Лабораторная работа №3. Метрические методы классификации

Цель лабораторной работы: изучение принципов построения информационных систем с использованием метрических методов классификации.

Основные задачи:

- изучение инструментария Python для реализации алгоритмов метрической классификации;
- изучение методов оптимизации параметров метрической классификации;
- освоение модификаций kNN-метода.

Ход выполнения индивидуального задания:

1. Построение модели классификации на основе метода ближайших соседей

1.1 Построение классификатора с заданием K (количества ближайших соседей)

Первым делом импортируем все необходимые библиотеки для выполнения лабораторной работы, а также загрузим наш набор данных с помощью **pandas**:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score

data_path = "https://archive.ics.uci.edu/ml/machine-learning-databases/gla

columns = [
    "Id", "RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "Type"
]
```

```
data = pd.read_csv(data_path, names=columns)
data.drop(columns=["Id"], inplace=True)
```

Признак **Id** удаляем из набора данных из-за ненадобности.

Построим классификатор, используя библиотеку **scikit**:

```
X_train = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
y_train = data['Type']

K = 6

knn = KNeighborsClassifier(n_neighbors=K)
knn.fit(X_train.values, y_train)

X_test = np.array([[1.51634, 14.9, 0.0, 2.3, 73.47, 0.0, 8.33, 1.4, 0.0]])
target = knn.predict(X_test)
print("Данный набор признаков был отнесён к классу:", target)
```

В данном случае мы указываем **новое признаковое описание объекта** (объект отсутствует в обучающей выборке), а алгоритм классификации относит новый объект к **одному из классов**.

В качестве количества ближайших соседей выбрано значение **K = 6**. Выбор данного значения **не обосновывается**. Новым признаковым описанием объекта являются следующие данные:

Признак	Данные
RI	1.51634
Na	14.9
Mg	0
Al	2.3
Si	73.47
K	0
Ca	8.33
Ba	1.4
Fe	0

Как результат, модель отнесла данный набор признаков к **одному из классов**.

```
Данный набор признаков был отнесён к классу: [7]
```

Поменяем данные на другие и запустим модель ещё раз.

Признак	Данные
RI	1.52137
Na	13.84
Mg	3.74
Al	2.7
Si	72.64
K	0.67
Ca	7.53
Ba	0
Fe	0

```
Данный набор признаков был отнесён к классу: [2]
```

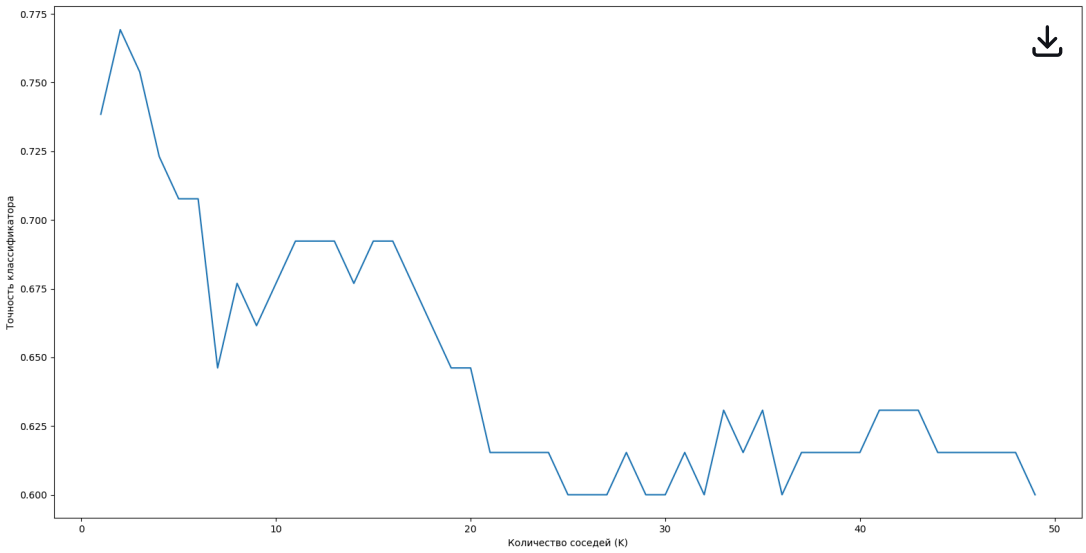
1.2 Вычисление оценки hold-out для различных значений K, а также для различных долей обучающей и тестирующей подвыборок

Для оценки точности классификатора с использованием методики **hold-out** используем следующий код:

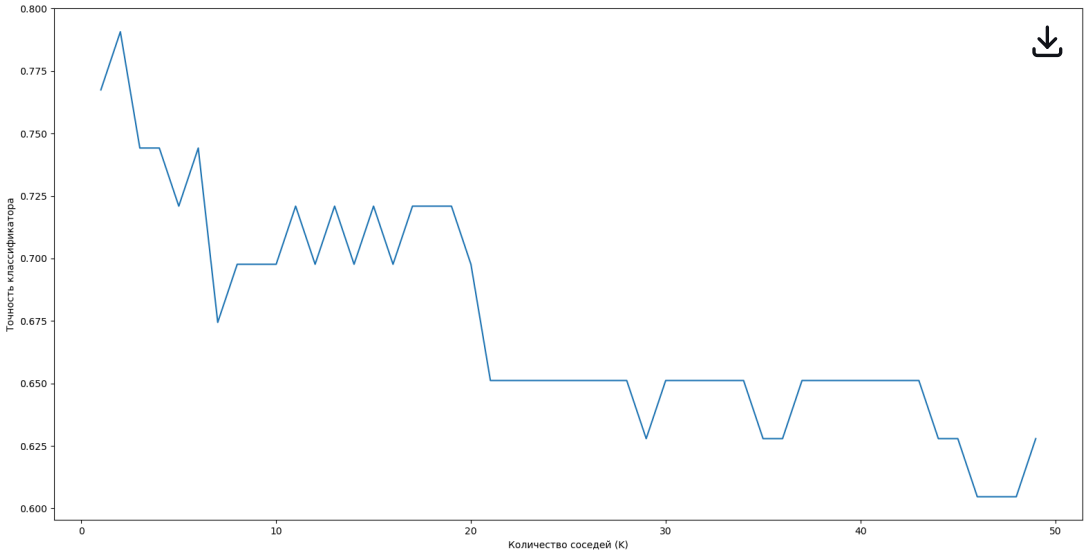
```
X_train, X_holdout, y_train, y_holdout = train_test_split(data[['RI', 'Na'
                                                                data['Type'],
                                                                test_size=0.3,
                                                                random_state=17,
                                                                stratify=data['T

k_list = list(range(1,50))
accuracy_array = []

for K in k_list:
    knn = KNeighborsClassifier(n_neighbors=K)
    knn.fit(X_train, y_train)
    knn_pred = knn.predict(X_holdout)
    accuracy = accuracy_score(y_holdout, knn_pred)
    accuracy_array.append(accuracy)

plt.figure(1)
plt.plot(k_list, accuracy_array)
```

```
plt.xlabel('Количество соседей (K)');
plt.ylabel('Точность классификатора')
```

Оценку точности будем измерять для различных значений **количества ближайших соседей (K),** а именно в диапазоне **от 1 до 49**. В первом случае на тестирующую подвыборку было отведено **30%** данных.



тестирующей подвыборки на **20%**.



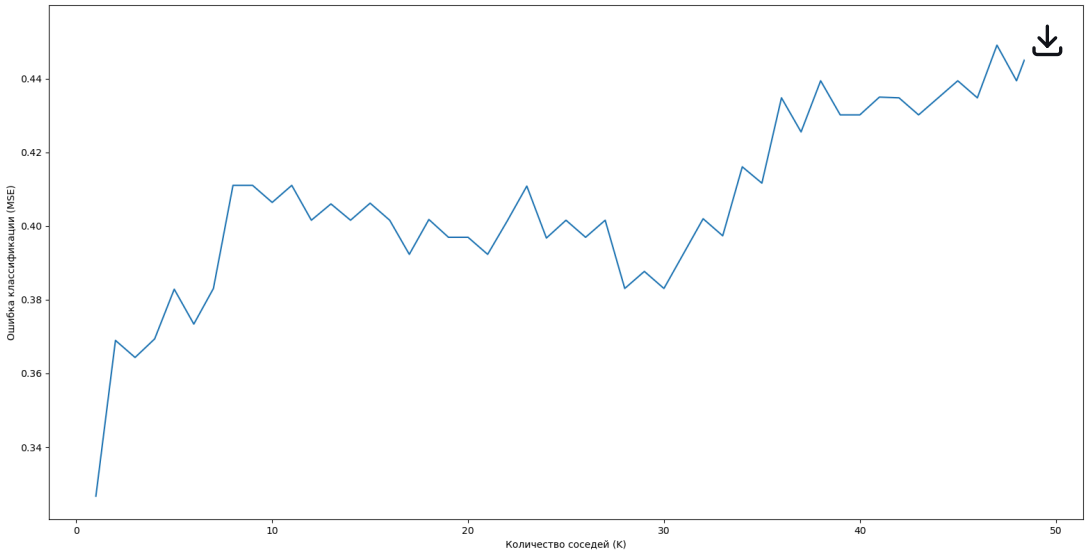
Как можно заметить, точность классификатора на **втором** графике **немного поднялась**.

1.3 Вычисление оценки cross validation для различных значений K, а также для различных значений fold (количества подмножеств при кросс-валидации)

Реализация процедуры выбора оптимального параметра на основе **cross validation error**:

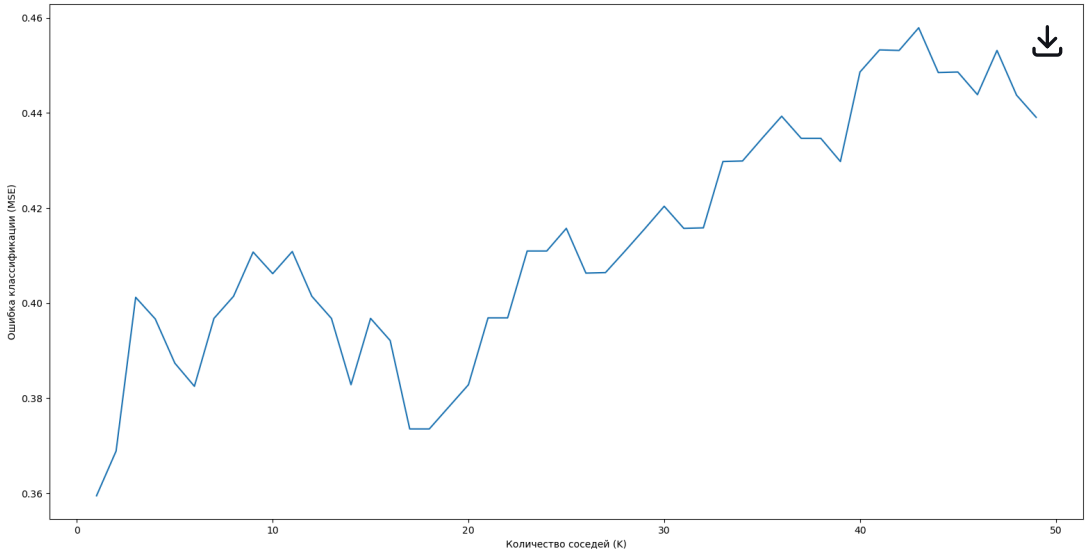
```
cv_scores = []

for K in k_list:
    knn = KNeighborsClassifier(n_neighbors=K)
    scores = cross_val_score(knn,
                             data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca'
                             data['Type'],
                             cv=9,
                             scoring='accuracy')
    cv_scores.append(scores.mean())

MSE = [1-x for x in cv_scores]

plt.figure(2)
plt.plot(k_list, MSE)
plt.xlabel('Количество соседей (K)');
plt.ylabel('Ошибка классификации (MSE)')

k_min = min(MSE)

all_k_min = []
for i in range(len(MSE)):
    if MSE[i] <= k_min:
        all_k_min.append(k_list[i])

print('Оптимальные значения K: ', all_k_min)
```

График **ошибки классификации (MSE)**, а также **оптимальные значения K** при значении подмножества при кросс-валидации **равным 9**:



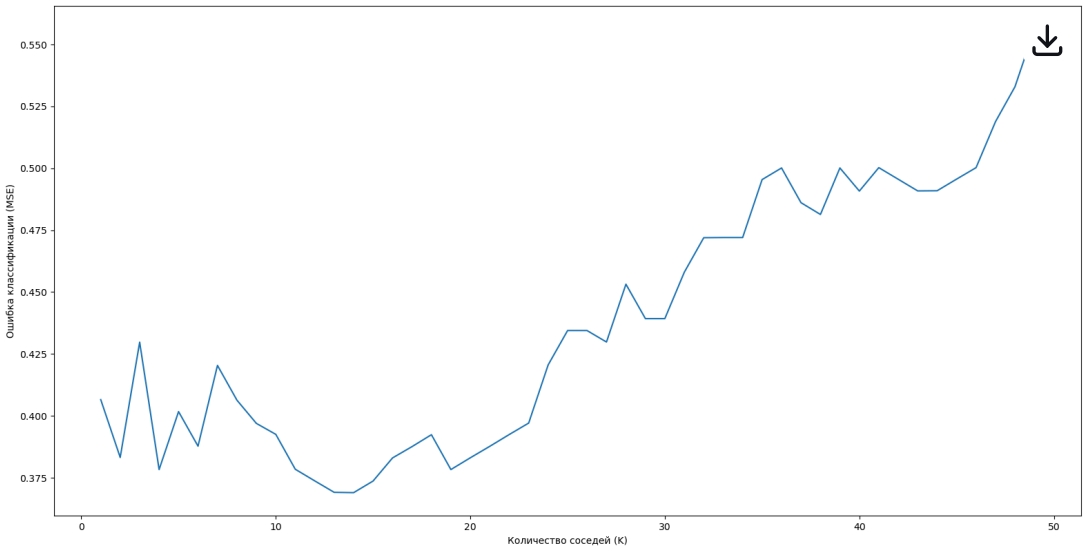

```
Оптимальные значения K:  [1]
```

График **ошибки классификации (MSE)**, а также **оптимальные значения K** при значении подмножества при кросс-валидации **равным 5**:




```
Оптимальные значения K:  [1]
```

График **ошибки классификации (MSE)**, а также **оптимальные значения K** при значении подмножества при кросс-валидации **равным 3**:




```
Оптимальные значения K:  [14]
```

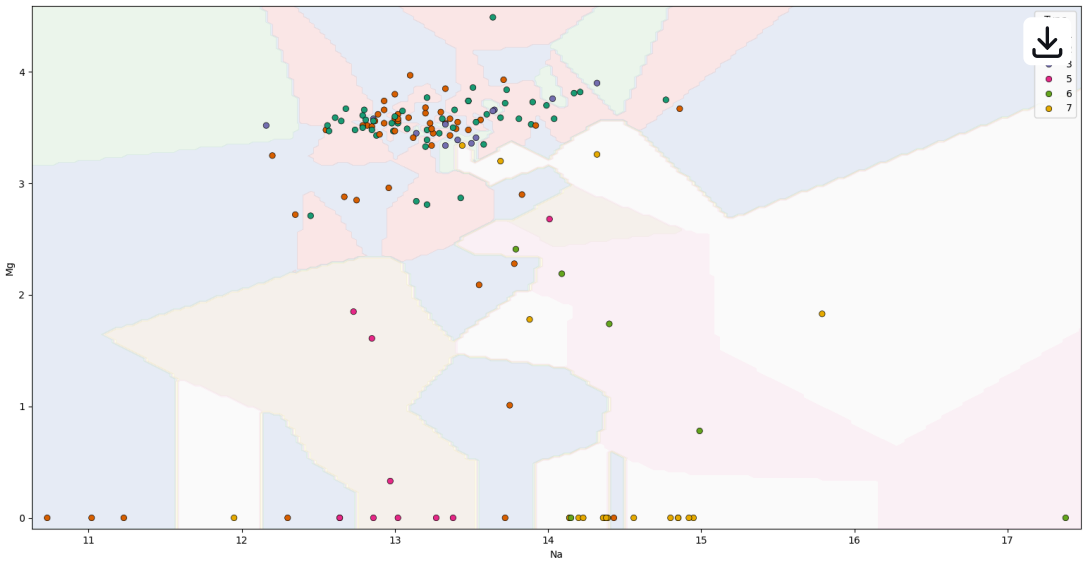
1.4 Демонстрация полученного классификатора при оптимальном значении K

В качестве оптимального количества **K** выберем значение **равное 1**. Для визуализации **решающих границ и распределения классов** для kNN был использован следующий код:

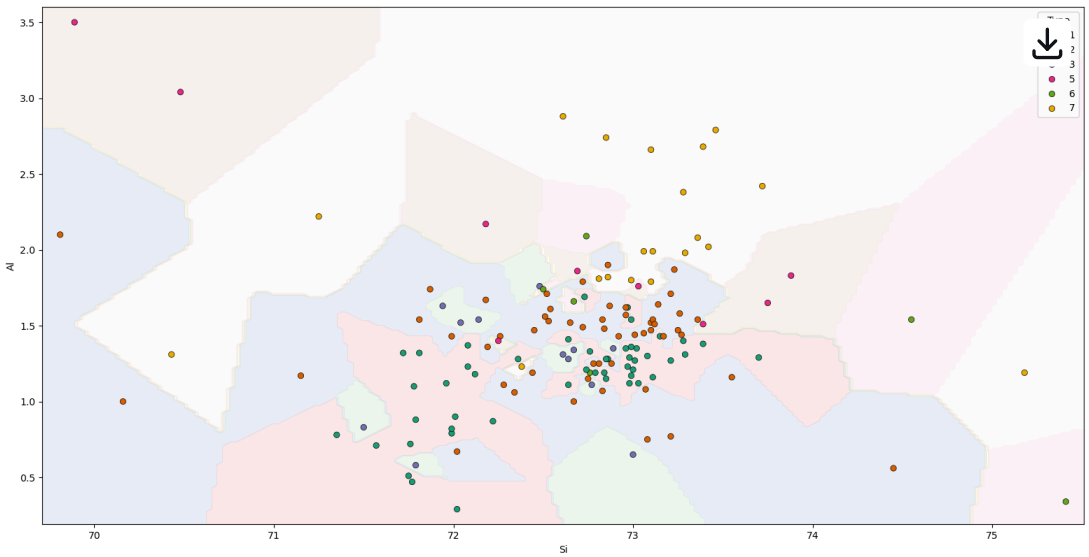
```
X = data[['Na', 'Mg']]
y = data['Type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

x_min, x_max = X.iloc[:, 0].min() - 0.1, X.iloc[:, 0].max() + 0.1
```

```
y_min, y_max = X.iloc[:, 1].min() - 0.1, X.iloc[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200), np.linspace(y_min, y_

Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(3)
plt.contourf(xx, yy, Z, alpha=0.3, cmap='Pastel1')
sns.scatterplot(x=X_train.iloc[:, 0], y=X_train.iloc[:, 1], hue=y_train, p
plt.xlabel('RI')
plt.ylabel('Na')
```

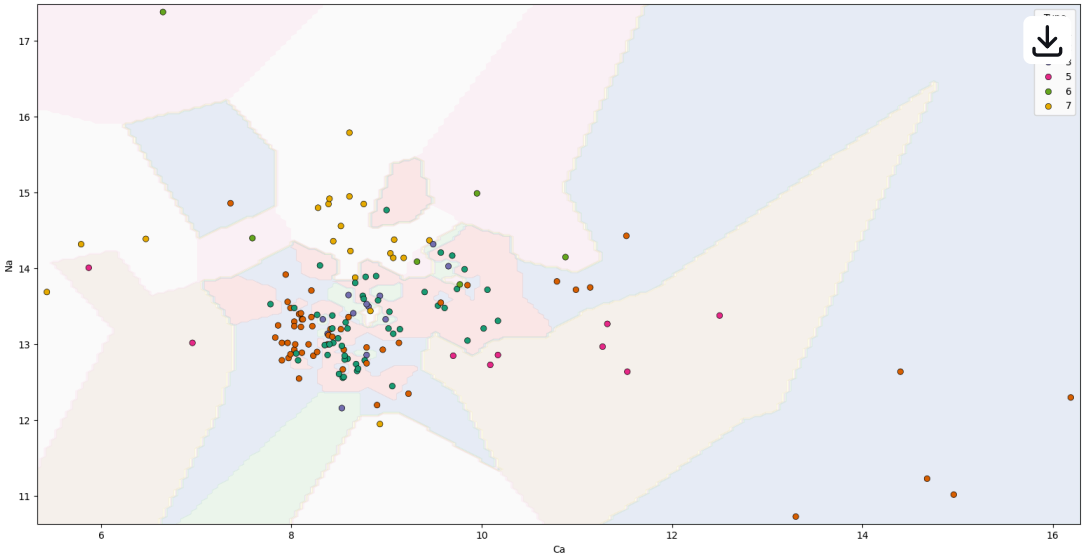
Классификационные границы для **натрия** и **магния**:



Классификационные границы для **кремния** и **алюминия**:



Классификационные границы для **кальция** и **натрия**:



Контрольные вопросы

1. Метод ближайшего соседа — это один из самых простых и интуитивно понятных методов классификации. В этом методе для классификации нового объекта используется только ближайший к нему объект из обучающего множества.

   Особенности:

   - Прост в реализации
   - Не требует обучения — достаточно иметь обучающее множество для классификации
   - Может быть чувствителен к выбросам

   Метод k ближайших соседей является обобщением метода 1-NN. Вместо одного ближайшего соседа в методе kNN учитываются k ближайших соседей, и классификация основывается на большинстве классов среди этих k соседей.

   Особенности:

   - Более устойчив к выбросам по сравнению с 1-NN
   - Как и в 1-NN, для классификации не требуется явного этапа обучения, все данные обрабатываются в процессе классификации
   - Для больших данных метод может быть вычислительно затратным
2. **Этапы реализации метода kNN:**
   - Для каждого объекта из обучающего множества вычисляется расстояние до нового объекта
   - Выбираются k ближайших соседей
   - Новый объект классифицируется по принципу голосования: назначается тот класс, который наиболее часто встречается среди k ближайших соседей

3. Наилучший результат можно получить с помощью комбинированного подхода: тестирование различных значений k, использование перекрестной проверки и ориентирование на характер данных.
4. Метод **парзеновского окна** — это один из методов, который используется для оценки плотности распределения данных. Этот метод основывается на идее построения непрерывной оценки плотности вероятности для случайной величины, используя данные выборки. Он часто используется в задачах классификации и регрессии.
5. **Метод потенциальных функций** — это метод, используемый в теории оптимизации и некоторых областях машинного обучения, в частности в задачах классификации. В контексте классификации этот метод используется для преобразования задачи о принятии решения в задачу оптимизации, где нужно минимизировать или максимизировать определённую потенциальную функцию.
6. Число соседей, метрика расстояния, веса соседей