-- 1. возраст
-- 2. секс
-- 3. тип боли в груди (4 значения)
-- 4. кровяное давление в состоянии покоя
-- 5. уровень холестерина в сыворотке крови в мг/дл
-- 6. уровень сахара в крови натощак > 120 мг/дл
-- 7. результаты электрокардиографии в состоянии покоя (значения 0,1,2)
-- 8. достигнутая максимальная частота сердечных сокращений
-- 9. стенокардия, вызванная физическими упражнениями
-- 10. oldpeak = депрессия ST, вызванная физической нагрузкой по отношению к отдыху
-- 11. наклон пика упражнения сегмента ST
-- 12. количество крупных сосудов (0-3), окрашенных флуороскопией
-- 13. Отсутствие (1) или наличие (2) сердечных заболеваний

Расмотрим пример на датасете из репозитория UCI

Описание данных - https://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29 (https://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29)

In [31]:

```
import pandas as pd
import numpy as np
data = pd.read_csv("heart.dat", header=None, delimiter = '.')
data.head(3)
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	70	0 1	0 4	0 130	0 322	0 0	0 2	0 109	0 0	0 2	4 2	0 3	0 3	0 2
1	67	0 0	0 3	0 115	0 564	0 0	0 2	0 160	0 0	0 1	6 2	0 0	0 7	0 1
2	57	0 1	0 2	0 124	0 261	0 0	0 0	0 141	0 0	0 0	3 1	0 0	0 7	0 2

In [32]:

```
data = data.replace(' ', '', regex= True)
```

In [33]:

```
data = data.apply(pd.to_numeric)
```

In [35]:

```
map_13 = { 2: 0, 1: 1}
data[13] = data[13].map(map_13)
```

In [36]:

```
data.head()
```

Out[36]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	70	1	4	130	322	0	2	109	0	2	42	3	3	0
1	67	0	3	115	564	0	2	160	0	1	62	0	7	1
2	57	1	2	124	261	0	0	141	0	0	31	0	7	0
3	64	1	4	128	263	0	0	105	1	0	22	1	7	1
4	74	0	2	120	269	0	2	121	1	0	21	1	3	1

У нас есть 13 признаков и 1 целевая переменная (бинарная) - нужно определить есть проблеммы с сердцем или нет

In [37]:

```
print(data.shape)
```

```
(270, 14)
```

Всего 270 пациентов

Посмотрим на соотношение классов

In [38]:

```
data.iloc[:, -1].value_counts()
```

Out[38]:

```
1    150
0    120
Name: 13, dtype: int64
```

Разбиваем выборку на тренировочную и тестовую части и обучаем модель (в примере - градиентный бустинг)

In [39]:

```
from sklearn.model_selection import train_test_split

x_data = data.iloc[:,:-1]
y_data = data.iloc[:,-1]

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2, random_s
```

In [42]:

```
import xgboost as xgb

model = xgb.XGBClassifier(use_label_encoder=False, eval_metric= 'logloss')

model.fit(x_train, y_train)
y_predict = model.predict(x_test)
```

Проверяем качество

In [43]:

```
from sklearn.metrics import recall_score, precision_score, roc_auc_score, accuracy_score, f

def evaluate_results(y_test, y_predict):
    print('Classification results:')
    f1 = f1_score(y_test, y_predict)
    print("f1: %.2f%%" % (f1 * 100.0))
    roc = roc_auc_score(y_test, y_predict)
    print("roc: %.2f%%" % (roc * 100.0))
    rec = recall_score(y_test, y_predict, average='binary')
    print("recall: %.2f%%" % (rec * 100.0))
    prc = precision_score(y_test, y_predict, average='binary')
    print("precision: %.2f%%" % (prc * 100.0))


evaluate_results(y_test, y_predict)
```

```
Classification results:
f1: 79.37%
roc: 76.76%
recall: 73.53%
precision: 86.21%
```

Теперь очередь за PU learning

Представим, что нам неизвестны негативы и часть позитивов

In [121]:

```
div = 0.7
mod_data = data.copy()
#get the indices of the positives samples
pos_ind = np.where(mod_data.iloc[:,-1].values == 1)[0]
#shuffle them
np.random.shuffle(pos_ind)
# leave just 25% of the positives marked
pos_sample_len = int(np.ceil(div * len(pos_ind)))
print(f'Using {pos_sample_len}/{len(pos_ind)} as positives and unlabeling the rest')
pos_sample = pos_ind[:pos_sample_len]
```

```
Using 105/150 as positives and unlabeling the rest
```

Создаем столбец для новой целевой переменной, где у нас два класса - P (1) и U (-1)

In [122]:

```
mod_data['class_test'] = -1
mod_data.loc[pos_sample,'class_test'] = 1
print('target variable:\n', mod_data.iloc[:,-1].value_counts())
```

```
target variable:
 -1    165
 1    105
Name: class_test, dtype: int64
```

- We now have just 153 positive samples labeled as 1 in the 'class_test' col while the rest is unlabeled as -1.

- Recall that col 4 still holds the actual label

In [123]:

```
mod_data.head(10)
```

Out[123]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	class_test
0	70	1	4	130	322	0	2	109	0	2	42	3	3	0	-1
1	67	0	3	115	564	0	2	160	0	1	62	0	7	1	1
2	57	1	2	124	261	0	0	141	0	0	31	0	7	0	-1
3	64	1	4	128	263	0	0	105	1	0	22	1	7	1	-1
4	74	0	2	120	269	0	2	121	1	0	21	1	3	1	1
5	65	1	4	120	177	0	0	140	0	0	41	0	7	1	1
6	56	1	3	130	256	1	2	142	1	0	62	1	6	0	-1
7	59	1	4	110	239	0	2	142	1	1	22	1	7	0	-1
8	60	1	4	140	293	0	2	170	0	1	22	2	7	0	-1
9	63	0	4	150	407	0	2	154	0	4	2	3	7	0	-1

Remember that this data frame (x_data) includes the former target variable that we keep here just to compare the results

[:-2] is the original class label for positive and negative data [:-1] is the new class for positive and unlabeled data

In [124]:

```
x_data = mod_data.iloc[:,:-2].values # just the X
y_labeled = mod_data.iloc[:,-1].values # new class (just the P & U)
y_positive = mod_data.iloc[:,-2].values # original class
```

1. random negative sampling

In [125]:

```
mod_data = mod_data.sample(frac=1)
neg_sample = mod_data[mod_data['class_test']==-1][:len(mod_data[mod_data['class_test']==1])
sample_test = mod_data[mod_data['class_test']==-1][len(mod_data[mod_data['class_test']==1])
pos_sample = mod_data[mod_data['class_test']==1]
print(neg_sample.shape, pos_sample.shape)
sample_train = pd.concat([neg_sample, pos_sample]).sample(frac=1)
```

```
(105, 15) (105, 15)
```

In [126]:

```
model = xgb.XGBClassifier(use_label_encoder=False, eval_metric = 'logloss')

model.fit(sample_train.iloc[:,:-2].values,
          sample_train.iloc[:,-2].values)
y_predict = model.predict(sample_test.iloc[:,:-2].values)
evaluate_results(sample_test.iloc[:,-2].values, y_predict)
```

```
Classification results:
f1: 70.59%
roc: 80.30%
recall: 94.74%
precision: 56.25%
```

In [127]:

```
f1 = f1_score(sample_test.iloc[:,-2].values, y_predict)
round(f1 * 100, 2)
```

Out[127]:

```
70.59
```

Таблица изменение значения Доли P

In [129]:

```
f1 = f1_score(sample_test.iloc[:,-2].values, y_predict)
roc = roc_auc_score(sample_test.iloc[:,-2].values, y_predict)
rec = recall_score(sample_test.iloc[:,-2].values, y_predict, average='binary')
prc = precision_score(sample_test.iloc[:,-2].values, y_predict, average='binary')



# res_table = pd.DataFrame(index = ['f1', 'roc', 'recall', 'precision'])
res_table[div] = [round(f1 * 100, 2), round(roc * 100, 2), round(rec * 100, 2), round(prc *
res_table
```

Out[129]:

	0.10	0.25	0.50	0.70
f1	76.92	79.44	75.68	70.59
roc	71.63	77.43	80.11	80.30
recall	86.61	88.54	91.30	94.74
precision	69.18	72.03	64.62	56.25

Домашнее задание

1. взять любой набор данных для бинарной классификации (можно скачать один из модельных с https://archive.ics.uci.edu/ml/datasets.php (https://archive.ics.uci.edu/ml/datasets.php))
2. сделать feature engineering
3. обучить любой классификатор (какой вам нравится)
4. далее разделить ваш набор данных на два множества: P (positives) и U (unlabeled). Причем брать нужно не все положительные (класс 1) примеры, а только лишь часть
5. применить random negative sampling для построения классификатора в новых условиях
6. сравнить качество с решением из пункта 4 (построить отчет - таблицу метрик)
7. поэкспериментировать с долей P на шаге 5 (как будет меняться качество модели при уменьшении/увеличении размера P)

Бонусный вопрос:

Как вы думаете, какой из методов на практике является более предпочтительным: random negative sampling или 2-step approach?

Ваш ответ здесь:

In [ ]:

```
2-step approach
```