Отчет по лабораторной работе
Метод поиска ближайшего соседа

3 декабря 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 крестикинолики и примере о спаме e-mail сообщений.

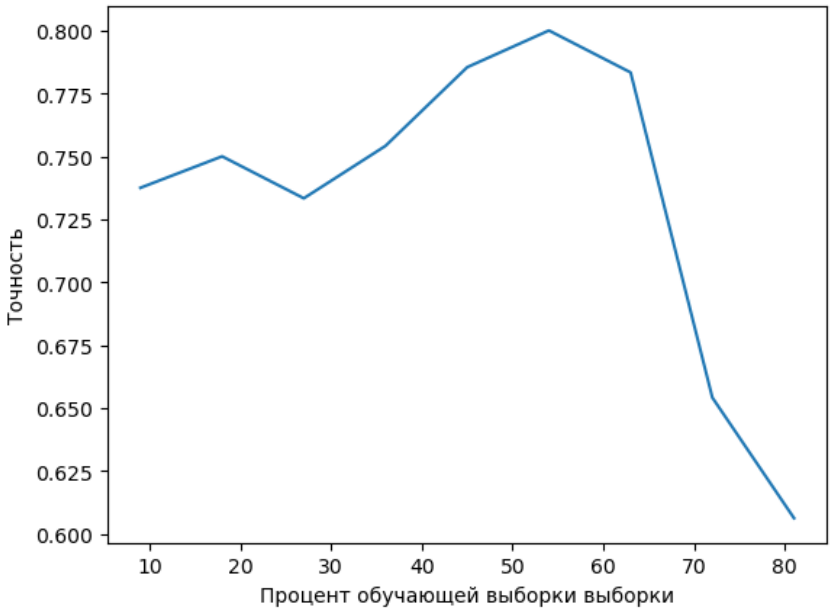
[1]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo
```

Зависимость точности классификации от объёма обучающей выборки:

[2]:
```
# Загружаем DS (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']
```

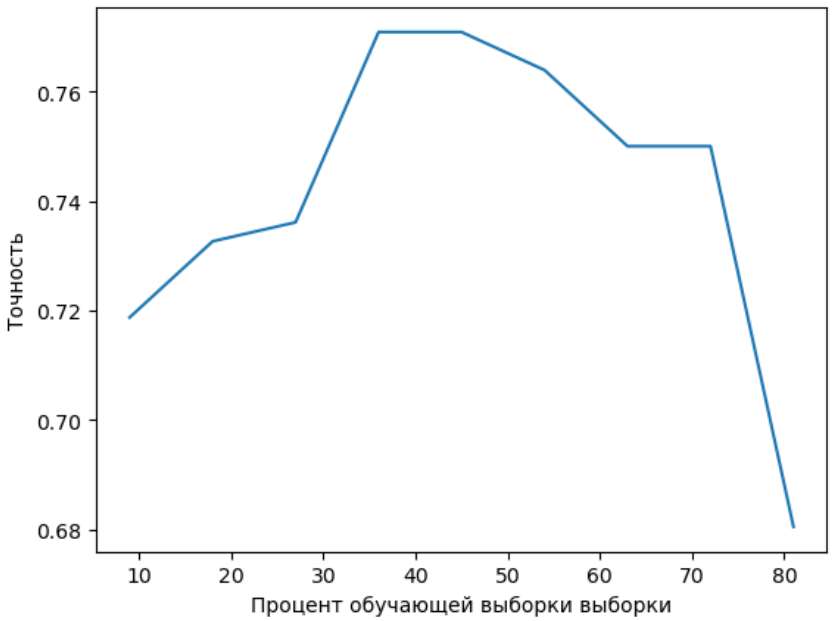
[3]:
```
# Отделяем тестуеую выборку в размере 10 % от всех данных
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
X = X_train
y = y_train

# 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 = 5

# 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 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 k-NN
            knn = KNeighborsClassifier(n_neighbors=2)  # Задайте␣
 ↪параметры k-NN по вашему выбору
            knn.fit(X_train, y_train)

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

             # Оценка качества классификации
            accuracy = knn.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()
```



[4]:
```
# Загружаем DS (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.



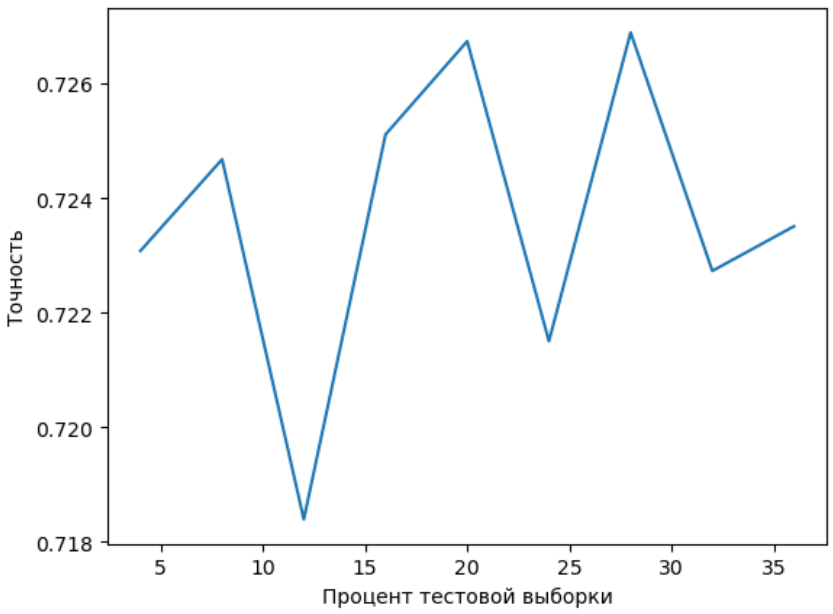
По мере увеличения объёма обучающей выборки точность классификации на тестовых данных сначала улучшается, поскольку новые данные помогают модели лучше обучиться. Однако, в дальнейшем увеличение обучающей выборки приводит к переобучению модели и, как следствие, снижению точности на тестовых данных.

Зависимость точности классификации от количества тестовых данных.

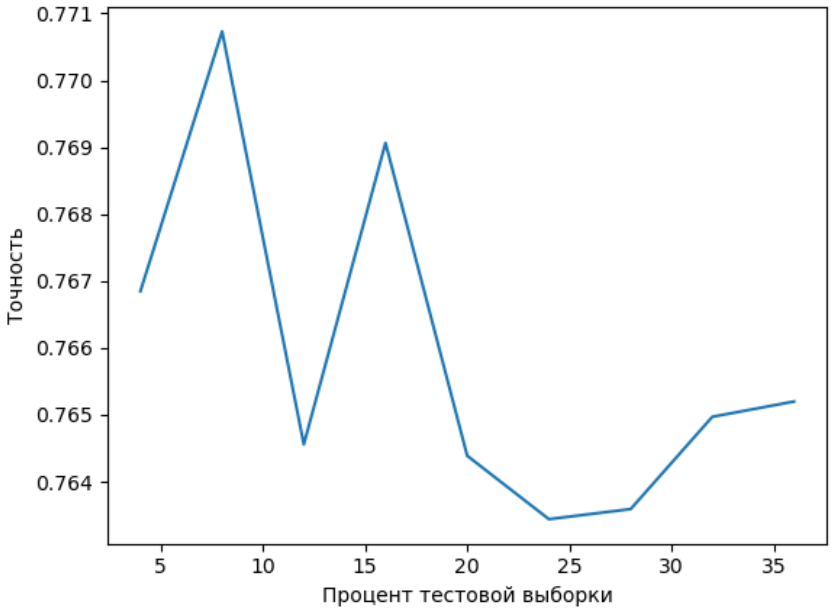
Загружаем DS крестики-нолики как в блоке [2]

[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.
 ↪4, 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 = 15

# 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 k-NN
            knn = KNeighborsClassifier(n_neighbors=2)  # Задайте␣
 ↪параметры k-NN по вашему выбору
            knn.fit(X_train, y_train)

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

             # Оценка качества классификации
            accuracy = knn.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 1%, при этом связь между величинами неочевидна. Такое поведение можно объяснить случайностью выбора данных при формировании тестовой выборки.

Задание №2

Постройте классификатор для обучающего множества Glass, данные которого характеризуются 10-ю признаками: 1. Id number: 1 to 214; 2. RI: показатель преломления; 3. Na: сода (процент содержания в соотвествующем оксиде); 4. Mg; 5. Al; 6. Si; 7. K; 8. Ca; 9. Ba; 10. Fe.

Классы характеризуют тип стекла:

(1) окна зданий, плавильная обработка

(2) окна зданий, не плавильная обработка

(3) автомобильные окна, плавильная обработка

(4) автомобильные окна, не плавильная обработка (нет в базе)

(5) контейнеры

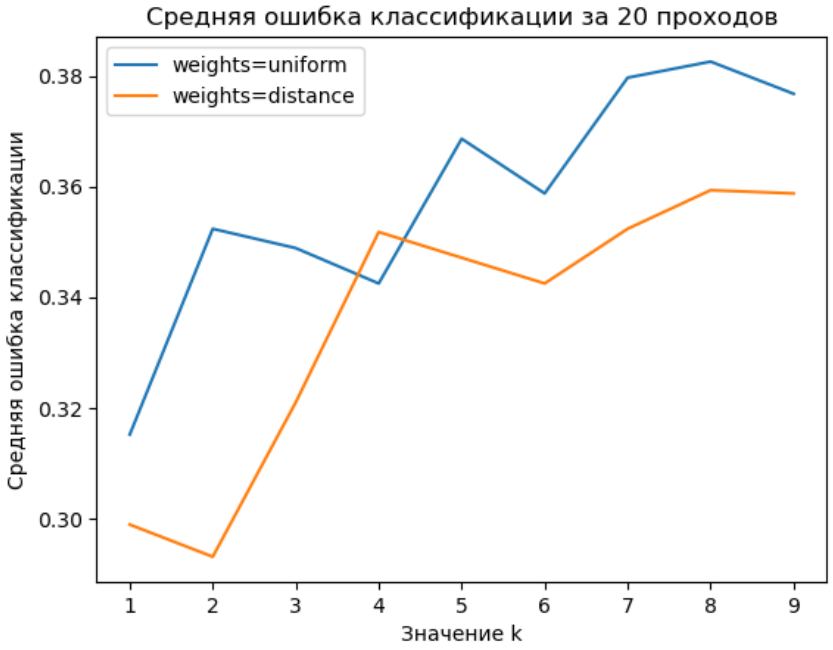
(6) посуда

(7) фары

Посмотрите заголовки признаков и классов. Перед построением классификатора необходимо также удалить первый признак Id number, который не несет никакой информационной нагрузки. Постройте графики зависимости ошибки классификации от значения k и от типа ядра. Исследуйте, как тип метрики расстояния (параметр distance) влияет на точность классификации. Определите, к какому типу стекла относится экземпляр с характеристиками RI =1.516 Na =11.7 Mg =1.01 Al =1.19 Si =72.59 K=0.43 Ca =11.44 Ba =0.02 Fe =0.1 Определите, какой из признаков оказывает наименьшее влияние на определение класса путем последовательного исключения каждого признака.

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

# Загрузка данных и удаление первого столбца "Id number"
glass_data = pd.read_csv('glass.csv')

# Определение признаков (X) и классов (y)
X = glass_data.drop('Type', axis=1)
y = glass_data['Type']

# Список значений k, которые мы хотим исследовать
k_values = range(1, 10)
```

```
# Список типов весов, которые мы хотим исследовать
weight_types = ['uniform', 'distance']

# Словарь для хранения результатов
results = {'uniform': [], 'distance': []}

for weight_type in weight_types:
    for k in k_values:
        X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=0.3, random_state=42)
        # Создание модели k-NN с заданным значением k и типом весов
        knn = KNeighborsClassifier(n_neighbors=k, weights=weight_type)

        # Обучение модели
        knn.fit(X_train, y_train)

        # Предсказание классов на тестовой выборке
        knn.predict(X_test)

        # Вычисление ошибки классификации и сохранение ее в результаты
        error = 1 - knn.score(X_test, y_test)
        results[weight_type].append(error)

# Построение графиков
for weight_type in weight_types:
    plt.plot(k_values, results[weight_type],␣
 ↪label=f'weights={weight_type}')

plt.xlabel('Значение k')
plt.ylabel('Ошибка классификации')
plt.title('Зависимость ошибки классификации от k и типа weights')
plt.legend()
plt.show()
```



Наименьшей ошибке соответствует метрика расстояния 2 при параметре weights='distance'.

[2]:
```
# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪3, random_state=42)

# Создание и обучение классификатора k-NN
knn = KNeighborsClassifier(n_neighbors=2, weights='distance')
knn.fit(X_train, y_train)

# Предсказание классов для тестовых данных
knn.predict(X_test)

# Оценка точности классификации
accuracy = knn.score(X_test,y_test)
print(f'Точность классификации: {accuracy}')

# Определение класса для нового экземпляра стекла
new_sample = np.array([[1.516, 11.7, 1.01, 1.19, 72.59, 0.43, 11.44, 0.
 ↪02, 0.1]])
predicted_class = knn.predict(new_sample)
print(f'Предсказанный класс для нового экземпляра:␣
 ↪{predicted_class[0]}')
```

```
# Исследование влияния признаков на классификацию
feature_names = X.columns
accuracies = []

for feature in feature_names:
    reduced_X = X.drop(feature, axis=1)
    X_train, X_test, y_train, y_test = train_test_split(reduced_X, y,␣
 ↪test_size=0.3, random_state=42)
    knn.fit(X_train, y_train)
    knn.predict(X_test)
    accuracy = knn.score(X_test,y_test)
    print (feature ,accuracy)
```

```
Точность классификации: 0.7692307692307693
Предсказанный класс для нового экземпляра: 5
RI 0.7692307692307693
Na 0.7230769230769231
Mg 0.7076923076923077
Al 0.7076923076923077
Si 0.7076923076923077
K 0.7230769230769231
Ca 0.7384615384615385
Ba 0.7692307692307693
Fe 0.7538461538461538
```

Таким образом экземпляр можно отнести к типу 5 - контейнеры. По результатам исследование Mg, Al, Si оказываю наименьшее влияние на определние класса.

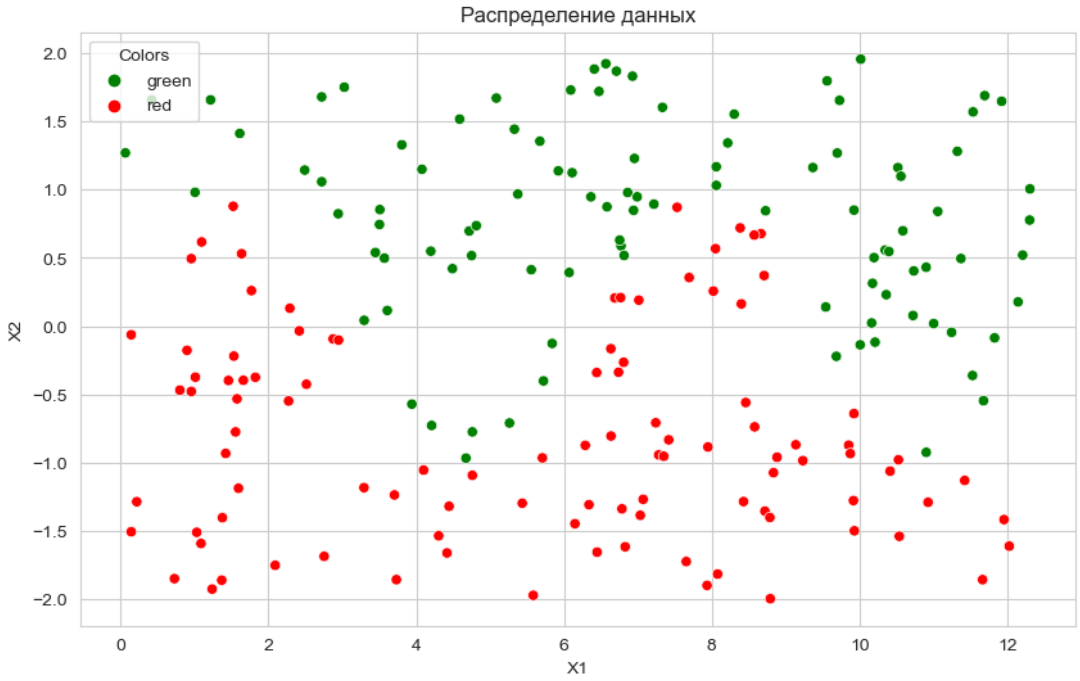
Задание №3

Для построения классификатора используйте заранее сгенерированные обучающие и тестовые выборки, хранящиеся в файлах svmdata4.txt, svmdata4test.txt. Найдите оптимальное значение k, обеспечивающее наименьшую ошибку классификации. Посмотрите, как выглядят данные на графике

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Загрузка данных из файла
data = pd.read_csv('svmdata4.txt', delimiter='\t')

# Разделение данных на признаки и целевую переменную
X_train = data[['X1', 'X2']]
y_train = data['Colors']

# Загрузка данных из файла
data = pd.read_csv('svmdata4test.txt', delimiter='\t')

# Разделение данных на признаки и целевую переменную
X_test = data[['X1', 'X2']]
y_test = data['Colors']

# Список значений k, которые мы хотим исследовать
k_values = range(1, 10)

# Список типов весов, которые мы хотим исследовать
weight_types = ['uniform', 'distance']

# Словарь для хранения результатов
results = {'uniform': [], 'distance': []}

for weight_type in weight_types:
    for k in k_values:
        # Создание модели k-NN с заданным значением k и типом весов
        knn = KNeighborsClassifier(n_neighbors=k, weights=weight_type)

        # Обучение модели
        knn.fit(X_train, y_train)
```

```
        # Предсказание классов на тестовой выборке
        y_pred = knn.predict(X_test)

        # Вычисление ошибки классификации и сохранение ее в результаты
        error = 1 - knn.score(X_test, y_test)
        results[weight_type].append(error)

# Построение графиков
for weight_type in weight_types:
    plt.plot(k_values, results[weight_type],␣
 ↪label=f'weights={weight_type}')

plt.xlabel('Значение k')
plt.ylabel('Ошибка классификации')
plt.title('Зависимость ошибки классификации от k и типа weights')
plt.legend()
plt.show()

# Визуализация данных
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))
sns.scatterplot(x='X1', y='X2', hue='Colors', data=data,␣
 ↪palette={"green": "g", "red": "r"})
plt.title('Распределение данных')
plt.xlabel('X1')
plt.ylabel('X2')
```

Для обучающего множества Colors оптимальное значение числа ближайших соседей k=3 при параметре weights='distance'. В этом случае ошибка классификатора составляет 0.6%.



Задание №4

Разработать классификатор на основе метода ближайших соседей 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
```

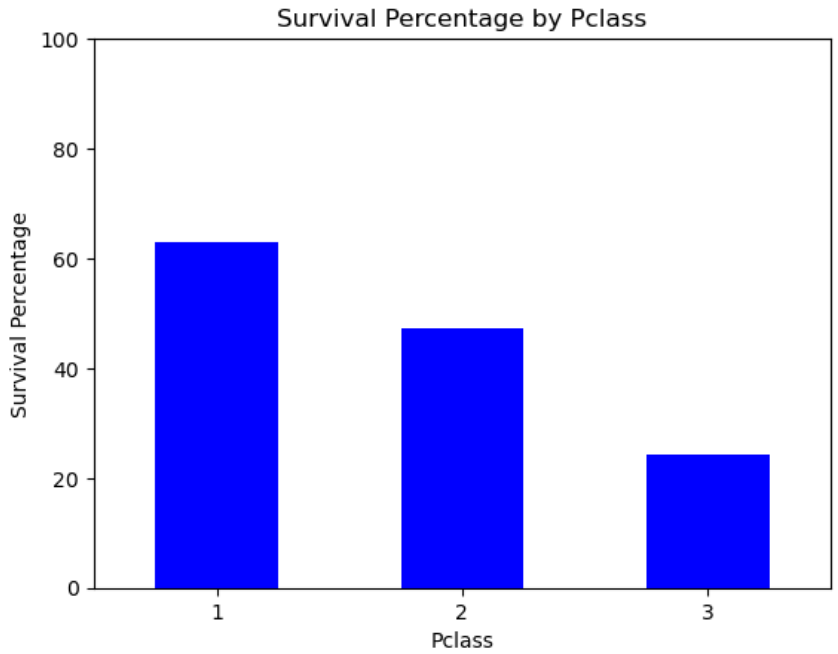
Рассмотрим только параметры Pclass, Sex, Age, SibSp, Parch и Embarked. Не у всех пассажиров заполнен каждый из выбранных нами параметров, исправим это.

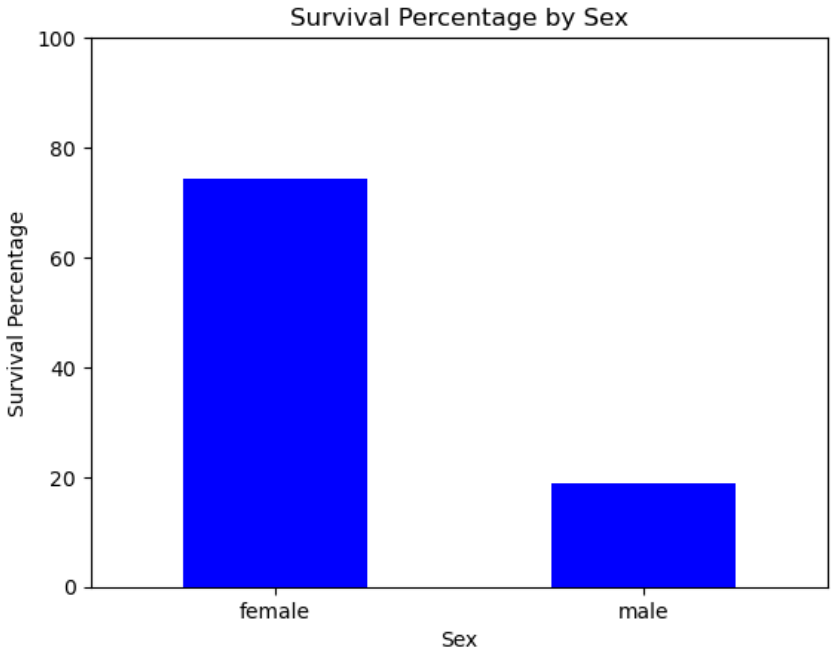
[2]:
```
myImputer = SimpleImputer (strategy= 'median')
DS.Age = myImputer.fit_transform(DS['Age'].values.reshape(-1,1))
myImputer = SimpleImputer (strategy= 'most_frequent')
```

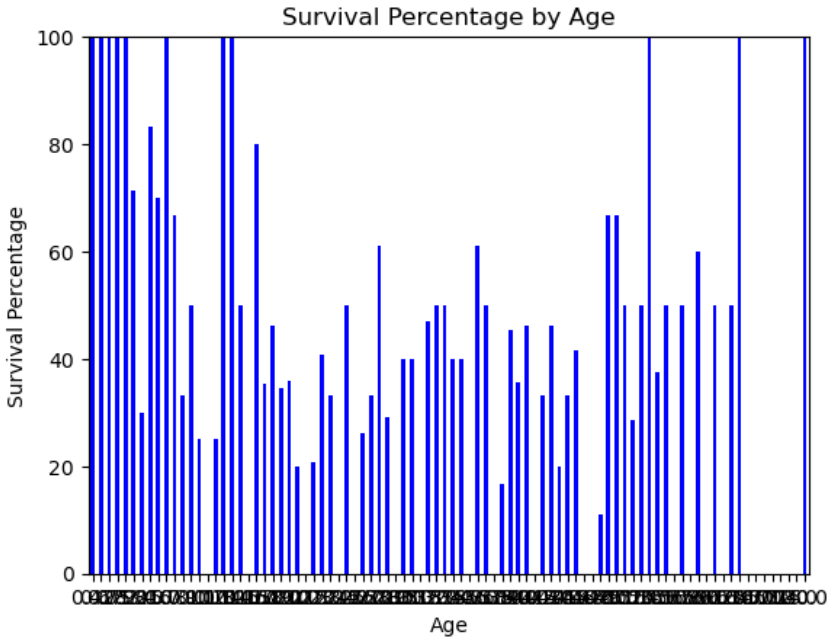
```
DS.Embarked = myImputer.fit_transform(DS['Embarked'].values.
 ↪reshape(-1,1))
DS.info()
```

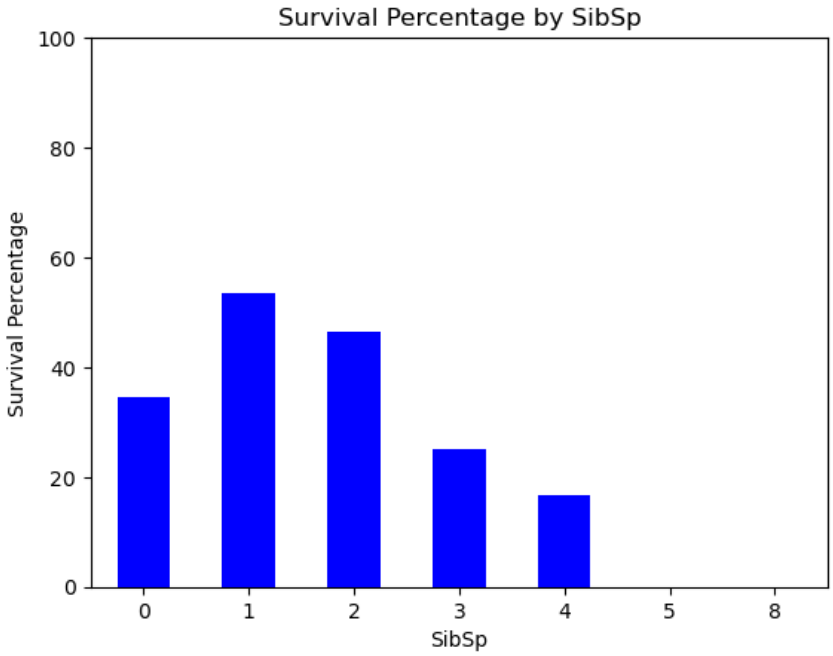
Рассмотрим влияние каждого параметра на выживаемость.

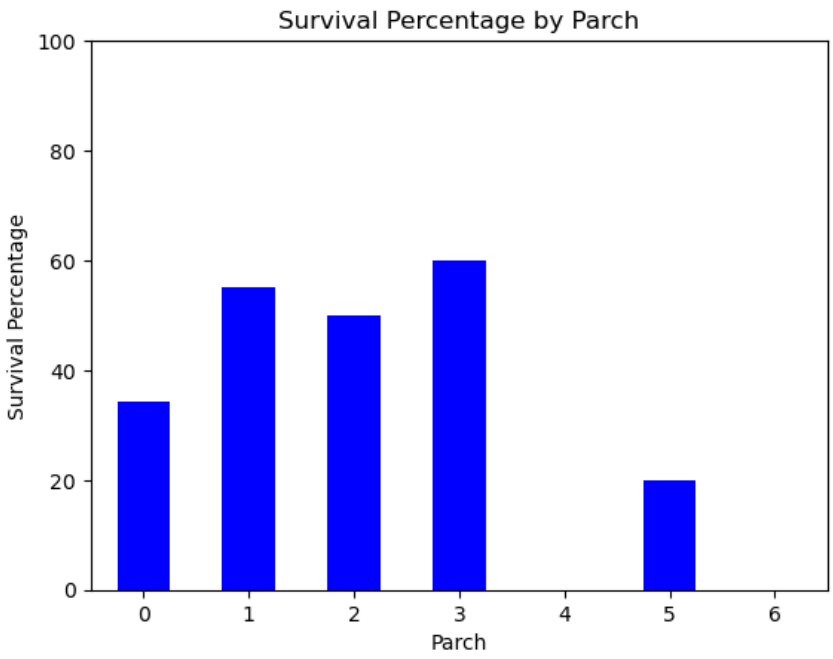
[3]:
```
# Рассчитываем процент выживаемости для каждого параметра
survival_percentages = {}
for column in DS.columns:
    if column not in ['Survived', 'Name', 'PassengerId', 'Ticket',␣
 ↪'Fare', 'Cabin'] :
        survival_percentages[column] = DS.groupby(column)['Survived'].
 ↪mean() * 100

# Создание графиков
for parameter, percentages in survival_percentages.items():
    plt.figure()
    percentages.plot(kind='bar', color='blue')
    plt.xlabel(parameter)
    plt.ylabel('Survival Percentage')
    plt.title(f'Survival Percentage by {parameter}')
    plt.ylim([0, 100])
    plt.xticks(rotation=0)
    plt.show()
```

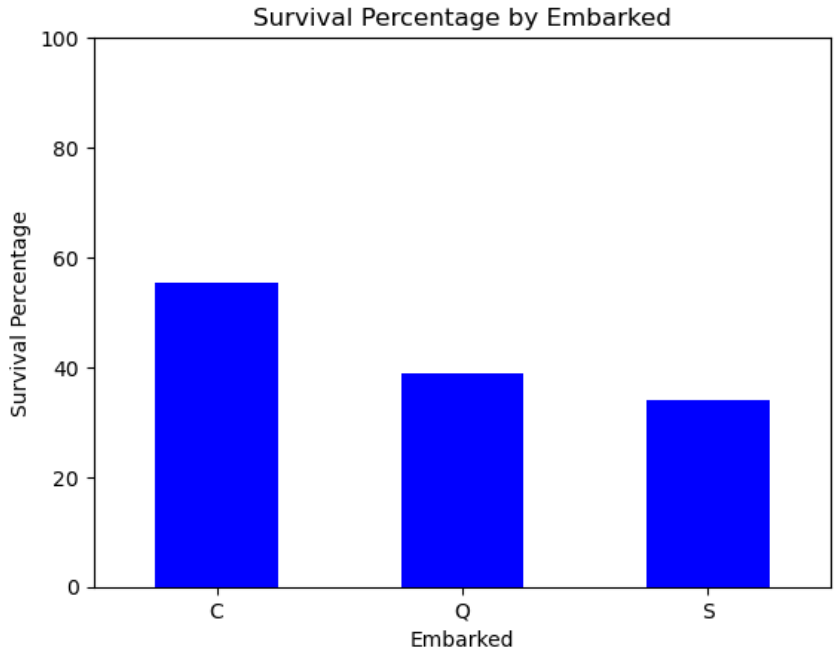



Survival Percentage by Sex
Survival Percentage
100
80
60
40
20
0
female
male
Sex


Survival Percentage by Age
Survival Percentage
100
80
60
40
20
0
Age


Survival Percentage by SibSp
Survival Percentage
100
80
60
40
20
0
0
1
2
3
4
5
8
SibSp


Survival Percentage by Parch
Survival Percentage
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.

[4]:
```
from sklearn.neighbors import KNeighborsClassifier

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_T = [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 = []

# Список значений k, которые мы хотим исследовать
k_values = range(1, 10)

for i in range(15):
    for N in C_T:
        X = DS[N]
        X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=0.4)

        # Создание и обучение модели
        knn = KNeighborsClassifier(n_neighbors=2, weights='distance')
        knn.fit(X_train, y_train)

        # Прогнозирование
        knn.predict(X_test)

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

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

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

for i, (first, second) in enumerate(results):
    results[i] = (first, second/15)

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

```
(['Pclass', 'Sex', 'Age'], 0.7721755368814194)
(['Pclass', 'Sex', 'Age', 'SibSp'], 0.754435107376284)
(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch'], 0.7566760037348275)
(['Pclass', 'Sex', 'Age', 'SibSp', 'Embarked'], 0.7523809523809522)
```

```
(['Pclass', 'Sex', 'Age', 'Parch'], 0.7521942110177403)
(['Pclass', 'Sex', 'Age', 'Parch', 'Embarked'], 0.7432306255835667)
(['Pclass', 'Sex', 'Age', 'Embarked'], 0.7436041083099906)
(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked'], 0.
↪7535014005602243)
```

Самая высокая точность в 77% выявляется при использовании параметров Pclass, Sex и Age.