Московский авиационный институт

(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Лабораторная работа №1 по искусственному интеллекту

6 семестр

Студент: Катермин Всеволод Сергеевич

Группа: М8О-308Б-18

Руководитель: Ахмед Самир Халид

Дата: 20.06.2021

Москва, 2021

Оглавление

Постановка задачи:

Найти себе набор данных (датасет), для следующей лабораторной работы, и проанализировать его. Выявить проблемы набора данных, устранить их. Визуализировать зависимости, показать распределения некоторых признаков. Реализовать алгоритмы К ближайших соседа с использованием весов и Наивный Байесовский классификатор и сравнить с реализацией библиотеки sklearn.

Датасет Titanic

Титаник — известная задача на Kaggle, ориентированная в большей мере на начинающих в машинном обучении. Датасет Титаник содержит данные пассажиров корабля. Цель задачи — построить модель, которая лучшим образом сможет предсказать, остался ли произвольный пассажир в живых или нет.

- Passengerid - это id пассажира.
- Survived - столбец, говорит о том выжил ли пассажир, принимает значение 0 или 1 . 1 если выжил , 0 если нет. Является целивой перменной, которую будем предсказывать. Только 342 человека выжило
- Pclass - фича, говорящая о том ,к какому классу в корабле относится пассажир. Может принимать значения 1,2,3. В основном люди 3 класса
- Name- имя пассажира.
- Sex - пол пассажира. Мужчин больше чем женщин
- Age - возраст пассажира. 5 самых распрастараненных возрастов 24, 22, 18, 18 , 30. В этом столбце есть пропуски.
- SibSp — содержит информацию о количестве родственников 2-го порядка (муж, жена, братья, сетры).Пропусков нет. Большинство пассажиров не имеют родственников. 209 имеют только одного родственника
- Parch-содержит информацию о количестве родственников на борту 1-го порядка (мать, отец, дети). У большинства людей нет на борту родственников первого порядка
- Ticket- код билета пассажира
- Fare - цена билета. Пропусков в данных нету. 5 самые частые цен на билет 8.0500, 3.0000, 7.8958, 7.7500, 26.0000.
- Cabin- вид каюты пассажира. Пропусков данных очень много, порядка 77%.
- Embarked- порт посадки. Есть немного пропусков. Возможные значения S,C,Q, где C — Cherbourg, Q — Queenstown, S — Southampton. Самый частый порт посадки S — Southampton

Загрузка датасета и первичный анализ

In [2]:
```
dataset=read_csv('train.csv')
```

Выводим первые строки

In [3]:
```
dataset.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Смотрим размерность

In [4]:
```
dataset.shape
```

Out[4]:
```
(891, 12)
```

Выводим информацию и описание датасета

In [5]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [6]:
```
dataset.describe()
```

Out[6]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Смотрим частотности некотрых столбцов и проверяем на наличие нулевых/пустых значений

In [7]:
```
#сколько выжило , сколько умерло
dataset['Survived'].value_counts()
```

Out[7]:
```
0    549
1    342
Name: Survived, dtype: int64
```

In [8]:
```
#смотрим частотность каждого класса в карабле
dataset['Pclass'].value_counts()
```

Out[8]:
```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

In [9]:
```
#смотрим частотность полов
dataset['Sex'].value_counts()
```

Out[9]:
```
male      577
female    314
Name: Sex, dtype: int64
```

In [10]:
```
#частотность возрастов пассажиров
dataset['Age'].value_counts()
```

Out[10]:
```
24.00    30
22.00    27
18.00    26
19.00    25
30.00    25
         ..
55.50     1
70.50     1
66.00     1
23.50     1
0.42      1
Name: Age, Length: 88, dtype: int64
```

In [11]:
```
#проверяем на наличие пропусков в столбце , если значение будет отлично от 0, значит в столбце есть пропуски
(dataset['Age'].isnull()).mean()
```

Out[11]:
```
0.19865319865319866
```

In [12]:
```
#проверяем на наличие пропусков в столбце , если значение будет отлично от 0, значит в столбце есть пропуски
(dataset['SibSp'].isnull()).mean()
```

Out[12]:
```
0.0
```

In [13]:
```
#смотрим частотность
dataset['SibSp'].value_counts()
```

Out[13]:
```
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

In [14]:
```
#проверяем на наличие пропусков в столбце , если значение будет отлично от 0, значит в столбце есть пропуски
(dataset['Parch'].isnull()).mean()
```

Out[14]:
```
0.0
```

In [15]:
```
#смотрим частотность
dataset['Parch'].value_counts()
```

Out[15]:
```
0    678
1    118
2     80
5      5
3      5
4      4
6      1
Name: Parch, dtype: int64
```

In [16]:
```
#проверяем на наличие пропусков в столбце , если значение будет отлично от 0, значит в столбце есть пропуски
(dataset['Fare'].isnull()).mean()
```

Out[16]:
```
0.0
```

In [17]:
```
#смотрим частотность
dataset['Fare'].value_counts()
```

Out[17]:
```
8.0500     43
13.0000    42
7.8958     38
7.7500     34
26.0000    31
           ..
8.4583      1
9.8375      1
8.3625      1
14.1083     1
17.4000     1
Name: Fare, Length: 248, dtype: int64
```

```
In [18]: #проверяем на наличие пропусков в столбце , если значение будет отлично от 0, значит в столбце есть пропуски
         (dataset['Cabin'].isnull()).mean()

Out[18]: 0.7710437710437711

In [19]: #смотрим частотность
         dataset['Cabin'].value_counts()

Out[19]: C23 C25 C27    4
         B96 B98        4
         G6             4
         D              3
         F33            3
                       ..
         C47            1
         E36            1
         B38            1
         D10 D12        1
         B39            1
         Name: Cabin, Length: 147, dtype: int64

In [20]: #проверяем на наличие пропусков в столбце , если значение будет отлично от 0, значит в столбце есть пропуски
         (dataset['Embarked'].isnull()).mean()

Out[20]: 0.002244668911335578

In [21]: #смотрим частотность
         dataset['Embarked'].value_counts()

Out[21]: S    644
         C    168
         Q     77
         Name: Embarked, dtype: int64
```

Выводы из первичного анализа данных

- Passengerid удаляем, потому что у датасета и так есть индексы, а id пассажира не дает нам никакой дополнительной полезной информации
- Name удаляем. Т к это категориальная переменная , где количество категорий равно количеству пассажиров. Бесполезная фича
- Sex принимает значения male и female. их Мы переведем в такой вид. male -1 , female-0
- Age в пропусках заполним через среднее.
- Ticket удаляем , по той же причине, что и Name
- Cabin может быть преобразована в логическое значение
- Embarked - порт посадки пропуски заполню через самое часто встречаемое значение

Преобразования признаков

In [22]:
```
#удаляем PassengerId, Name, Ticket
dataset=dataset.drop(['PassengerId','Name','Ticket'],axis=1)
dataset.head()
```

Out[22]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	NaN	S
1	1	1	female	38.0	1	0	71.2833	C85	C
2	1	3	female	26.0	0	0	7.9250	NaN	S
3	1	1	female	35.0	1	0	53.1000	C123	S
4	0	3	male	35.0	0	0	8.0500	NaN	S

In [23]:
```
#функция преобразующая пол
def priobr_sex(sex):
    if sex=='male':
        return 1
    elif sex=='female':
        return 0

dataset['Sex'] = dataset['Sex'].apply(priobr_sex)
dataset.head()
```

Out[23]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	1	22.0	1	0	7.2500	NaN	S
1	1	1	0	38.0	1	0	71.2833	C85	C
2	1	3	0	26.0	0	0	7.9250	NaN	S
3	1	1	0	35.0	1	0	53.1000	C123	S
4	0	3	1	35.0	0	0	8.0500	NaN	S

In [24]:
```
#Заполняем возраст через среднее
dataset['Age'] = dataset['Age'].fillna(dataset['Age'].mean())
(dataset['Age'].isnull()).mean()
```

Out[24]:
```
0.0
```

In [25]:
```
#приобразуем Cabin
dataset['Cabin'] =dataset['Cabin'].fillna(0)
dataset['Cabin'][dataset['Cabin']!=0] = 1
dataset.head()
```

```
<ipython-input-25-d6439218d592>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataset['Cabin'][dataset['Cabin']!=0] = 1
```

Out[25]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	1	22.0	1	0	7.2500	0	S
1	1	1	0	38.0	1	0	71.2833	1	C
2	1	3	0	26.0	0	0	7.9250	0	S
3	1	1	0	35.0	1	0	53.1000	1	S
4	0	3	1	35.0	0	0	8.0500	0	S

In [26]:
```
#функция приобразующая S,C,Q в 0,1,2
def priobr_cabin(cabin):
    if cabin == 'S':
        return 0
    elif cabin == 'C':
        return 1
    elif cabin == 'Q':
        return 2
dataset['Embarked'] = dataset['Embarked'].fillna('S')
dataset['Embarked'] = dataset['Embarked'].apply(priobr_cabin)
dataset.head()
```

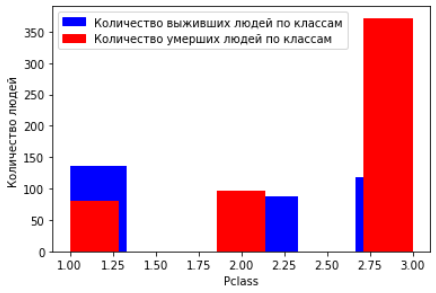
Out[26]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	1	22.0	1	0	7.2500	0	0
1	1	1	0	38.0	1	0	71.2833	1	1
2	1	3	0	26.0	0	0	7.9250	0	0
3	1	1	0	35.0	1	0	53.1000	1	0
4	0	3	1	35.0	0	0	8.0500	0	0

Визуализация распределений

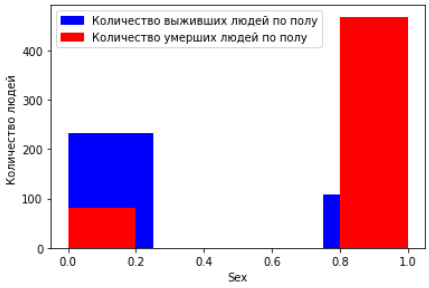
In [27]:
```
#Визуализация выживших и умерших по соц. положению
plt.xlabel('Pclass')
plt.ylabel('Количество людей ')
plt.hist(x=dataset[dataset['Survived']==1]['Pclass'], bins=6,color='blue')
plt.hist(x=dataset[dataset['Survived']==0]['Pclass'], bins=7,color='red')
life = mpatches.Patch(color='blue', label='Количество выживших людей по классам')
unlife = mpatches.Patch(color='red', label='Количество умерших людей по классам')
plt.legend(handles=[life,unlife])
```

Out[27]: <matplotlib.legend.Legend at 0xbbb0280>



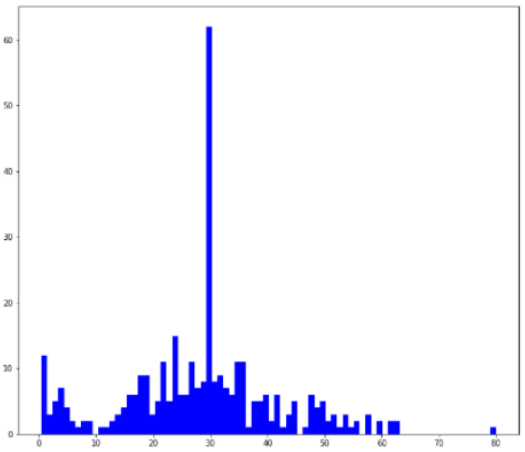
In [28]:
```
#Распределение выживших и у умерших по полу
plt.xlabel('Sex')
plt.ylabel('Количество людей ')
plt.hist(x=dataset[dataset['Survived']==1]['Sex'], bins=4,color='blue')
plt.hist(x=dataset[dataset['Survived']==0]['Sex'], bins=5,color='red')
life = mpatches.Patch(color='blue', label='Количество выживших людей по полу')
unlife = mpatches.Patch(color='red', label='Количество умерших людей по полу')
plt.legend(handles=[life,unlife])
```

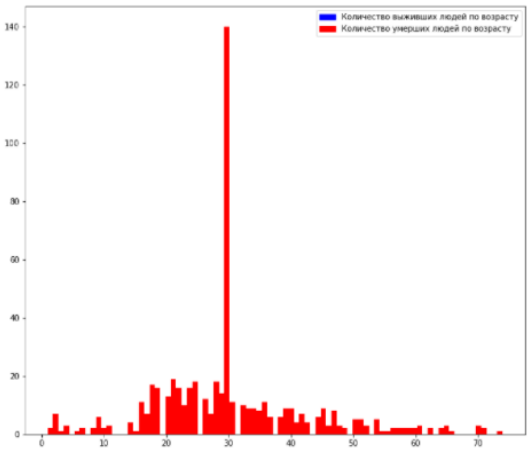
Out[28]: <matplotlib.legend.Legend at 0xbbc9550>



In [29]:
```
#Распределение выживших и у умерших по возрастам
fig = plt.figure(figsize=(25,10))
plt.xlabel('Age')
plt.ylabel('Количество людей ')
plt.subplot(1, 2, 1)
plt.hist(x=dataset[dataset['Survived']==1]['Age'], bins=80,color='blue')
plt.subplot(1, 2, 2)
plt.hist(x=dataset[dataset['Survived']==0]['Age'], bins=85,color='red')
life = mpatches.Patch(color='blue', label='Количество выживших людей по возрасту')
unlife = mpatches.Patch(color='red', label='Количество умерших людей по возрасту')
plt.legend(handles=[life,unlife])
```

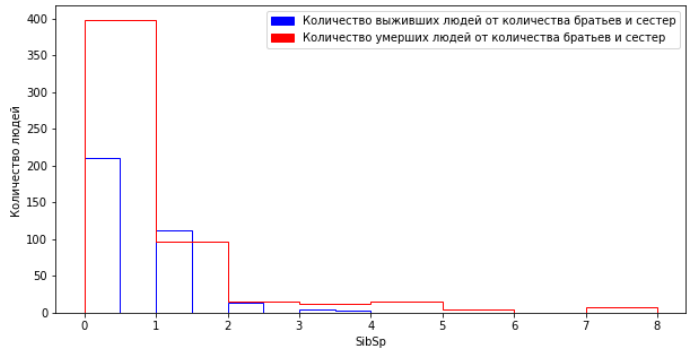
Out[29]: <matplotlib.legend.Legend at 0xbc53340>





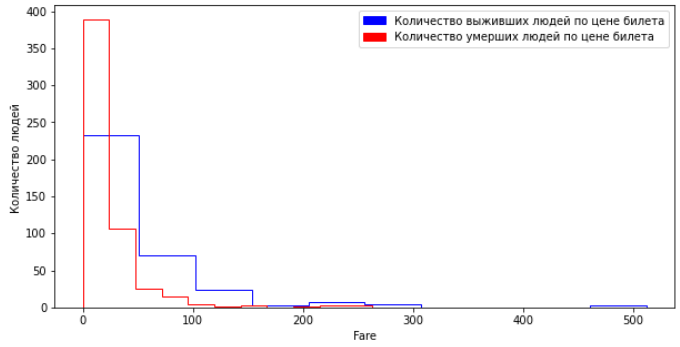
In [30]:
```
# Распределение выживших и умерших от количества братьев и сестер
fig = plt.figure(figsize=(10,5))
plt.xlabel('SibSp')
plt.ylabel('Количество людей ')
plt.hist(x=dataset[dataset['Survived']==1]['SibSp'], bins=8,color='blue',histtype='step')
plt.hist(x=dataset[dataset['Survived']==0]['SibSp'], bins=8,color='red',histtype='step')
life = mpatches.Patch(color='blue', label='Количество выживших людей от количества братьев и сестер')
unlife = mpatches.Patch(color='red', label='Количество умерших людей от количества братьев и сестер')
plt.legend(handles=[life,unlife])
```

Out[30]: <matplotlib.legend.Legend at 0xbdff6a0>



In [31]:
```
# Распределение выживших и умерших по цене билета Fare
fig = plt.figure(figsize=(10,5))
plt.xlabel('Fare')
plt.ylabel('Количество людей ')
plt.hist(x=dataset[dataset['Survived']==1]['Fare'], bins=10,color='blue',histtype='step')
plt.hist(x=dataset[dataset['Survived']==0]['Fare'], bins=11,color='red',histtype='step')
life = mpatches.Patch(color='blue', label='Количество выживших людей по цене билета')
unlife = mpatches.Patch(color='red', label='Количество умерших людей по цене билета')
plt.legend(handles=[life,unlife])
```

Out[31]: <matplotlib.legend.Legend at 0xc457790>



Выводы по визуализации

- По первому графику можно сделать вывод, что люди из 3-ого класса умирают чаще всех остальных. Выживают же чаще всех люди из первого класса. Люди из второго класса могут с примерно равной вероятностью выжить или умереть
- По второму графику можно сделать вывод, что мужчины значительно чаще умирали на титанике чем женщины. При том женщины выживали примерно в 2 раза чаще чем умирали
- По третьему графику можно сделать вывод, что количество выживших и умерших людей примерно 30 лет больше всех остальных.
- По четвертому графику можно сделать вывод, что если у пассажира нету братьев или сестер, то он с большей вероятностью погибнет
- По пятому графику можно сделать вывод, что больше всех умирают люди с билетами до примерно 100 долларов. Также можно сказать, что люди чья стоимость билетов находится около 500 долларов не умирают.

Алгоритма k ближайших соседей с весами

Алгоритм

Для классификации каждого из объектов тестовой выборки необходимо последовательно выполнить следующие операции:

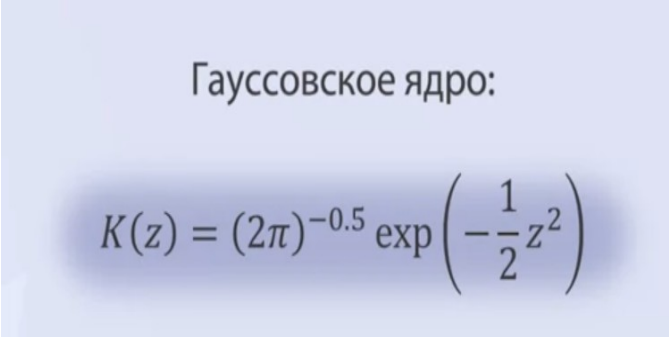
1) Вычислить расстояние до каждого из объектов обучающей выборки и посчитать веса для каждого объекта обучающей выборки
2) Отобрать k объектов обучающей выборки, расстояние до которых минимально
3) Считать количество каждого класса учитывая веса.
4) Самый частый класс учитывая веса является классом классифицируемого объекта

Реализация

Метод k-ближайших соседей реализован в виде класса Knn с 2 публичными методами fit и predict. fit -нужен для обучения. predict для предсказания.

Приватный метод _jadro_K нужен для подсчета Гауссовского ядра в Парзеновском окне для расчета весов.

В конструктор класса при инициализации можно передать параметр k-количество соседей и параметр h-ширина окна



```
class KNN:
    #при инициализации в конструктор передается 2 параметра k и h. k- количство соседей , h -длинна парзеновского окна
    def __init__(self, k,h=1):
        self.h=h
        self.k = k

    #получаю обучающую выборку
    def fit(self, X, y):
        self.X_train = X
        self.y_train = y


    #метод _jadro_K является приватным. Он реализует вычисление Гауссовского ядра,используемового в парзеновском окне
    def _jadro_K(self,z):

        return ((2*pi)**(-0.5))*exp(-0.5*z**(2))
    #метод predict классифицирует объекты
    def predict(self, X_test):
        output = []#предсказанные метки
        for i in range(len(X_test)):
            d = []#растояния между тестовым объектом и объектами обучающей выборки
            votes = []#метки ближайших объектов
            for j in range(len(X_train)):
                #считаем растояние
                dist = scipy.spatial.distance.euclidean(X_train.iloc[j] , X_test.iloc[i])
                 #считается расстояние и применяется парзеновское окно с гауссовским ядром
                #для реализации весов. Чем меньше h , тем меньше мы учитываем далекие объекты
                weight=self._jadro_K(scipy.spatial.distance.euclidean(X_train.iloc[j] , X_test.iloc[i])/self.h)
                d.append([dist, j,weight])

            #сортируем расстояния до тестового объекта
            d.sort()
            d = d[0:self.k]#берем k ближайших тестовых объектов
            zero_score=0
            one_score=0
            #достаем метки
            for a, j, k in d:
                votes.append(y_train.iloc[j])
            #считаем частотность 1 и 0 учитывая веса
            for j in range(len(votes)):
                if votes[j]==0:
                    zero_score=zero_score+1*d[j][2]
                if votes[j]==1:
                    one_score=one_score+1*d[j][2]
            #смотрим какого класса больше
            if zero_score>one_score:
                ans=0
            if  one_score>zero_score:
                ans=1
            output.append(ans)
        return output
```

Обучение и метрики

In [38]:
```
%%time
#обучаю свою модель с k=5 и h=5 . Что значит,смотреть по 5  соседям и использовать парзеновское окно равное 5
my_KNN = KNN(5,5)
my_KNN.fit(X_train, y_train)
#делаем предсказания на трейне и на тесте и смотрим метрики
print('Метрики на обучающей выборки ')
metrics(my_KNN.predict(X_train),y_train)
print('Метрики на тестовой выборки ')
metrics(my_KNN.predict(X_test),y_test)
```

```
Метрики на обучающей выборки
Accuracy:  0.8057784911717496
Pprecision:  0.8057784911717496
Recall:  0.8057784911717496
F1:  0.8057784911717496
Метрики на тестовой выборки
Accuracy:  0.7052238805970149
Pprecision:  0.7052238805970149
Recall:  0.7052238805970149
F1:  0.7052238805970149
Wall time: 7min 47s
```

In [39]:
```
%%time
#обучаем моледь из sklearn
sk_knn=KNeighborsClassifier(n_neighbors=5,weights='distance')
sk_knn.fit(X_train,y_train)
#делаем предсказания на трейне и на тесте и смотрим метрики
print('Метрики на обучающей выборки ')
metrics(sk_knn.predict(X_train),y_train)
print('Метрики на тестовой выборки ')
metrics(sk_knn.predict(X_test),y_test)
```

```
Метрики на обучающей выборки
Accuracy:  0.9823434991974318
Pprecision:  0.9823434991974318
Recall:  0.9823434991974318
F1:  0.9823434991974318
Метрики на тестовой выборки
Accuracy:  0.7164179104477612
Pprecision:  0.7164179104477612
Recall:  0.7164179104477612
F1:  0.7164179104477613
Wall time: 27 ms
```

Выводы по метрикам

- Разница метрик на тесте между моей моделью и моделью из sklearn минимальна
- Сильного переобучения не наблюдается у обоих моделей. Тем не менее модель из sklearn более склона к переобучению чем моя , потому что разница метрик на трейне и тесте модели из sklearn в 3 раза больше чем разница метрик на трейни и тесте моей модели. Это разница в переобучаемости вызвана на мой взгляд тем, как считаются веса в модели. Моя модель использует парзеновское окно с гауссовским ядром для расчета веса, а модель из skleark использует формулу w_i=1/p(x,x_i), где p-функция расчета расстояния.
- Также моя модель обучается значительно дольше чем модель из sklearn. На мой взгляд это обусловленно тем, что моя модель на питоне написана, а модель из sklearn написано на c/c++.

Наивный байесовский классификатор

Алгоритм

1. Преобразуем набор данных в частотную таблицу (frequency table).
2. Создадим таблицу правдоподобия (likelihood table), рассчитав соответствующие вероятности.
3. С помощью теоремы Байеса рассчитаем апостериорную вероятность для каждого класса
4. Класс с наибольшей апостериорной вероятностью будет результатом прогноза.

Реализация

Наивный байесовский классификатор реализован в виде класса NaivBaisClassificator в нем есть 2 главных публичный метода

fit - для обучения и predict - для предсказания, остальные методы приватные и используются в публичных.

```
class NaivBaisClassificator:

    #при инициализации мы создаем out - это словарь. Ключи этого словаря будут название фич. А значения по ключу будет
    #датафрейм.
    #Каждый датафрейм является Likelihood Table т.е. таблицей правдоподобия. На основе этих таблиц в методе -onepredict будут
    #делаться предсказания о пренадлежности объекта к классу 1 или 0
    def __init__(self):
        self.out={}

    #метод _make_likelihood_Table считает Likelihood Table для конкретного признака
    def _make_likelihood_Table(self,labels_and_serie):
        a=(labels_and_serie[labels_and_serie['Survived']==1][labels_and_serie.keys()[1]].value_counts()
        /len(labels_and_serie[labels_and_serie['Survived']==1][labels_and_serie.keys()[1]]))
        b=(labels_and_serie[labels_and_serie['Survived']==0][labels_and_serie.keys()[1]].value_counts()
        /len(labels_and_serie[labels_and_serie['Survived']==0][labels_and_serie.keys()[1]]))
        out_dataframe=DataFrame()
        if set(a.keys())==set(b.keys()):

            out_dataframe[a.name+'_1']=a
            out_dataframe[b.name+'_0']=b
            return out_dataframe
        if len(set(a.keys())-set(b.keys()))!=0:
            for m in list(set(a.keys())-set(b.keys())):
                b[m]=1/len(labels_and_serie[labels_and_serie['Survived']==0][labels_and_serie.keys()[1]])
        if len(set(b.keys())-set(a.keys()))!=0:
            for m in list(set(b.keys())-set(a.keys())):
                        a[m]=1/len(labels_and_serie[labels_and_serie['Survived']==1][labels_and_serie.keys()[1]])
        out_dataframe[a.name+'_1']=a
        out_dataframe[b.name+'_0']=b
        return out_dataframe


    #метод fit заполняет словарь out датафреймами, которые являются Likelihood Table
    def fit(self,train,test):
        data=DataFrame()
        data=train.copy()
        data['Survived']=test.values
        dict_of_df={}
        for i in data.columns[0:-1]:
            dict_of_df[i]=self._make_likelihood_Table(data[['Survived',i]])
        self.out=dict_of_df


    #метод _onepredict делает предсказания класса для одного объекта используя словарь out
    def _onepredict(self,test):
        zero=1
        one=1
        for i in test.keys():
            #использую отловщик ошибок, потому что возможна такая ситуация, что нет ключа в таблице на обучении, это возможно
            #если не было этого значения в обучающей выборке. Особенно это актуально для параметров являющимися действительными
            #числами. К примеру Fare. Я не удалил этот признак, потому что во первых он все-таки может быть в обуч выборке
            # во вторых он может быть полезен в других моделях
            try:
                one=one*float(self.out[i][i+'_1'][test[i]])
                zero=zero*float(self.out[i][i+'_0'][test[i]])
            except:
                one=one*1
                zero=zero*1
        if zero>one:

            return 0
        else:

            return 1

    #метод predict использует метод _onepredict для предсказания класса для одного объекта, при помощи
    #цикла метод predict делает
    #предсказания для кажлого объекта  и возвращает в итоге список меток для каждого объекта
    def predict(self,test_dataset):
        final_predict=[]
        for k in range(test_dataset.shape[0]):
            final_predict.append(self._onepredict(test_dataset.iloc[k]))
        return final_predict
```

Обучение и метрики

In [48]:
```
#обучаю свою модель модель
my_NaivBais=NaivBaisClassificator()
my_NaivBais.fit(X_train,y_train)
#делаем предсказания на трейне и на тесте и смотрим метрики
print('Метрики на обучающей выборки ')
metrics(my_NaivBais.predict(X_train),y_train)
print('Метрики на тестовой выборки ')
metrics(my_NaivBais.predict(X_test),y_test)
```

```
Метрики на обучающей выборки
Accuracy:  0.7929373996789727
Pprecision:  0.7929373996789727
Recall:  0.7929373996789727
F1:  0.7929373996789727
Метрики на тестовой выборки
Accuracy:  0.7686567164179104
Pprecision:  0.7686567164179104
Recall:  0.7686567164179104
F1:  0.7686567164179104
```

In [49]:
```
#обучаем моледь из sklearn
NaivBais=BernoulliNB()
NaivBais.fit(X_train,y_train)
#делаем предсказания на трейне и на тесте и смотрим метрики
print('Метрики на обучающей выборки ')
metrics(NaivBais.predict(X_train),y_train)
print('Метрики на тестовой выборки ')
metrics(NaivBais.predict(X_test),y_test)
```

```
Метрики на обучающей выборки
Accuracy:  0.7897271268057785
Pprecision:  0.7897271268057785
Recall:  0.7897271268057785
F1:  0.7897271268057785
Метрики на тестовой выборки
Accuracy:  0.7910447761194029
Pprecision:  0.7910447761194029
Recall:  0.7910447761194029
F1:  0.7910447761194029
```

Выводы по метрикам

- Моя модель не переобучились, потому что разница метрик на трейне и на тесте моей модели не велика.А sklearn модели вообще метрики выше на тесте чем на трейне
- Модель из sklearn чуть-чуть лучще предсказывает на тесте ,а моя чуть лучше на трейне. Примерно на 0.02-0.03 в каждой метрики
- Метрики обоих моделей находятся между 0.76-0.80.