750781223
```

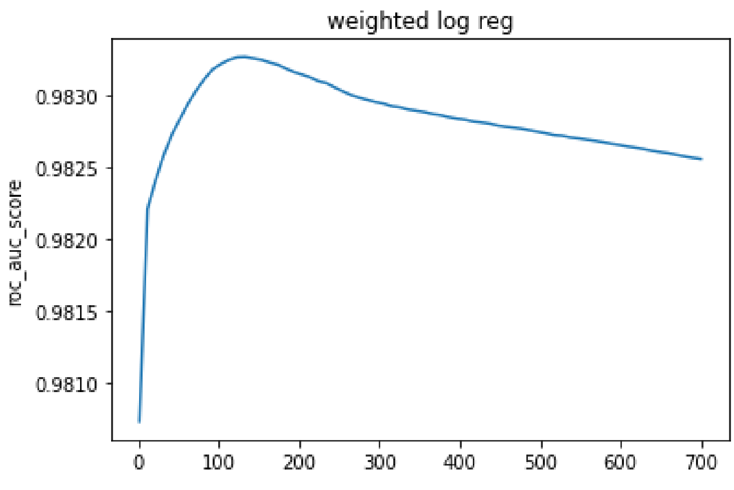
In [192]:

```
w = np.linspace(1, 700, 70)
y = []
for elem in tqdm(w):
    log_reg = LogisticRegression(class_weight={0:1, 1:elem}, max_iter=1000)
    log_reg.fit(X_train, y_train)
    y.append(roc_auc_score(y_test, log_reg.predict_proba(X_test)[:, 1]))
```

```
100%|██████████| 70/70 [05:57<00:00,  5.11s/it]
```

In [195]:

```
plt.plot(w, y);
plt.xlabel('weight of 1')
plt.ylabel('roc_auc_score')
plt.title('weighted log reg');
```



weight of 1

In [212]:

```
print('Лучший вес')
print(w[np.argmax(y)])
print('Лучший скор')
print(np.max(y))
```

```
Лучший вес
132.69565217391303
Лучший скор
0.9832611488333831
```

In [231]:

```
from sklearn.metrics import f1_score

t = np.linspace(0, 1, 50)
y = log_reg_pred_proba[:, 1]

precision = []
recall = []
f1 = []
accuracy = []

for threshold in t:
    ans = y > threshold
    precision.append(precision_score(y_test, ans))
    recall.append(recall_score(y_test, ans))
    f1.append(f1_score(y_test, ans))
    accuracy.append(accuracy_score(y_test, ans))
```

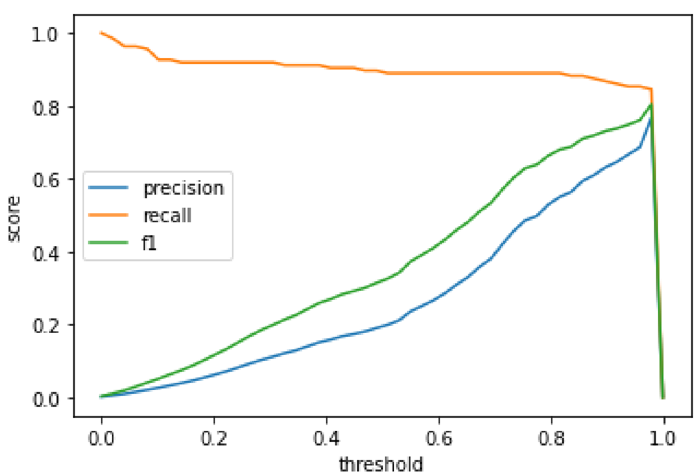
```
/home/fedor/.local/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1308: U
ndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [226]:

```
plt.plot(t, precision)
plt.plot(t, recall)
plt.plot(t, f1)
plt.xlabel('threshold')
plt.ylabel('score')
plt.legend(['precision', 'recall', 'f1'])
```

Out[226]:

```
<matplotlib.legend.Legend at 0x7f6669287af0>
```




In [233]:

```
ind = np.argmax(f1)
print(f'precision = {precision[ind]}')
```

```python
print(f'recall = {recall[ind]}')
print(f'best threshold = {t[ind]}')
print(f'accuracy = {accuracy[ind]}')
```

```
precision = 0.7666666666666667
recall = 0.8455882352941176
best threshold = 0.9795918367346939
accuracy = 0.9993445923013002
```

In [140]:

```python
import seaborn as sns
```

In [141]:

```python
!ls
```

```
 bodyPerformance.csv   inclass_classification.ipynb  'Task 4.ipynb'
 creditcard.csv        pulsar_stars.csv
```

In [142]:

```python
data = pd.read_csv('bodyPerformance.csv')
```

In [143]:

```python
data.head()
```

Out[143]:

| | age | gender | height_cm | weight_kg | body fat_% | diastolic | systolic | gripForce | sit and bend forward_cm | sit-ups counts | broad jump_cm | class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 27.0 | M | 172.3 | 75.24 | 21.3 | 80.0 | 130.0 | 54.9 | 18.4 | 60.0 | 217.0 | C |
| 1 | 25.0 | M | 165.0 | 55.80 | 15.7 | 77.0 | 126.0 | 36.4 | 16.3 | 53.0 | 229.0 | A |
| 2 | 31.0 | M | 179.6 | 78.00 | 20.1 | 92.0 | 152.0 | 44.8 | 12.0 | 49.0 | 181.0 | C |
| 3 | 32.0 | M | 174.5 | 71.10 | 18.4 | 76.0 | 147.0 | 41.4 | 15.2 | 53.0 | 219.0 | B |
| 4 | 28.0 | M | 173.8 | 67.70 | 17.1 | 70.0 | 127.0 | 43.5 | 27.1 | 45.0 | 217.0 | B |

In [144]:

```python
print(len(data))
```

```
13393
```

In [145]:

```python
data.describe()
```

Out[145]:

| | age | height_cm | weight_kg | body fat_% | diastolic | systolic | gripForce | sit and bend forward_cm | |
|---|---|---|---|---|---|---|---|---|---|
| count | 13393.000000 | 13393.000000 | 13393.000000 | 13393.000000 | 13393.000000 | 13393.000000 | 13393.000000 | 13393.000000 | 133 |
| mean | 36.775106 | 168.559807 | 67.447316 | 23.240165 | 78.796842 | 130.234817 | 36.963877 | 15.209268 | |
| std | 13.625639 | 8.426583 | 11.949666 | 7.256844 | 10.742033 | 14.713954 | 10.624864 | 8.456677 | |
| min | 21.000000 | 125.000000 | 26.300000 | 3.000000 | 0.000000 | 0.000000 | 0.000000 | -25.000000 | |
| 25% | 25.000000 | 162.400000 | 58.200000 | 18.000000 | 71.000000 | 120.000000 | 27.500000 | 10.900000 | |
| 50% | 32.000000 | 169.200000 | 67.400000 | 22.800000 | 79.000000 | 130.000000 | 37.900000 | 16.200000 | |
| 75% | 48.000000 | 174.800000 | 75.300000 | 28.000000 | 86.000000 | 141.000000 | 45.200000 | 20.700000 | |
| max | 64.000000 | 193.800000 | 138.100000 | 78.400000 | 156.200000 | 201.000000 | 70.500000 | 213.000000 | |

In [146]:

```
print(data['gender'].value_counts())
```

```
M    8467
F    4926
Name: gender, dtype: int64
```

In [147]:

```
man_col = data['gender'] == 'M'
wom_col = data['gender'] == 'F'
```

In [148]:

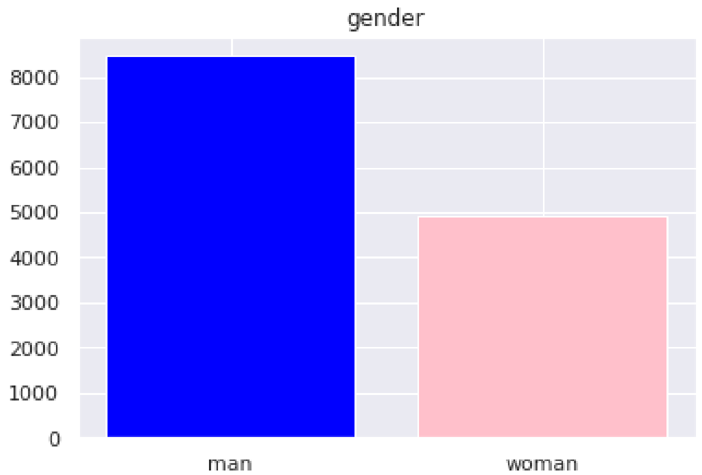
```
data.head()
```

Out[148]:

| | age | gender | height_cm | weight_kg | body fat_% | diastolic | systolic | gripForce | sit and bend forward_cm | sit-ups counts | broad jump_cm | class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 27.0 | M | 172.3 | 75.24 | 21.3 | 80.0 | 130.0 | 54.9 | 18.4 | 60.0 | 217.0 | C |
| 1 | 25.0 | M | 165.0 | 55.80 | 15.7 | 77.0 | 126.0 | 36.4 | 16.3 | 53.0 | 229.0 | A |
| 2 | 31.0 | M | 179.6 | 78.00 | 20.1 | 92.0 | 152.0 | 44.8 | 12.0 | 49.0 | 181.0 | C |
| 3 | 32.0 | M | 174.5 | 71.10 | 18.4 | 76.0 | 147.0 | 41.4 | 15.2 | 53.0 | 219.0 | B |
| 4 | 28.0 | M | 173.8 | 67.70 | 17.1 | 70.0 | 127.0 | 43.5 | 27.1 | 45.0 | 217.0 | B |

In [149]:

```
plt.bar(['man', 'woman'], [man_count, wom_count], color = ('blue', 'pink'))
plt.title('gender');
```




In [150]:

```
data.head()
```

Out[150]:

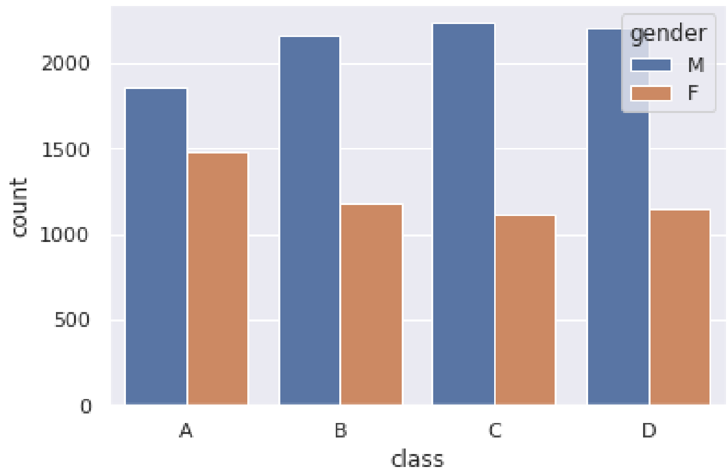
| | age | gender | height_cm | weight_kg | body fat_% | diastolic | systolic | gripForce | sit and bend forward_cm | sit-ups counts | broad jump_cm | class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 27.0 | M | 172.3 | 75.24 | 21.3 | 80.0 | 130.0 | 54.9 | 18.4 | 60.0 | 217.0 | C |
| 1 | 25.0 | M | 165.0 | 55.80 | 15.7 | 77.0 | 126.0 | 36.4 | 16.3 | 53.0 | 229.0 | A |
| 2 | 31.0 | M | 179.6 | 78.00 | 20.1 | 92.0 | 152.0 | 44.8 | 12.0 | 49.0 | 181.0 | C |
| 3 | 32.0 | M | 174.5 | 71.10 | 18.4 | 76.0 | 147.0 | 41.4 | 15.2 | 53.0 | 219.0 | B |
| 4 | 28.0 | M | 173.8 | 67.70 | 17.1 | 70.0 | 127.0 | 43.5 | 27.1 | 45.0 | 217.0 | B |

In [151]:

```python
sns.set_theme(style="darkgrid")
sns.countplot(x='class', hue='gender', order=['A', 'B', 'C', 'D'], data=data)
```

Out[151]:

```
<AxesSubplot:xlabel='class', ylabel='count'>
```




In [152]:

```python
one_hot_class = pd.get_dummies(data['class'])
one_hot.head()
```

Out[152]:

| | F | M |
|---|---|---|
| **0** | 0 | 1 |
| **1** | 0 | 1 |
| **2** | 0 | 1 |
| **3** | 0 | 1 |
| **4** | 0 | 1 |

In [153]:

```python
one_hot_gender = pd.get_dummies(data['gender'])
one_hot.head()
```

Out[153]:

| | F | M |
|---|---|---|
| **0** | 0 | 1 |
| **1** | 0 | 1 |
| **2** | 0 | 1 |
| **3** | 0 | 1 |
| **4** | 0 | 1 |

In [154]:

```python
data.head()
```

Out[154]:

| | age | gender | height_cm | weight_kg | body fat_% | diastolic | systolic | gripForce | sit and bend forward_cm | sit-ups counts | broad jump_cm | class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 27.0 | M | 172.3 | 75.24 | 21.3 | 80.0 | 130.0 | 54.9 | 18.4 | 60.0 | 217.0 | C |
| **1** | 25.0 | M | 165.0 | 55.80 | 15.7 | 77.0 | 126.0 | 36.4 | 16.3 | 53.0 | 229.0 | A |

| | age | gender | height_cm | weight_kg | body fat_% | diastolic | systolic | gripForce | sit and bend forward_cm | sit-ups counts | broad jump_cm | class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 31.0 | M | 179.6 | 78.00 | 20.1 | 92.0 | 152.0 | 44.8 | 12.0 | 49.0 | 181.0 | C |
| 3 | 32.0 | M | 174.5 | 71.10 | 18.4 | 76.0 | 147.0 | 41.4 | 15.2 | 53.0 | 219.0 | B |
| 4 | 28.0 | M | 173.8 | 67.70 | 17.1 | 70.0 | 127.0 | 43.5 | 27.1 | 45.0 | 217.0 | B |

In [191]:

```
x = data.drop(columns=['class'])
y = data['class']
```

In [158]:

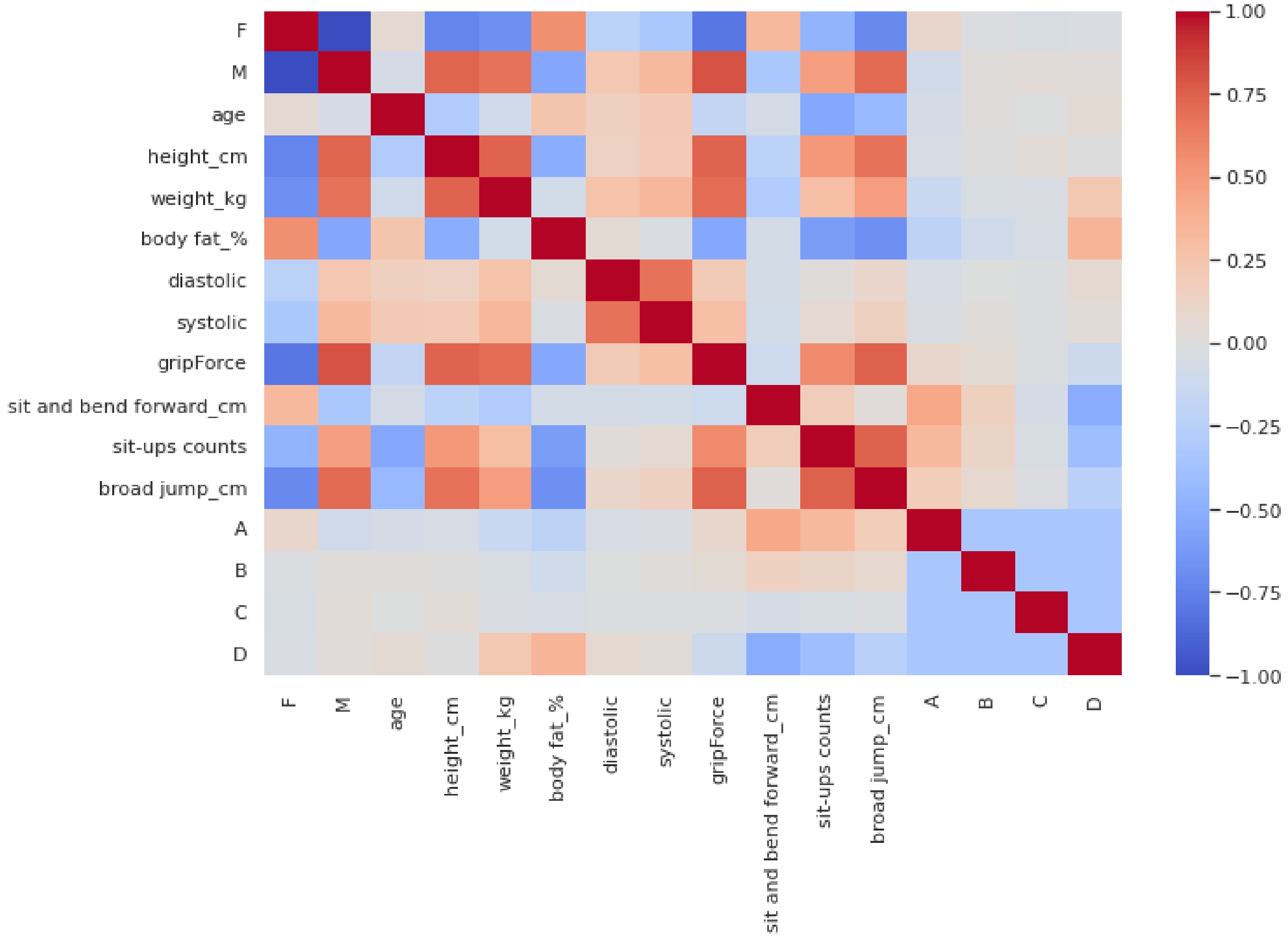
```
X = pd.concat([one_hot_gender, data, one_hot_class], axis=1).drop(columns=['gender', 'cl
ass'])
X.head()
```

Out[158]:

| | F | M | age | height_cm | weight_kg | body fat_% | diastolic | systolic | gripForce | sit and bend forward_cm | sit-ups counts | broad jump_cm | A | B | C | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 27.0 | 172.3 | 75.24 | 21.3 | 80.0 | 130.0 | 54.9 | 18.4 | 60.0 | 217.0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 25.0 | 165.0 | 55.80 | 15.7 | 77.0 | 126.0 | 36.4 | 16.3 | 53.0 | 229.0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 31.0 | 179.6 | 78.00 | 20.1 | 92.0 | 152.0 | 44.8 | 12.0 | 49.0 | 181.0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 1 | 32.0 | 174.5 | 71.10 | 18.4 | 76.0 | 147.0 | 41.4 | 15.2 | 53.0 | 219.0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 28.0 | 173.8 | 67.70 | 17.1 | 70.0 | 127.0 | 43.5 | 27.1 | 45.0 | 217.0 | 0 | 1 | 0 | 0 |

In [169]:

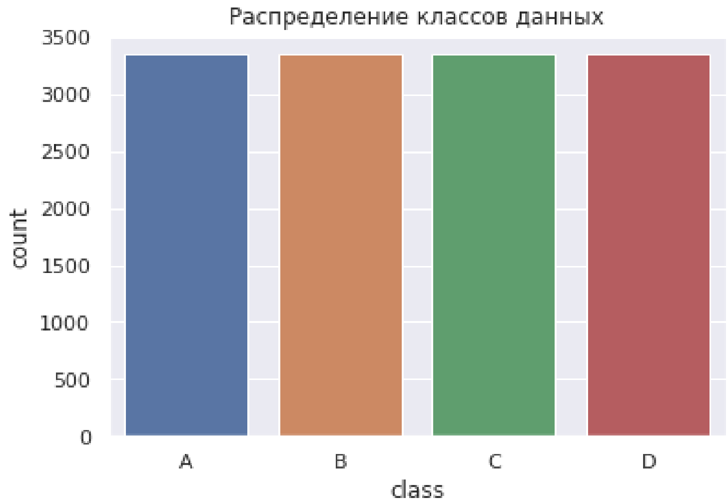
```
plt.figure(figsize=(11, 7))
sns.heatmap(X.corr(), cmap='coolwarm');
```




In [188]:

```
sns.countplot(x='class', data=data, order=['A', 'B', 'C', 'D'])
```

```
plt.title('Распределение классов данных');
```




## Обучение модели

In [207]:

```
X_data = X.drop(columns=['A', 'B', 'C', 'D'])
X_data.head()
```

Out[207]:

| | F | M | age | height_cm | weight_kg | body fat_% | diastolic | systolic | gripForce | sit and bend forward_cm | sit-ups counts | broad jump_cm |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 27.0 | 172.3 | 75.24 | 21.3 | 80.0 | 130.0 | 54.9 | 18.4 | 60.0 | 217.0 |
| **1** | 0 | 1 | 25.0 | 165.0 | 55.80 | 15.7 | 77.0 | 126.0 | 36.4 | 16.3 | 53.0 | 229.0 |
| **2** | 0 | 1 | 31.0 | 179.6 | 78.00 | 20.1 | 92.0 | 152.0 | 44.8 | 12.0 | 49.0 | 181.0 |
| **3** | 0 | 1 | 32.0 | 174.5 | 71.10 | 18.4 | 76.0 | 147.0 | 41.4 | 15.2 | 53.0 | 219.0 |
| **4** | 0 | 1 | 28.0 | 173.8 | 67.70 | 17.1 | 70.0 | 127.0 | 43.5 | 27.1 | 45.0 | 217.0 |

In [211]:

```
X_train, X_test, y_train, y_test = train_test_split(X_data, y, test_size=0.2, random_sta
te=42)
```

In [221]:

```
log_reg = LogisticRegression(max_iter=10000)
log_reg.fit(X_train, y_train)
```

Out[221]:

```
LogisticRegression(max_iter=10000)
```

In [222]:

```
y_pred = log_reg.predict(X_test)
```

In [225]:

```
print("regression accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
regression accuracy:
0.6196341918626354
```

In [226]:

```
knn = KNeighborsClassifier(n_neighbors=5)
```

```
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

In [227]:

```
print('KNN accuracy')
print(accuracy_score(y_test, y_pred_knn))
```

```
KNN accuracy
0.5655095184770437
```

## Нормализация

In [234]:

```
std = StandardScaler()
X_train_std = std.fit_transform(X_train)
X_test_std = std.fit_transform(X_test)
```

**log_reg.fit(X_train, y_train)**

In [237]:

```
log_reg.fit(X_train_std, y_train)
```

Out[237]:

```
LogisticRegression(max_iter=10000)
```

In [238]:

```
y_pred = log_reg.predict(X_test_std)
print("regression accuracy:")
print(accuracy_score(y_test, y_pred))
```

```
regression accuracy:
0.6207540126913027
```

In [239]:

```
knn.fit(X_train_std, y_train)
y_pred_knn = knn.predict(X_test_std)
```

In [240]:

```
print('KNN accuracy')
print(accuracy_score(y_test, y_pred_knn))
```

```
KNN accuracy
0.6061963419186264
```

In [276]:

```
grid = {"C" : np.linspace(0, 2, 10), "penalty" : ["l1", "l2"]}
logreg_cv = GridSearchCV(log_reg, grid, scoring="accuracy")
logreg_cv.fit(X_train_std, y_train)
```

Out[276]:

```
GridSearchCV(estimator=LogisticRegression(max_iter=10000),
             param_grid={'C': array([0.        , 0.22222222, 0.44444444, 0.66666667, 0.8
8888889,
       1.11111111, 1.33333333, 1.55555556, 1.77777778, 2.        ]),
                         'penalty': ['l1', 'l2']},
             scoring='accuracy')
```

In [278]:

```
print(logreg_cv.best_score_)
```

```
print(logreg_cv.best_params_)
```

```
0.6145234762126969
{'C': 0.2222222222222222, 'penalty': 'l2'}
```

In [267]:

```
grid = {"n_neighbors" : np.linspace(15, 30, 15, dtype=int), "weights" : ['uniform', 'dis
tance']}
knn_cv = GridSearchCV(knn, grid, scoring='accuracy')
knn_cv.fit(X_train_std, y_train)
```

Out[267]:

```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25
, 26, 27, 28, 30]),
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

In [268]:

```
print(knn_cv.best_score_)
print(knn_cv.best_params_)
```

```
0.6264691722076916
{'n_neighbors': 25, 'weights': 'distance'}
```

In [275]:

```
knn = KNeighborsClassifier(n_neighbors = 25, weights = 'distance')
knn.fit(X_train_std, y_train)
y_pred_knn = knn.predict(X_test_std)
```

In [280]:

```
log_reg = LogisticRegression(C = 0.22, penalty='l2')
log_reg.fit(X_train_std, y_train)
y_pred_log_reg = log_reg.predict(X_test_std)
```

## Скоры моделей

In [283]:

```
print(accuracy_score(y_test, y_pred_log_reg))
print(accuracy_score(y_test, y_pred_knn))
```

```
0.6196341918626354
0.6312056737588653
```

In [284]:

```
print(precision_score(y_test, y_pred_log_reg, average='macro'))
print(precision_score(y_test, y_pred_knn, average='macro'))
```

```
0.6189677857688459
0.6517613932115671
```

In [285]:

```
print(precision_score(y_test, y_pred_log_reg, average='micro'))
print(precision_score(y_test, y_pred_knn, average='micro'))
```

```
0.6196341918626354
0.6312056737588653
```

In [286]:

```
print(recall_score(y_test, y_pred_log_reg, average='macro'))
print(recall_score(y_test, y_pred_knn, average='macro'))
```

```
0.6173742566475557
0.6291267701416291
```

In [287]:

```python
print(recall_score(y_test, y_pred_log_reg, average='micro'))
print(recall_score(y_test, y_pred_knn, average='micro'))
```

```
0.619634191862635 4
0.6312056737588653
```

In [290]:

```python
print(f1_score(y_test, y_pred_log_reg, average='macro'))
print(f1_score(y_test, y_pred_knn, average='macro'))
```

```
0.6180874875333604
0.6325489171551821
```

In [291]:

```python
print(f1_score(y_test, y_pred_log_reg, average='macro'))
print(f1_score(y_test, y_pred_knn, average='macro'))
```

```
0.6180874875333604
0.6325489171551821
```

In [353]:

```python
def precisionA(y_pred, y_test):
    TP = np.sum((y_pred == 'A') & (y_test == 'A'))
    FP = np.sum((y_pred == 'A') & (y_test != 'A'))
    return TP / (TP + FP)

def recallA(y_pred, y_test):
    TP = np.sum((y_pred == 'A') & (y_test == 'A'))
    FP = np.sum((y_pred == 'A') & (y_test != 'A'))
    FN = np.sum((y_pred != 'A') & (y_test == 'A'))
    return TP / (TP + FN)
def f1score(prec, rec):
    return 2 * prec * rec / (prec + rec)
```

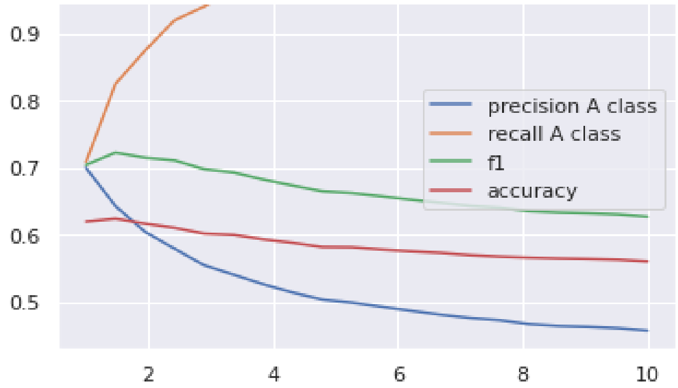
In [354]:

```python
ws = np.linspace(1, 10, 20)
acc = []
precA = []
recA = []
f1 = []
for w in tqdm(ws):
    log_reg = LogisticRegression(class_weight={'A' : w, 'B' : 1, 'C' : 1, 'D' : 1},
                                 C=0.22, penalty='l2', max_iter=10000)
    log_reg.fit(X_train_std, y_train)
    y_pred = log_reg.predict(X_test_std)
    acc.append(accuracy_score(y_pred, y_test))
    precA.append(precisionA(y_pred, y_test))
    recA.append(recallA(y_pred, y_test))
    f1.append(f1score(precisionA(y_pred, y_test), recallA(y_pred, y_test)))
```

```
100%|██████████| 20/20 [00:06<00:00,  3.23it/s]
```

In [359]:

```python
plt.plot(ws, precA)
plt.plot(ws, recA)
plt.plot(ws, f1)
plt.plot(ws, acc)
plt.legend(['precision A class', 'recall A class', 'f1', 'accuracy']);
```




In [366]:

```
ind = 3
w = ws[ind]
print('optimal weight:')
print(w)
print('precision on A class')
print(precA[ind])
print('recall on A class')
print(recA[ind])
print('f1 on A class')
print(f1[ind])
print('accuracy all dataset')
print(acc[ind])
```

```
optimal weight:
2.4210526315789473
precision on A class
0.5790441176470589
recall on A class
0.9197080291970803
f1 on A class
0.7106598984771574
accuracy all dataset
0.6103023516237402
```