Оглавление

1 Описание задания и набора данных

Требуется создать модель машинного обучения для предсказания выживания пассажиров во время крушения Титаника с точностью более 80%.

Описание атрибутов набора данных:

- PassengerId – уникальный идентификатор пассажира.
- Survived – выжил пассажир или нет (1 или 0 соответственно).
- Pclass – класс пассажира (1, 2 или 3).
- Name – имя пассажира в формате «Фамилия, Титул. Имя».
- Sex – пол пассажира («female» или «male»).
- Age – возраст пассажира.
- SibSp – количество братьев, сестёр, супругов на борту.
- Parch – количество родителей, детей на борту.
- Ticket – уникальный номер билета.
- Fare – цена билета.
- Cabin – номер каюты.
- Embarked – порт отправления.

Загрузим обучающий набор данных и просмотрим первые записи:

```
import numpy as np   # для работы с массивами
import pandas as pd  # для работы с файлами csv

print('Обучающая выборка')
train_data = pd.read_csv("/content/drive/MyDrive/titanic/train.csv")
train_data.head()
```

```
Обучающая выборка
```

	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

Загрузим набор данных для тестирования и просмотрим первые записи (тестовая выборка не содержит информации о выживаемости пассажира (Survived) – эти значения необходимо предсказать):

```
print('Тестовая выборка')
test_data = pd.read_csv("/content/drive/MyDrive/titanic/test.csv")
test_data.head()
```

```
Тестовая выборка
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

2 Подготовка атрибутов к анализу

- Все используемые атрибуты было решено преобразовать в индикаторные столбцы, каждое значение такого столбца принимает значение 1 или 0 в зависимости от присутствия атрибута. Например, столбец Survived уже имеет такой вид (1 = выжил, 0 = не выжил).
- PassengerId – не используется (идентификаторы пассажиров не имеют взаимосвязи).
- Можно вычислить размер семьи как сумму столбцов **SibSp** и **Parch**. Приводим полученные значения к строковому виду для дальнейшего преобразования столбца в индикатор:

```
train_data["familysize"]=(train_data['SibSp']+train_data['Parch']).astype(str)
```

Столбцы с информацией о поле (**Sex**) и порте отправления (**Embarked**) изначально являются категориальными. С помощью метода pd.get_dummies преобразуем атрибуты familysize, Sex и Embarked в индикаторные столбцы и добавляем их в новый набор данных «data»:

```
# получаем индикаторные столбцы
data = pd.get_dummies(train_data[["Sex","Embarked","familysize"]])
```

Первые строки набора data:

```
data.head()
```

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	familysize_0	familysize_1	familysize_10
0	0	1	0	0	1	0	1	0
1	1	0	1	0	0	0	1	0
2	1	0	0	0	1	1	0	0
3	1	0	0	0	1	0	1	0
4	0	1	0	0	1	1	0	0

familysize_2	familysize_3	familysize_4	familysize_5	familysize_6	familysize_7
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

- С помощью метода np.where получаем индикаторные столбцы для каждого из трёх классов из столбца **Pclass**:

```
data["class1"]= np.where((train_data['Pclass']==1), 1, 0)
data["class2"]= np.where((train_data['Pclass']==2), 1, 0)
data["class3"]= np.where((train_data['Pclass']==3), 1, 0)
```

- По столбцу **Age** с помощью метода mean() находим средний возраст по обучающей выборке. Методом fillna() заполняем средним возрастом пропущенные значение в наборах данных:

```
# замена пропущенных значений возраста

mean_age = train_data["Age"].mean() # средний возраст
print('Средний возраст')
print(mean_age)

train_data["Age"] = train_data["Age"].fillna(mean_age)

test_data["Age"] = test_data["Age"].fillna(mean_age)
```

```
Средний возраст
29.69911764705882
```

Разделим возраст на 4 интервала и создадим столбец-индикатор для каждого из них:

$$age1 \leq 16,$$

$$16 < age2 < mean\_age,$$

$$mean\_age < age3 < 2 * mean\_age,$$

$$age4 \geq 2 * mean\_age.$$

```
data["age1"]= np.where((train_data['Age'] <=16), 1, 0)
data["age2"]=np.where((train_data.Age >16) & (train_data.Age <mean_age),1,0)
data["age3"]=np.where((train_data.Age >mean_age) & (train_data.Age <mean_age*2),1,0)
data["age4"]=np.where( (train_data.Age >=mean_age*2),1,0)
```

- По столбцу **Fare** с помощью метода mean() находим среднюю цену билета по обучающей выборке. Методом fillna() заполняем средней ценой пропущенные значение в наборах данных:

```
# замена пропущенных значений цены билета

mean_fare = train_data["Fare"].mean() # средняя цена
print('Средняя цена билета')
print(mean_fare)

train_data["Fare"] = train_data["Fare"].fillna(mean_fare)
test_data["Fare"] = test_data["Fare"].fillna(mean_fare)
```

```
Средняя цена билета
32.204207968574636
```

Разделим цену на 4 интервала и создадим столбец-индикатор для каждого из них:

$$fare1 \leq mean\_fare/2\,,$$

$$mean\_fare/2 < fare2 < mean\_fare,$$

$$mean\_fare < fare3 < 2 * mean\_fare,$$

$$fare4 \geq 2 * mean\_fare.$$

```
data["fare1"]= np.where((train_data['Fare'] <=mean_fare/2), 1, 0)
data["fare2"]=np.where((train_data.Fare >mean_fare/2) & (train_data.Fare <mean_fare),1,0)
data["fare3"]=np.where((train_data.Fare >mean_fare) & (train_data.Fare <mean_fare*2),1,0)
data["fare4"]=np.where( (train_data.Fare >=mean_fare*2),1,0)
```

- Столбец Cabin не используется из-за большого процента пропущенных значений.
- Столбец Ticket было решено не использовать из-за уникальности значений и сильной логической связи с атрибутами Pclass и Fare.
- По столбцу **Name** можно узнать титул пассажира. Создадим столбцы-индикаторы для наиболее часто встречающихся титулов – Master, Mr, Mrs, Miss. Для этого методом str.partition делим имя по символу точки и методом str.endswith проверяем, что часть имени до точки заканчивается на необходимый титул:

```
data["master"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Master') , 1,0)
data["mr"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Mr') , 1,0)
data["mrs"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Mrs') , 1,0)
data["miss"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Miss') , 1,0)
```

3 Создание модели и тестирование

3.1 Анализ корреляции атрибутов со столбцом Survived

Добавляем в набор данных data атрибут Survived с информацией о выживаемости и вычисляем корреляции остальных столбцов с ним, используя метод corr():

```
data["SURVIVED"]=train_data["Survived"]

# печатаем  корреляции со столбцом Survived
print(data.corr()['SURVIVED'])
```

В полученном результате найдем наибольшие (хотя бы 0.09) корреляции (сам столбец Survived не считаем). Это позволит узнать, какие признаки сильнее всего влияли на выживаемость пассажира:

```
Sex_female       0.543351
Sex_male        -0.543351
Embarked_C       0.168240
Embarked_Q       0.003650
Embarked_S      -0.155660
familysize_0    -0.203367
familysize_1     0.163157
familysize_10   -0.070234
familysize_2     0.143869
familysize_3     0.128347
familysize_4    -0.049466
familysize_5    -0.080968
familysize_6    -0.012134
familysize_7    -0.064988
```

```
class1           0.285904
class2           0.093349
class3          -0.322308
age1             0.121485
age2            -0.039670
age3             0.050201
age4            -0.040857
fare1           -0.276811
fare2            0.082939
fare3            0.070220
fare4            0.243110
master           0.085221
mr              -0.549199
mrs              0.339040
miss             0.327093
SURVIVED         1.000000
Name: SURVIVED, dtype: float64
```

Видно, что размер семьи можно разделить на 3 группы: у людей, путешествовавших в одиночестве (familysize_0), корреляция = -0.2; у тех, с кем было еще от 1 до 3 человек, корреляция колеблется от 0.128 до 0.16; у пассажиров, с которыми было от 4 до 10 человек отрицательная корреляция. Таким образом, объединим атрибуты familysize в 3 столбца:

- **lonely**: $familysize = 0$.
- **small_family**: $0 < familysize \leq 3$.
- **big_family**: $familysize > 3$.

Создадим соответствующие индикаторные столбцы:

```
# столбцы по размеру семьи

data["lonely"]= np.where((train_data['SibSp']+train_data['Parch'])==0, 1, 0)
data["small_family"]= np.where(((train_data['SibSp']+train_data['Parch'])>0) & ((train_data['SibSp']+train_data['Parch'])<=3) , 1, 0)
data["big_family"]= np.where((train_data['SibSp']+train_data['Parch'])>3, 1, 0)
```

Снова вычислим корреляции и выделим наиболее важные признаки, не обращая внимание на старые атрибуты для размера семьи:

```
Sex_female       0.543351
Sex_male        -0.543351
Embarked_C       0.168240
Embarked_Q       0.003650
Embarked_S      -0.155660
familysize_0    -0.203367
familysize_1     0.163157
familysize_10   -0.070234
familysize_2     0.143869
familysize_3     0.128347
familysize_4    -0.049466
familysize_5    -0.080968
familysize_6    -0.012134
familysize_7    -0.064988
class1           0.285904
class2           0.093349
class3          -0.322308
age1             0.121485
age2            -0.039670
age3             0.050201
age4            -0.040857
```

```
fare1           -0.276811
fare2            0.082939
fare3            0.070220
fare4            0.243110
master           0.085221
mr              -0.549199
mrs              0.339040
miss             0.327093
SURVIVED         1.000000
lonely          -0.203367
small_family     0.279855
big_family      -0.125147
Name: SURVIVED, dtype: float64
```

3.2 Создание модели

Выбираем для итогового обучающего набора данных data только выделенные атрибуты (с наибольшей корреляцией):

```
# атрибуты для обучения
data = pd.get_dummies(train_data[["Sex"]])
data["C"] = np.where((train_data["Embarked"]=='C'),1,0)
data["S"] = np.where((train_data["Embarked"]=='S'),1,0)
data["class1"]= np.where((train_data['Pclass']==1), 1, 0)
data["class2"]= np.where((train_data['Pclass']==2), 1, 0)
data["class3"]= np.where((train_data['Pclass']==3), 1, 0)
data["age1"]= np.where((train_data['Age'] <=16), 1, 0)
data["fare1"]= np.where((train_data['Fare'] <=mean_fare/2), 1, 0)
data["fare4"]=np.where( (train_data.Fare >=mean_fare*2),1,0)
data["mr"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Mr') , 1,0)
data["mrs"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Mrs') , 1,0)
data["miss"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Miss') , 1,0)
data["lonely"]= np.where((train_data['SibSp']+train_data['Parch'])==0, 1, 0)
data["small_family"]= np.where(((train_data['SibSp']+train_data['Parch'])>0) & ((train_data['SibSp']+train_data['Parch'])<=3) , 1, 0)
data["big_family"]= np.where((train_data['SibSp']+train_data['Parch'])>3, 1, 0)
```

Создаём аналогичный набор данных tdata для тестирования, используя тестовую выборку из файла test_data.csv вместо обучающей выборки train_data.csv, на основе которой создавался набор data:

```
# атрибуты для тестирования
tdata = pd.get_dummies(test_data[["Sex"]])
tdata["C"] = np.where((test_data["Embarked"]=='C'),1,0)
tdata["S"] = np.where((test_data["Embarked"]=='S'),1,0)
tdata["class1"]= np.where((test_data['Pclass']==1), 1, 0)
tdata["class2"]= np.where((test_data['Pclass']==2), 1, 0)
tdata["class3"]= np.where((test_data['Pclass']==3), 1, 0)
tdata["age1"]= np.where((test_data['Age'] <=16), 1, 0)
tdata["fare1"]= np.where((test_data['Fare'] <=mean_fare/2), 1, 0)
tdata["fare4"]=np.where( (test_data.Fare >=mean_fare*2),1,0)
tdata["mr"]= np.where( ((test_data.Name).str.partition('.')[0]).str.endswith('Mr') , 1,0)
tdata["mrs"]= np.where( ((test_data.Name).str.partition('.')[0]).str.endswith('Mrs') , 1,0)
tdata["miss"]= np.where( ((test_data.Name).str.partition('.')[0]).str.endswith('Miss') , 1,0)
tdata["lonely"]= np.where((test_data['SibSp']+test_data['Parch'])==0, 1, 0)
tdata["small_family"]= np.where(((test_data['SibSp']+test_data['Parch'])>0) & ((test_data['SibSp']+test_data['Parch'])<=3) , 1, 0)
tdata["big_family"]= np.where((test_data['SibSp']+test_data['Parch'])>3, 1, 0)
```

Для каждого набора создаём столбец «chance», который представляет собой линейную комбинацию остальных столбцов с коэффициентами, являющимися округлёнными до сотых значениями корреляции соответствующих атрибутов с атрибутом Survived:

```
data["chance"] = (data["Sex_female"]*0.54 - data["Sex_male"]*0.54
                  +data["C"]*0.17-data["S"]*0.16+data["class1"]*0.29
                  +data["class2"]*0.09-data["class3"]*0.32+ data["age1"]*0.12
                  - data['fare1']*0.28+data['fare4']*0.24-data['lonely']*0.2
                  + data['small_family']*0.28-data['big_family']*0.13
                  -data["mr"]*0.55+data["mrs"]*0.34+data["miss"]*0.33)
```

```
tdata["chance"] = (tdata["Sex_female"]*0.54 - tdata["Sex_male"]*0.54
                  +tdata["C"]*0.17-tdata["S"]*0.16+tdata["class1"]*0.29
                  +tdata["class2"]*0.09-tdata["class3"]*0.32+ tdata["age1"]*0.12
                  - tdata['fare1']*0.28+tdata['fare4']*0.24-tdata['lonely']*0.2
                  + tdata['small_family']*0.28-tdata['big_family']*0.13
                  -tdata["mr"]*0.55+tdata["mrs"]*0.34+tdata["miss"]*0.33)
```

Только этот столбец и будет использоваться в предсказании.

Для обучения модели было решено применить гауссовский наивный байесовский классификатор (GaussianNB в sklearn), т.к. байесовские классификаторы отлично работают на независимых атрибутах (как было сказано выше, в модели будет использован всего один независимый столбец «chance»). Так как «chance» содержит непрерывные значения, классификатор выбран не категориальный, а именно гауссовский.

Импортируем нужную библиотеку и создаем метки для обучения:

```
# гауссовский наивный байесовский классификатор
from sklearn.naive_bayes import GaussianNB

y = train_data['Survived'] # метки
```

Обучаем модель и выводим на экран точность предсказаний на обучающем наборе:

```
model = GaussianNB() # создаём модель
model.fit(data[['chance']], y) # обучаем
# результат на обучающей выборке
display(model.score(data[['chance']], y))
```

```
0.7901234567901234
```

Видно, что полученный результат не достигает желаемого процента точности (80%). Следовательно, перед проверкой на тестовом наборе модель необходимо улучшить. Это можно сделать, если учесть не только корреляции атрибутов со столбцом Survived, но и между собой.

3.3 Анализ корреляции атрибутов с учетом пола пассажира

Результаты из п. 3.1 показали, что наибольшее влияние на выживаемость имел пол (Sex) пассажира (корреляция со столбцом Survived равна ±0.543351). Таким образом, имеет смысл создать атрибуты с информацией о выживаемости женщин («female_survived»: 1 – соответствует выжившей женщине, 0 – в противном случае) и мужчин («male_survived»: 1 – соответствует выжившему мужчине, 0 – в противном случае) и вычислить их корреляции с остальными атрибутами:

```
# выжившие женщины
data["female_survived"]=train_data.loc[train_data.Sex == 'female']["Survived"]

# выжившие мужчины
data["male_survived"]=train_data.loc[train_data.Sex == 'male']["Survived"]

print("Влияние на выживаемость женщин")
print(data.corr()["female_survived"])

print("Влияние на выживаемость мужчин")
print(data.corr()["male_survived"])
```

Выделим атрибуты, оказывающие наибольшее влияние (корреляция >=0.09) на выживаемость женщин и мужчин:

```
Влияние на выживаемость женщин
Sex_female              NaN
Sex_male                NaN
Embarked_C         0.169413
Embarked_Q         0.006549
Embarked_S        -0.161915
class1             0.337723
class2             0.231215
class3            -0.509155
age1              -0.067392
age2              -0.011307
age3               0.097282
age4               0.066975
fare1             -0.158651
fare2             -0.084936
fare3              0.054252
fare4              0.235028
master                  NaN
mr                      NaN
mrs                0.092869
miss              -0.118723
lonely             0.081726
small_family       0.145363
big_family        -0.367598
female_survived    1.000000
male_survived           NaN
Name: female_survived, dtype: float64
```

```
Влияние на выживаемость мужчин
Sex_female              NaN
Sex_male                NaN
Embarked_C         0.131966
Embarked_Q        -0.081775
Embarked_S        -0.065808
class1             0.238041
class2            -0.038618
class3            -0.167757
age1               0.192877
age2              -0.058136
age3               0.022042
age4              -0.026726
fare1             -0.219967
fare2              0.128121
fare3              0.107660
fare4              0.086801
master             0.269199
mr                -0.241734
mrs                     NaN
miss                    NaN
lonely            -0.133419
small_family       0.188539
big_family        -0.090754
female_survived         NaN
male_survived      1.000000
Name: male_survived, dtype: float64
```

3.4 Создание улучшенной модели

Так же, как и в п. 3.2, включаем выделенные атрибуты в наборы data и tdata:

```
# атрибуты для обучения
data = pd.get_dummies(train_data[["Sex"]])
data["C"] = np.where((train_data["Embarked"]=='C'),1,0)
data["S"] = np.where((train_data["Embarked"]=='S'),1,0)
data["class1"]= np.where((train_data['Pclass']==1), 1, 0)
data["class2"]= np.where((train_data['Pclass']==2), 1, 0)
data["class3"]= np.where((train_data['Pclass']==3), 1, 0)
data["age1"]= np.where((train_data['Age'] <=16), 1, 0)
data["age3"]=np.where((train_data.Age >mean_age) & (train_data.Age <mean_age*2),1,0)
data["fare1"]= np.where((train_data['Fare'] <=mean_fare/2), 1, 0)
data["fare2"]=np.where((train_data.Fare >mean_fare/2) & (train_data.Fare <mean_fare),1,0)
data["fare3"]=np.where((train_data.Fare >mean_fare) & (train_data.Fare <mean_fare*2),1,0)
data["fare4"]=np.where( (train_data.Fare >=mean_fare*2),1,0)
data["master"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Master') , 1,0)
data["mr"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Mr') , 1,0)
data["mrs"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Mrs') , 1,0)
data["miss"]= np.where( ((train_data.Name).str.partition('.')[0]).str.endswith('Miss') , 1,0)
data["lonely"]= np.where((train_data['SibSp']+train_data['Parch'])==0, 1, 0)
data["small_family"]= np.where(((train_data['SibSp']+train_data['Parch'])>0) & ((train_data['SibSp']+train_data['Parch'])<=3) , 1, 0)
data["big_family"]= np.where((train_data['SibSp']+train_data['Parch'])>3, 1, 0)
```

```
# атрибуты для тестирования
tdata = pd.get_dummies(test_data[["Sex"]])
tdata["C"] = np.where((test_data["Embarked"]=='C'),1,0)
tdata["S"] = np.where((test_data["Embarked"]=='S'),1,0)
tdata["class1"]= np.where((test_data['Pclass']==1), 1, 0)
tdata["class2"]= np.where((test_data['Pclass']==2), 1, 0)
tdata["class3"]= np.where((test_data['Pclass']==3), 1, 0)
tdata["age1"]= np.where((test_data['Age'] <=16), 1, 0)
tdata["age3"]=np.where((test_data.Age >mean_age) & (test_data.Age <mean_age*2),1,0)
tdata["fare1"]= np.where((test_data['Fare'] <=mean_fare/2), 1, 0)
tdata["fare2"]=np.where((test_data.Fare >mean_fare/2) & (test_data.Fare <mean_fare),1,0)
tdata["fare3"]=np.where((test_data.Fare >mean_fare) & (test_data.Fare <mean_fare*2),1,0)
tdata["fare4"]=np.where( (test_data.Fare >=mean_fare*2),1,0)
tdata["master"]= np.where( ((test_data.Name).str.partition('.')[0]).str.endswith('Master') , 1,0)
tdata["mr"]= np.where( ((test_data.Name).str.partition('.')[0]).str.endswith('Mr') , 1,0)
tdata["mrs"]= np.where( ((test_data.Name).str.partition('.')[0]).str.endswith('Mrs') , 1,0)
tdata["miss"]= np.where( ((test_data.Name).str.partition('.')[0]).str.endswith('Miss') , 1,0)
tdata["lonely"]= np.where((test_data['SibSp']+test_data['Parch'])==0, 1, 0)
tdata["small_family"]= np.where(((test_data['SibSp']+test_data['Parch'])>0) & ((test_data['SibSp']+test_data['Parch'])<=3) , 1, 0)
tdata["big_family"]= np.where((test_data['SibSp']+test_data['Parch'])>3, 1, 0)
```

Для каждого набора создаём столбец «chance», который вычисляется следующим образом:

$$chance = Sex\_female * \left(female\_survived\_corr + \sum_{i=1}^{n} attribute_i * female\_attribute_i\_corr\right) + Sex\_male * \left(male\_survived\_corr + \sum_{i=1}^{m} attribute_i * male\_attribute_i\_corr\right),$$

где Sex_female, Sex_male – индикаторные столбцы для пола пассажира; female_survived_corr и male_survived_corr – вычисленные ранее и округленные до сотых корреляции между столбцами Sex_female, Sex_male и столбцом Survived (0.54 – для женщин, -0.54 – для мужчин); n – количество атрибутов, влияющих на выживаемость женщин; m – количество атрибутов, влияющих на выживаемость мужчин; $attribute_i$ – атрибут из набора данных, $female\_attribute_i\_corr$ – округленное до сотых значение корреляции между атрибутом и индикаторным столбцом выживаемости женщин (female_survived); $male\_attribute_i\_corr$ – округленное до сотых значение корреляции между атрибутом и индикаторным столбцом выживаемости мужчин (male_survived).

Таким образом, для каждой строки набора данных будет вычисляться только одна из скобок – потому что человек может быть только мужчиной или женщиной (если мужчина – первая скобка обнулится, если женщина – обнулится вторая скобка).

```
data["chance"]=(data["Sex_female"]*(0.54+0.1*data['age3']-0.12*data['miss']
                                    +0.09*data['mrs']+data['C']*0.17-data['S']*0.16
                                    +data['small_family']*0.15-data["big_family"]*0.37
                                    +data['class1']*0.34+data['class2']*0.23-data['class3']*0.51
                                    -data['fare1']*0.16+data['fare4']*0.24)
                                    +data["Sex_male"]*(-0.54+0.27*data["master"]-0.24*data["mr"]
                                                       +0.13*data['C']-0.13*data['lonely']
                                                       +0.19*data['small_family']-0.09*data['big_family']
                                                       +0.24*data['class1']-0.17*data['class3']
                                                       +0.19*data['age1']-0.22*data['fare1']+0.13*data['fare2']
                                                       +0.11*data['fare3']))

tdata["chance"]=(tdata["Sex_female"]*(0.54+0.1*tdata['age3']-0.12*tdata['miss']
                                    +0.09*tdata['mrs']+tdata['C']*0.17-tdata['S']*0.16
                                    +tdata['small_family']*0.15-tdata["big_family"]*0.37
                                    +tdata['class1']*0.34+tdata['class2']*0.23-tdata['class3']*0.51
                                    -tdata['fare1']*0.16+tdata['fare4']*0.24)
                                    +tdata["Sex_male"]*(-0.54+0.27*tdata["master"]-0.24*tdata["mr"]
                                                       +0.13*tdata['C']-0.13*tdata['lonely']
                                                       +0.19*tdata['small_family']-0.09*tdata['big_family']
                                                       +0.24*tdata['class1']-0.17*tdata['class3']
                                                       +0.19*tdata['age1']-0.22*tdata['fare1']+0.13*tdata['fare2']
                                                       +0.11*tdata['fare3']))
```

Снова обучаем модель гауссовским наивным байесовским классификатором и узнаём точность предсказания по обучающей выборке:

```
# гауссовский наивный байесовский классификатор
from sklearn.naive_bayes import GaussianNB

y = train_data['Survived'] # метки

model = GaussianNB() # создаём модель
model.fit(data[['chance']], y) # обучаем
# результат на обучающей выборке
display(model.score(data[['chance']], y))
```

```
0.8148148148148148
```

Точность превысила 80%. Теперь можно переходить к проверке на тестовых данных. Используем для этого метод predict и формируем csv-файл в формате «PassengerId, Survived», включая заголовок:

```
predictions = model.predict(tdata[['chance']])

output = pd.DataFrame({'PassengerId': test_data.PassengerId, 'Survived': predictions})
output.to_csv('try.csv', index=False)
```

Так выглядит начало файла:

	A
1	PassengerId,Survived
2	892
3	893,1
4	894
5	895
6	896,1

После загрузки полученного файла на проверку, получили точность предсказания 0.80861 на тестовой выборке.

4 Выводы

В результате выполнения работы была получена точность предсказания 0.814815 на обучающей выборке и 0.80861 на тестовой. Примечательно то, что метод дал хорошее обобщение – результаты на обучающей и тестовой выборке отличаются менее чем на 0.01. В ходе выполнения работы было выявлено, что, выделяя наибольшие корреляции между атрибутами, можно улучшить точность предсказания.