Отчет по лабораторной работе
Наивный Байесовский классификатор

12 ноября 2023 г.

Шаронов Артем
Группа: 5140201/30302

Задание №1

Исследуйте, как объем обучающей выборки и количество тестовых данных, влияет на точность классификации или на вероятность ошибочной классификации в примере крестики-нолики и примере о спаме e-mail сообщений.

Результат:

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.preprocessing import LabelEncoder
from ucimlrepo import fetch_ucirepo
```

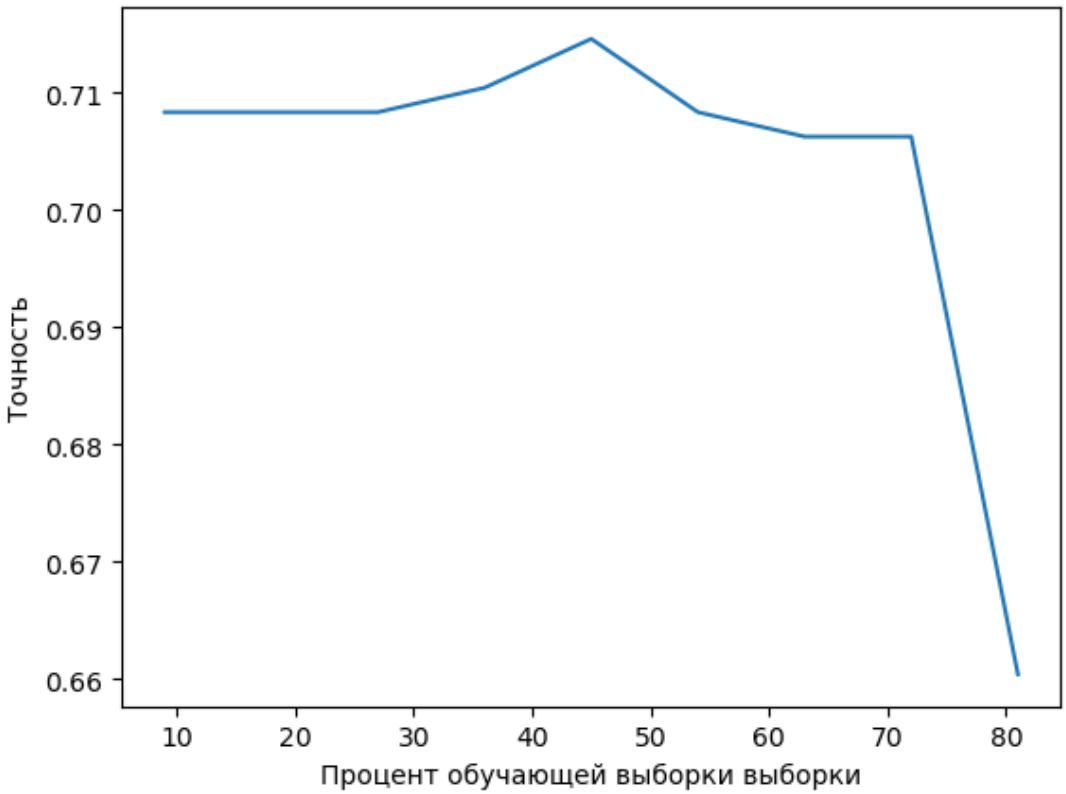
Зависимость точности классификации от объёма обучающей выборки:

[3]:
```
# Загружаем DS крестики-нолики (Data set)
data_XO = pd.read_table('Tic_tac_toe.txt',␣
 ↪sep=',',names=['V1','V2','V3','V4','V5','V6','V7','V8','V9','V10'])

# Преобразуем символы в числа в DS крестики нолики
label_encoder = LabelEncoder()
for column in data_XO.columns:
    data_XO[column] = label_encoder.fit_transform(data_XO[column])

# Извлечение признаков и целевой переменной
X = data_XO.drop('V10', axis=1)
y = data_XO['V10']
```

[4]:
```
# Отделяем тестую выборку в размере 10 % от всех данных
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪1, random_state=42)
X = X_train
y = y_train

# Список для хранения результатов
results = []

# Задаем диапазон процента обучающей выборки
test_sizes = np.arange(0.1, 1.0, 0.1)
print (test_sizes)
```

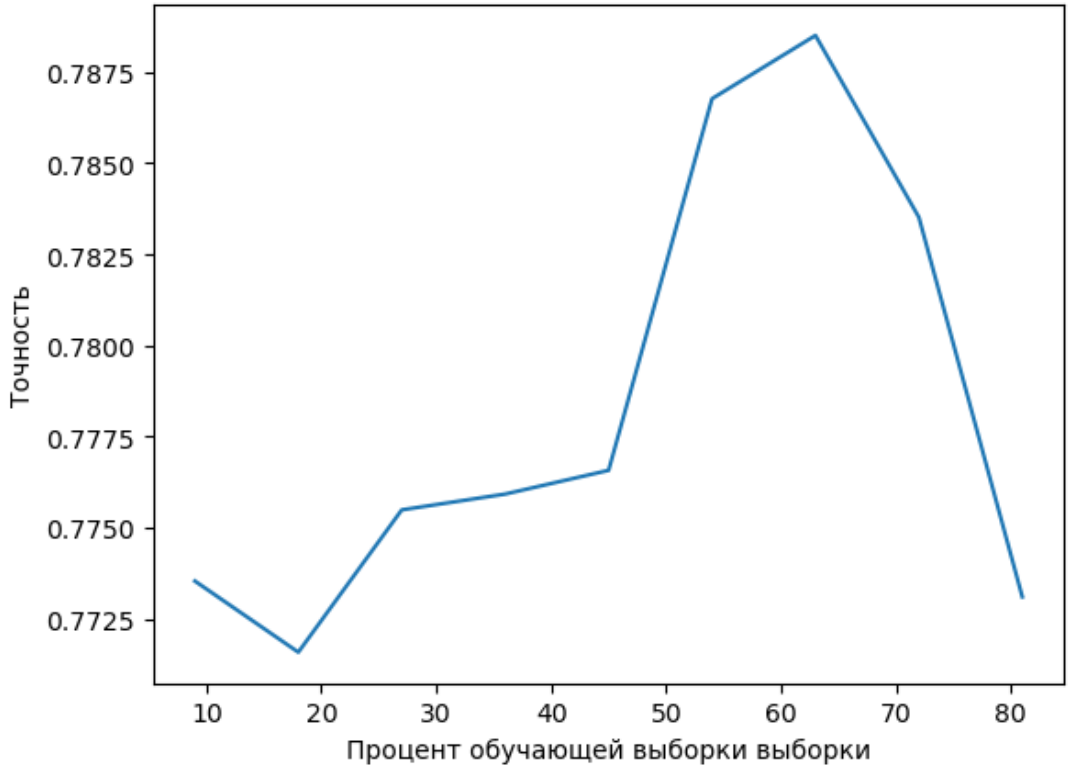
```
# Задаем N (Будет находиться средний из N результатов для каждой␣
 ↪обучающей выборки)
N =10

# Вычисляем точность класофикации N раз
for i in range(N):
        for test_size in test_sizes:  # Меняем размер обучающей выборки␣
 ↪от 81% до 9%
            X_train, X_non, y_train, y_non = train_test_split(X, y,␣
 ↪test_size=test_size)

            # Создаем и обучаем модель
            nbais = MultinomialNB()
            nbais.fit(X_train, y_train)

            # Предсказываем классы на тестовой выборке
            nbais.predict(X_test)

             # Оценка качества классификации
            accuracy = nbais.score(X_test,y_test)

            # Добавление результатов в список
            found = False
            for i, (first, second) in enumerate(results):
                if first == test_size:
                    results[i] = (first, second + accuracy)
                    found = True
                    break

            if not found:
                results.append((test_size, accuracy))

# Создание DataFrame из списка результатов
for i, (first, second) in enumerate(results):
    results[i] = (first*90, second/N)

results_df = pd.DataFrame(results, columns=['Test Size', 'Accuracy'])


# Создание графика
plt.plot(results_df['Test Size'], results_df['Accuracy'])
plt.xlabel('Процент обучающей выборки выборки')
plt.ylabel('Точность')
# Отображение графика
```

```
plt.show()
```



[5]:
```
# Загружаем DS спам-сообщений (Data set)
data_spam = fetch_ucirepo(id=94)

#DS E-mail_spam
X = data_spam.data.features
y = data_spam.data.targets.values.ravel()
```

Повторяем код из блока [4] с DS спам-сообщений.



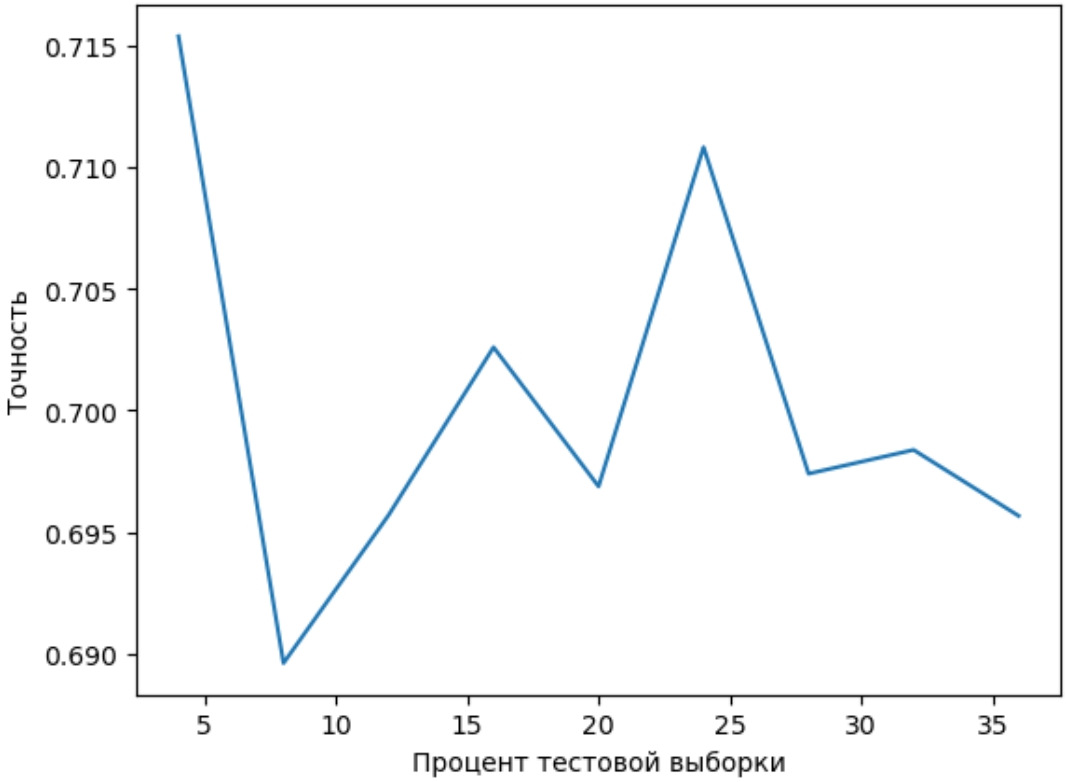
По мере увеличения объёма обучающей выборки точность классификации на тестовых данных сначала улучшается, поскольку новые данные помогают модели лучше обучиться. Однако, в дальнейшем увеличение обучающей выборки приводит к переобучению модели и, как следствие, снижению точности на тестовых данных.

Зависимость точности классификации от количества тестовых данных

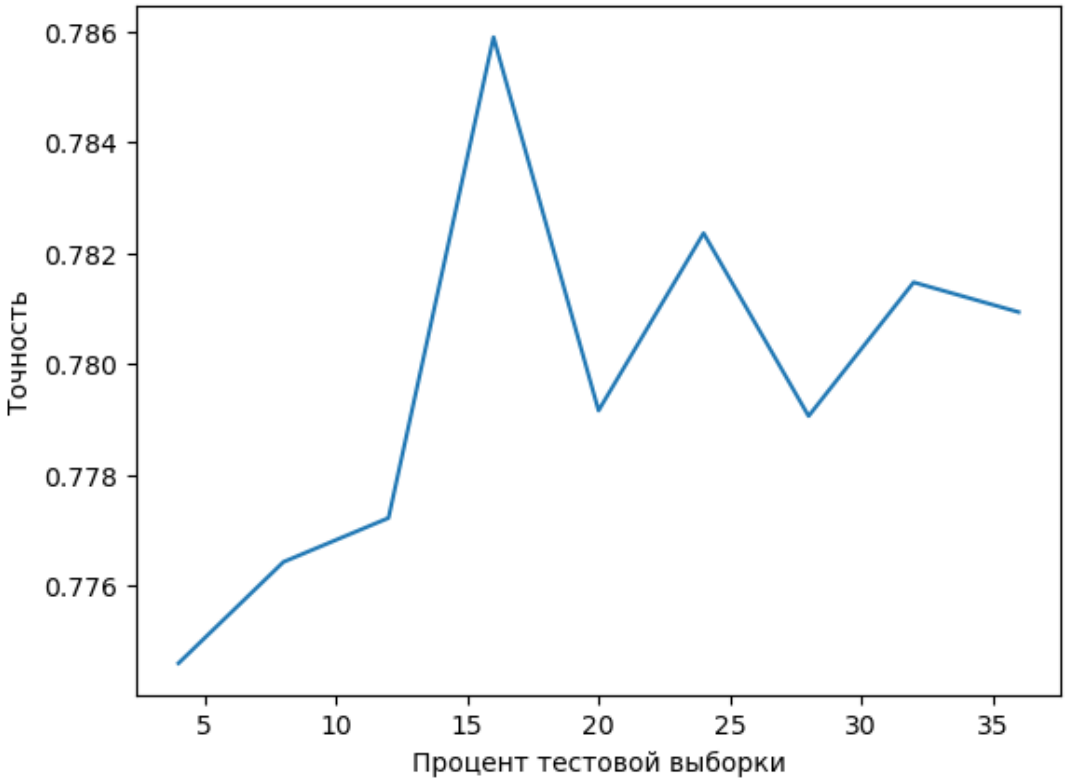
Загружаем DS крестики-нолики как в блоке [2]

[6]:
```
# Отделяем обучающую выбоку в размере 60 % от всех данных
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪40, random_state=42)
X = X_test
y = y_test

# Список для хранения результатов
results = []

# Задаем диапазон процента тестовой выборки
test_sizes = np.arange(0.1, 1.0, 0.1)
print (test_sizes)
```

```
# Задаем N (Будет находиться средний из N результатов для каждой␣
 ↪тестовой выборки)
N = 10

# Цикл по значениям i от 0 до N-1
for i in range(N):
        # Изменение размера обучающей и тестовой выборок и оценка␣
 ↪точности
        for test_size in test_sizes:  # Меняем размер тестовой выборки␣
 ↪выборки от 4% до 36%
            X_non, X_test, y_non, y_test = train_test_split(X, y,␣
 ↪test_size=test_size)

            # Создаем и обучаем модель
            nbais = MultinomialNB()
            nbais.fit(X_train, y_train)

            # Предсказываем классы на тестовой выборке
            nbais.predict(X_test)

             # Оценка качества классификации
            accuracy = nbais.score(X_test,y_test)

            # Добавление результатов в список
            found = False
            for i, (first, second) in enumerate(results):
                if first == test_size:
                    results[i] = (first, second + accuracy)
                    found = True
                    break

            if not found:
                results.append((test_size, accuracy))

# Создание DataFrame из списка результатов
for i, (first, second) in enumerate(results):
    results[i] = (first*40, second/N)
results_df = pd.DataFrame(results, columns=['Test Size', 'Accuracy'])

# Создание графика
plt.plot(results_df['Test Size'], results_df['Accuracy'])
plt.xlabel('Процент тестовой выборки')
plt.ylabel('Точность')
# Отображение графика
plt.show()
```



Загружаем DS спам-сообщений как в блоке [5] и повторяем для него блок [6].



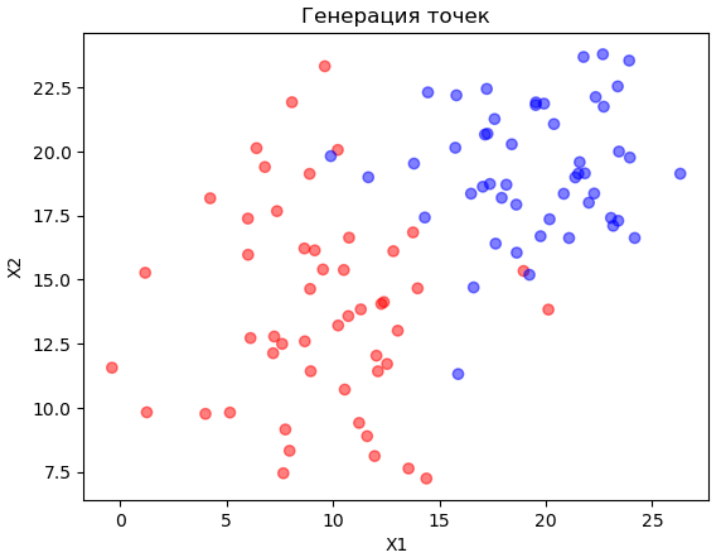
Разброс значений точности для обоих примеров не превышает 2.5%, при этом связь между величинами неочевидна. Такое поведение можно объяснить случайностью выбора данных при формировании тестовой выборки.

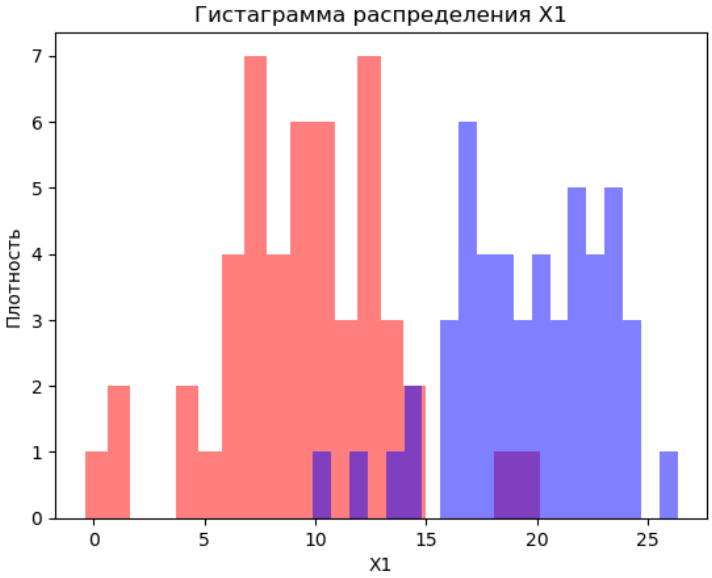
Задание №2

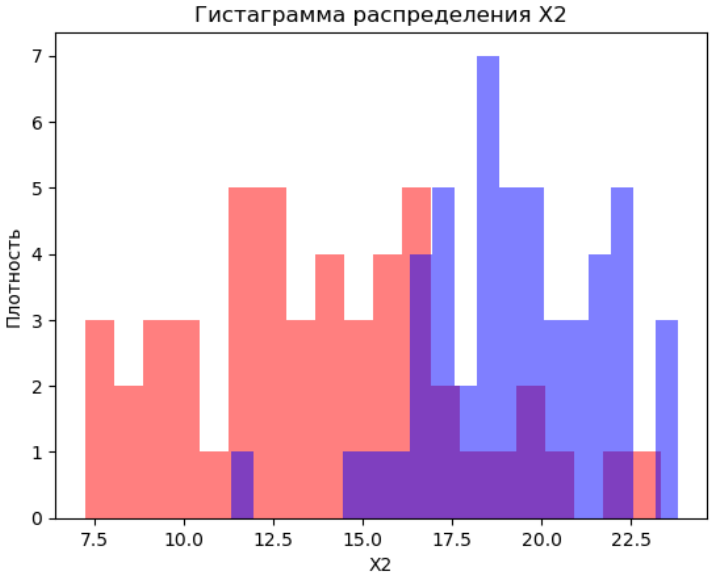
Сгенерируйте 100 точек с двумя признаками X1 и X2 в соответствии с нормальным распределением так, что первые 50 точек (class -1) имеют параметры: мат. ожидание X1 равно 10, мат. ожидание X2 равно 14, среднеквадратические отклонения для обеих переменных равны 4. Вторые 50 точек (class +1) имеют параметры: мат. ожидание X1 равно 20, мат. ожидание X2 равно 18, среднеквадратические отклонения для обеих переменных равны 3. Построить соответствующие диаграммы, иллюстрирующие данные. Построить байесовский классификатор и оценить качество классификации.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Генерация точек
X1_1 = np.random.normal(10, 4, size =50)
X1_2 = np.random.normal(20, 3, size =50)
X1 = np.concatenate((X1_1,X1_2))

X2_1 = np.random.normal(14, 4, size =50)
X2_2 = np.random.normal(18, 3, size =50)
X2 = np.concatenate((X2_1,X2_2))

X = pd.DataFrame({'X1': X1, 'X2': X2})
y = np.concatenate((np.ones(50)*-1,np.ones(50)))

# Создание графика множества сгенерированных точек на плоскости
colors = ['red'] * 50 + ['blue'] * 50
plt.scatter(X['X1'], X['X2'], c=colors, alpha=0.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Генерация точек')

# Отображение графика
plt.show()

# Создание графика распределения X1
plt.hist(X1_1, bins=20, color='red', alpha=0.5)
plt.hist(X1_2, bins=20, color='blue', alpha=0.5)
plt.xlabel('X1')
plt.ylabel('Плотность')
plt.title('Гистаграмма распределения X1')
# Отображение графика
```

```
plt.show()

# Создание графика распределения X2
plt.hist(X2_1, bins=20, color='red', alpha=0.5)
plt.hist(X2_2, bins=20, color='blue', alpha=0.5)
plt.xlabel('X2')
plt.ylabel('Плотность')
plt.title('Гистаграмма распределения X2')
# Отображение графика
plt.show()

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)

# Байесовский классификатор
nbais = GaussianNB()
nbais.fit(X_train, y_train)

# Прогнозирование
nbais.predict(X_test)

# Оценка качества классификации
print(nbais.score(X_test,y_test))
```







Множество сгенерированных точек соответствует требованиям задания. Сформируем обучающую выборку, содержащую 60 точек. Остальные 40 точек поместим в тестовую выборку. Байесовский классификатор показывает точность в 92,5%.

Задание №3

Разработать байесовский классификатор для данных Титаник (Titanic dataset) - https://www.kaggle.com/c/titanic

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, roc_auc_score,␣
 ↪accuracy_score
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import LabelEncoder

DS = pd.read_csv('train.csv')

DS.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
```

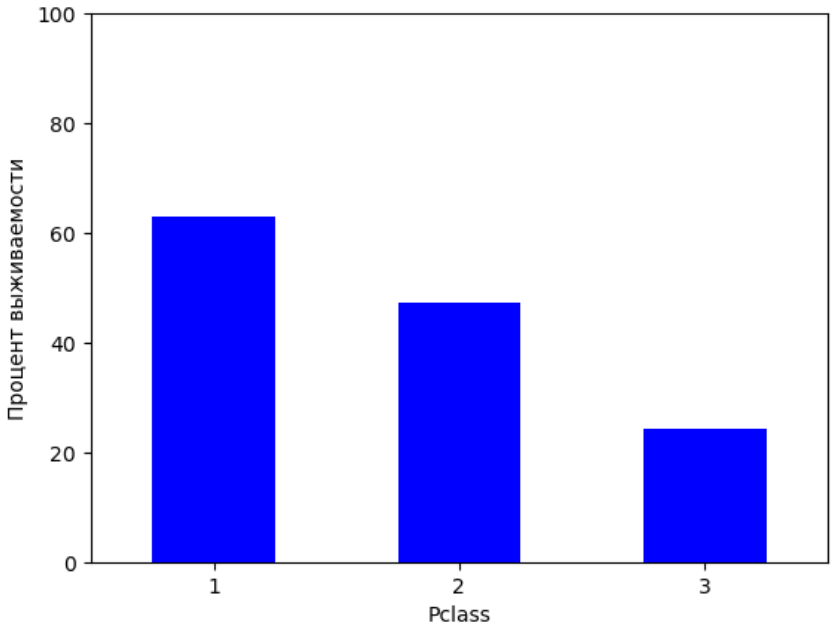
Рассмотрим только параметры Pclass, Sex, Age, SibSp, Parch и Embarked. Не у всех пассажиров заполнен каждый из выбранных нами параметров, исправим это.

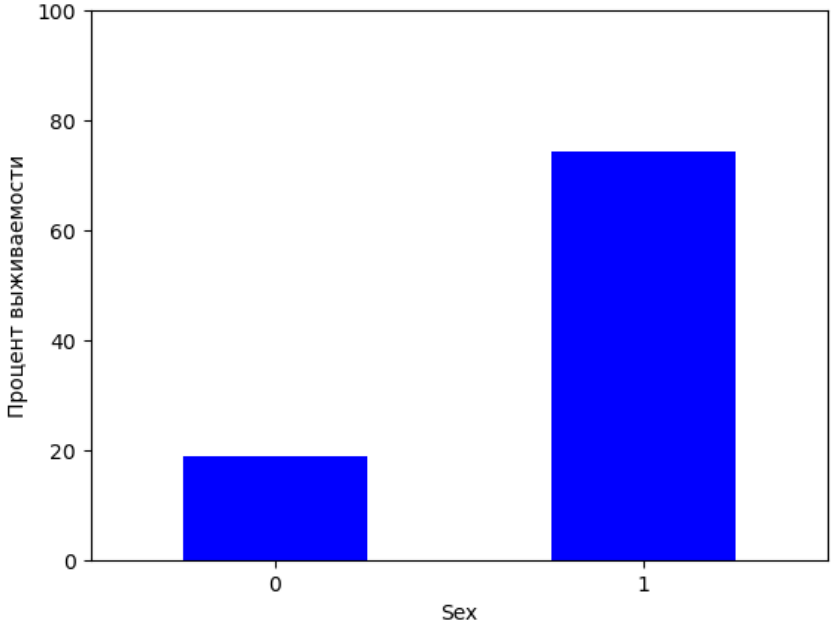
[2]:
```
myImputer = SimpleImputer (strategy= 'median')
DS.Age = myImputer.fit_transform(DS['Age'].values.reshape(-1,1))
myImputer = SimpleImputer (strategy= 'most_frequent')
```

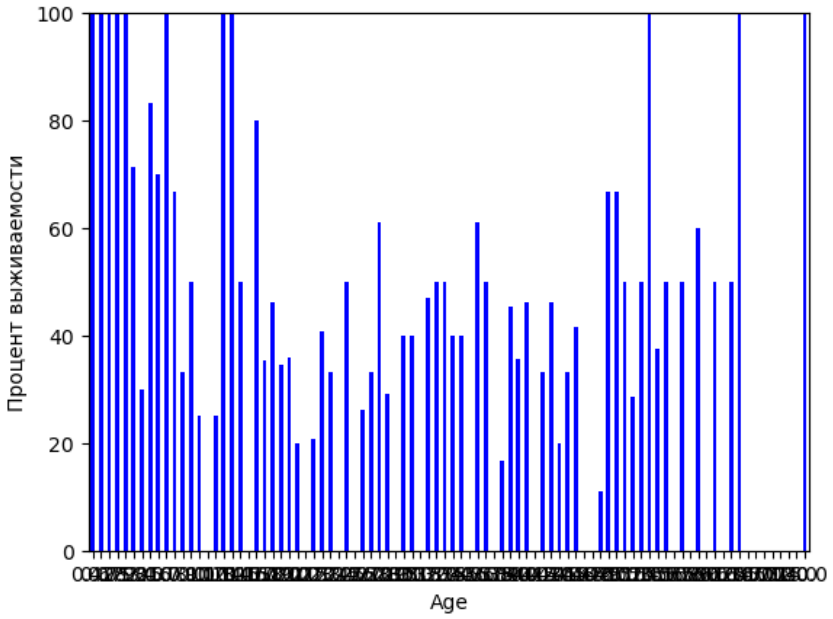
```
DS.Embarked = myImputer.fit_transform(DS['Embarked'].values.
 ↪reshape(-1,1))
```

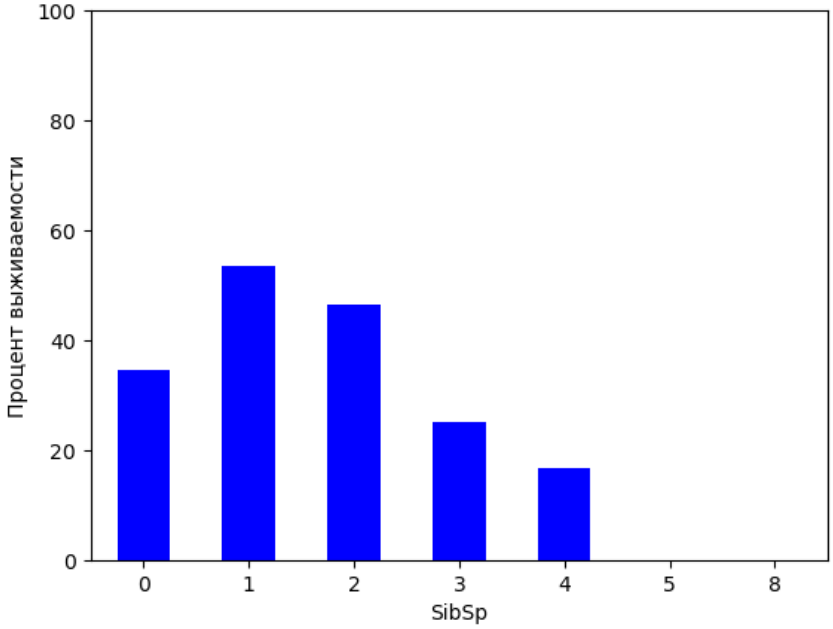
Рассмотрим влияние каждого параметра на выживаемость.

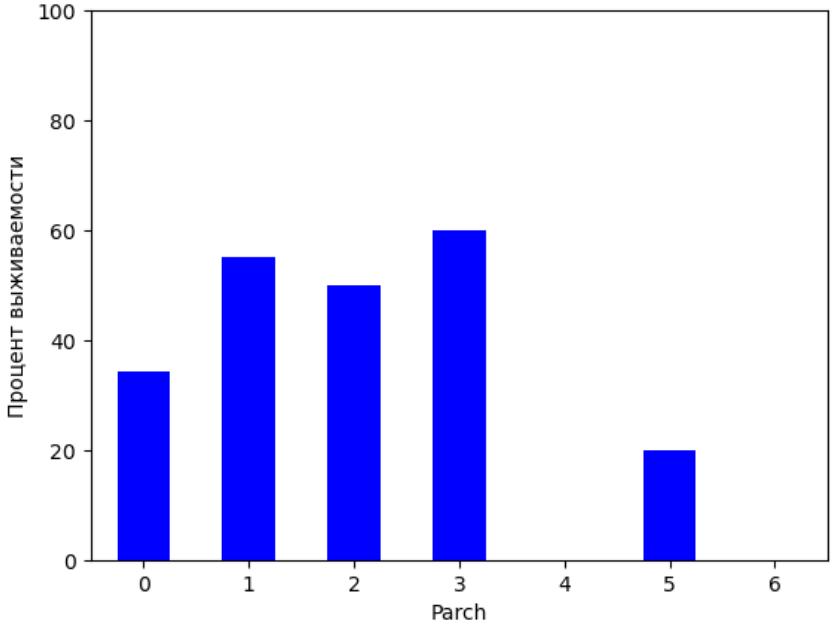
[4]:
```
# Рассчитываем процент выживаемости для каждого параметра
survival_percentages = {}
for column in DS.columns:
    if column not in ['Survived', 'Name', 'PassengerId', 'Ticket',␣
 ↪'Fare', 'Cabin'] :
        survival_percentages[column] = DS.groupby(column)['Survived'].
 ↪mean() * 100

# Создание графиков
for parameter, percentages in survival_percentages.items():
    plt.figure()
    percentages.plot(kind='bar', color='blue')
    plt.xlabel(parameter)
    plt.ylabel('Процент выживаемости')
    plt.ylim([0, 100])
    plt.xticks(rotation=0)
    plt.show()
```

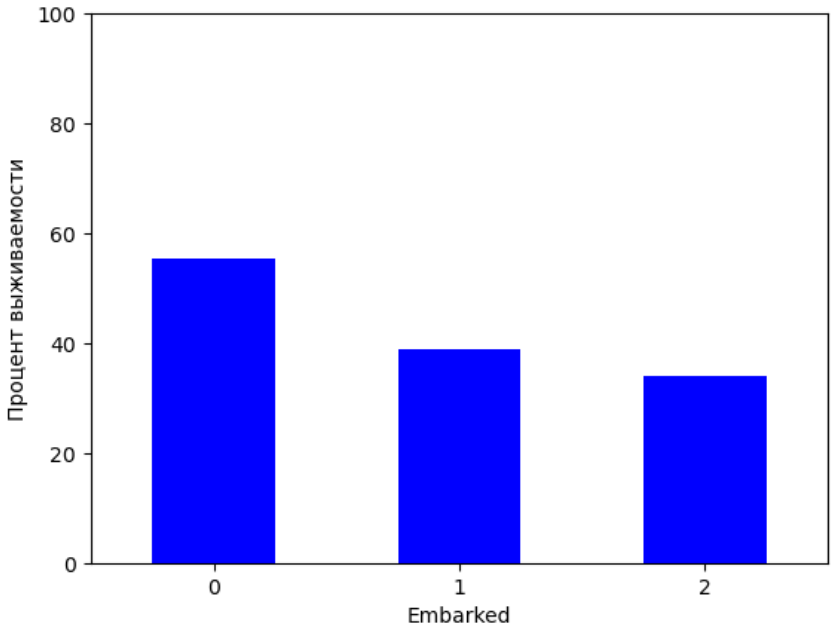



100
80
60
40
20
0
Процент выживаемости
0
1
Sex


100
80
60
40
20
0
Процент выживаемости
Age


Процент выживаемости
100
80
60
40
20
0
0
1
2
3
4
5
8
SibSp


Процент выживаемости
100
80
60
40
20
0
0
1
2
3
4
5
6
Parch



Параметры Pclass, Sex и Age сильнее всего влияют на выживаемость. Рассмотрим как комбинация с другими параметрами влияет на точность.

[6]:
```
# Преобразуем символы и строки в числа в DS
s = {'male':0,'female':1}
DS['Sex'] = DS['Sex'].apply(lambda x:s[x])
e = {'C':0,'Q':1,'S':2}
DS['Embarked'] = DS['Embarked'].apply(lambda x:e[x])


# Рассмотрим точноть для разных столбцов
colum_target = ['Survived']
colum_param_0 = ['Pclass','Sex','Age']
colum_param_1 = ['Pclass','Sex','Age','SibSp']
colum_param_2 = ['Pclass','Sex','Age','SibSp','Parch']
colum_param_3 = ['Pclass','Sex','Age','SibSp','Embarked']
colum_param_4 = ['Pclass','Sex','Age','Parch']
colum_param_5 = ['Pclass','Sex','Age','Parch','Embarked']
colum_param_6 = ['Pclass','Sex','Age','Embarked']
colum_param_7 = ['Pclass','Sex','Age','SibSp','Parch','Embarked']


C_P = [colum_param_0, colum_param_1, colum_param_2, colum_param_3,␣
 ↪colum_param_4, colum_param_5, colum_param_6, colum_param_7]

y = DS[colum_target]
```

```
# Создание списка для хранения результатов
results = []


for P in C_P:
    X = DS[P]
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=0.2, random_state=42)

    # Создание и обучение модели
    nbais = GaussianNB()
    nbais.fit(X_train, y_train)

    # Прогнозирование
    nbais.predict(X_test)

    # Оценка качества классификации
    accuracy = nbais.score(X_test,y_test)

    results.append((P, accuracy))

# Вывод точности
for i in results:
    print (i)
```

```
(['Pclass', 'Sex', 'Age'], 0.7541899441340782)
(['Pclass', 'Sex', 'Age', 'SibSp'], 0.7653631284916201)
(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch'], 0.7597765363128491)
(['Pclass', 'Sex', 'Age', 'SibSp', 'Embarked'], 0.7597765363128491)
(['Pclass', 'Sex', 'Age', 'Parch'], 0.7486033519553073)
(['Pclass', 'Sex', 'Age', 'Parch', 'Embarked'], 0.7653631284916201)
(['Pclass', 'Sex', 'Age', 'Embarked'], 0.7653631284916201)
(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked'], 0.
 ↪770949720670391)
```

Самая высокая точность в 77% выявляется при использовании всех выбранных нами параметров.