|  |  |  |
| --- | --- | --- |
|  | Изображение выглядит как зарисовка, рисунок, символ, эмблема  Автоматически созданное описание |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| **Институт комплексной безопасности и специального приборостроения**  Кафедра КБ-14 «Технологии искусственного интеллекта и анализ данных»  Дисциплина «Введение в машинное обучение» | | |

**Отчёт**

«Практическая работа №4»

Выполнил:

Студент 3 курса

Группа БСБО-06-22

Шифр 22Б1240

Белявцева Екатерина Александровна

Москва, 2024

# Задание

Разработать KNN-классификатор (метод ближайших соседей) для набора данных. В качестве языка программирования использовать Pyhton. Разбить выборку на обучающую и тестовую. Обучить, проверить качество классификатора на обучающей и тестовой выборках: рассчитать общую точность, Recall, Precision, F1-measure.

Оценить число опорных векторов. Рассмотреть различные типы ядра (линейное, RBF, полиномиальное, сигмоидное (тангенсальное)), различные сочетания значений параметра регуляризации С и параметров ядра.

Перебор по сетке (grid search).

# Ход работы

Для выполнения первой части был использован язык программирования Python в приложении PyCharm. Для корректной работы кода был импортирован текстовый документ «Ирисы». В коде выборка была разбита на обучающую и тестовую, была обучена и проверено качество классификатора (были рассчитаны общая точность, Recall, Precision и F1-measure для обучающей и тестовой выборок). После этого код был немного изменён и добавлен перебор по сетке (grid search). Листинг кода на данном этапе представлен ниже (см. Листинг 1).

Листинг 1. Код для модели KNN

|  |
| --- |
| import pandas as pd from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report  # Загружаем данные из текстового файла data = pd.read\_csv('Ирисы.txt', delimiter='\t', encoding='windows-1251')  # Заменяем запятые на точки в числовых данных data = data.replace(',', '.', regex=True)  # Разделение данных на признаки (X) и метки (y) X = data.iloc[:, :-1] # Все столбцы, кроме последнего — это признаки y = data.iloc[:, -1] # Последний столбец — метки классов  # Преобразуем все числовые данные к float X = X.astype(float)  # Разделяем данные на обучающую и тестовую выборки X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  # Функция для оценки модели kNN def evaluate\_knn():  # Определяем параметры для перебора по сетке  param\_grid = {  'n\_neighbors': [3, 5, 7, 9], # Количество соседей  'weights': ['uniform', 'distance'], # Способы взвешивания  'metric': ['euclidean', 'manhattan', 'minkowski'] # Метрики расстояния  }   # Создаем модель kNN  knn\_classifier = KNeighborsClassifier()   # Настраиваем перебор по сетке  grid\_search = GridSearchCV(estimator=knn\_classifier, param\_grid=param\_grid,  scoring='accuracy', cv=5, n\_jobs=-1)   # Обучаем модель  grid\_search.fit(X\_train, y\_train)   # Получаем наилучшие параметры и модель  best\_params = grid\_search.best\_params\_  best\_model = grid\_search.best\_estimator\_   # Предсказания для обучающей выборки  y\_train\_pred = best\_model.predict(X\_train)  # Метрики для обучающей выборки  train\_accuracy = accuracy\_score(y\_train, y\_train\_pred)  train\_precision = precision\_score(y\_train, y\_train\_pred, average='weighted', zero\_division=0)  train\_recall = recall\_score(y\_train, y\_train\_pred, average='weighted', zero\_division=0)  train\_f1 = f1\_score(y\_train, y\_train\_pred, average='weighted', zero\_division=0)   # Предсказания для тестовой выборки  y\_test\_pred = best\_model.predict(X\_test)  # Метрики для тестовой выборки  test\_accuracy = accuracy\_score(y\_test, y\_test\_pred)  test\_precision = precision\_score(y\_test, y\_test\_pred, average='weighted', zero\_division=0)  test\_recall = recall\_score(y\_test, y\_test\_pred, average='weighted', zero\_division=0)  test\_f1 = f1\_score(y\_test, y\_test\_pred, average='weighted', zero\_division=0)   # Выводим результаты  print("Наилучшие параметры:", best\_params)  print("\nМетрики на обучающей выборке:")  print(f"Общая точность: {train\_accuracy:.2f}")  print(f"Средняя точность (Precision): {train\_precision:.2f}")  print(f"Средняя полнота (Recall): {train\_recall:.2f}")  print(f"Средняя F1-метрика: {train\_f1:.2f}")   print("\nМетрики на тестовой выборке:")  print(f"Общая точность: {test\_accuracy:.2f}")  print(f"Средняя точность (Precision): {test\_precision:.2f}")  print(f"Средняя полнота (Recall): {test\_recall:.2f}")  print(f"Средняя F1-метрика: {test\_f1:.2f}")   print("\nОтчет по классификации для тестовой выборки:\n", classification\_report(y\_test, y\_test\_pred))  print("=" \* 50)  # Запускаем оценку модели kNN evaluate\_knn() |

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание

Рисунок 1. Вывод кода

Для алгоритма KNN понятие «опорных векторов» не существует, поскольку это не метод опорных векторов, как в SVM. Вместо этого, KNN использует все точки данных, а при классификации для нового объекта анализируются ближайшие к нему точки. В контексте KNN можно анализировать качество модели, точность, Precision, Recall и F1-метрику, но не число опорных векторов.

Также для KNN невозможно рассмотреть и настроить различные типы ядер и сочетания значений параметра регуляризации С и параметров ядра. В этом алгоритме нет этих понятий, так как они используются только в SVM. Для KNN работа с ядрами не применима, так как модель основана на расстояниях между точками данных, и ядра здесь просто не задействуются.

Параметр регуляризации С также невозможно рассмотреть, так как этот параметр регулирует баланс между правильной классификацией обучающего набора и максимизацией разделяющей гиперплоскости, а в KNN нет разделяющей гиперплоскости, поэтому и параметр С невозможно оценить.