Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет программной инженерии и компьютерной техники**

**Отчёт по лабораторной работе №4**

по дисциплине

«Системы искусственного интеллекта»

*Выполнил:*

Студент группы P3333

Анисимов Максим Дмитриевич

*Преподаватель:*

Авдюшина Анна Евгеньевна



Санкт-Петербург, 2024

**Задание**

Выбор датасета:

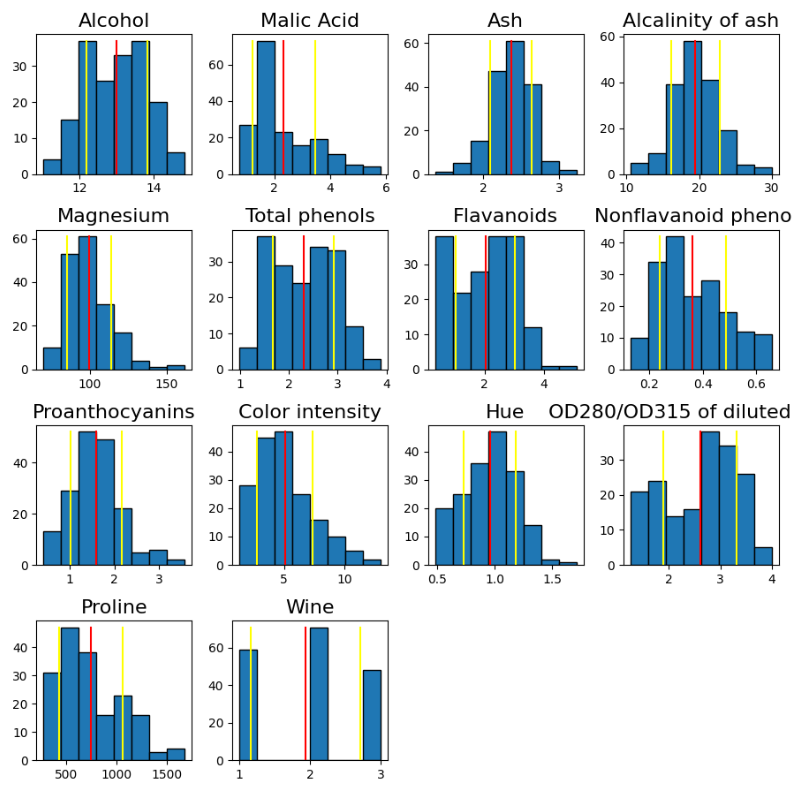
Четный номер в группе - Датасет о вине

Проведите предварительную обработку данных, включая обработку отсутствующих значений, кодирование категориальных признаков и масштабирование

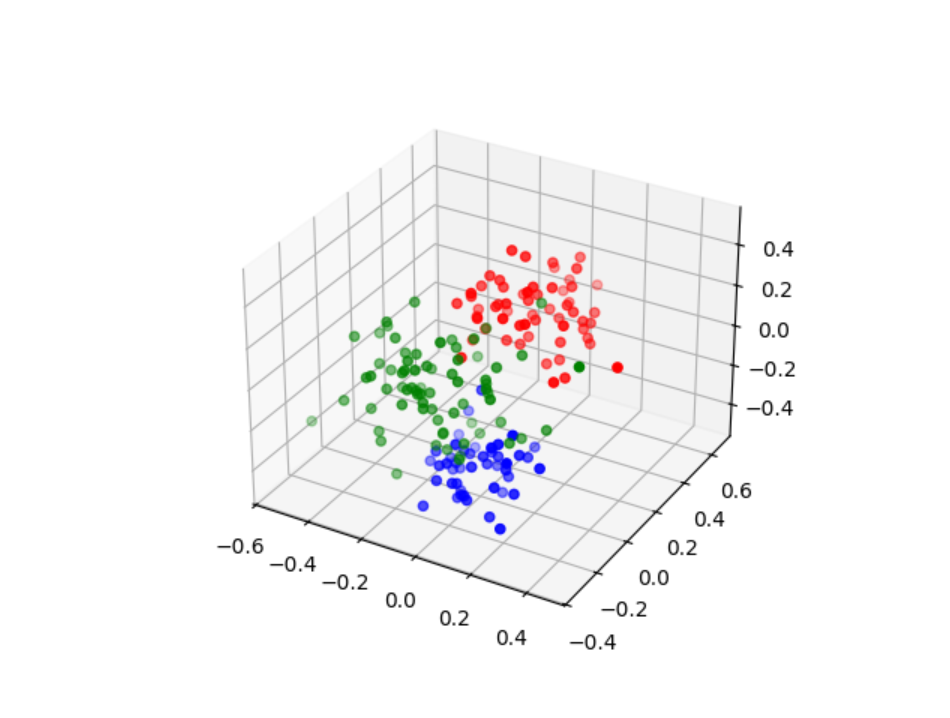
if not data.isna().any().any():  
 print("В данных нет пустых значений.")  
else:  
 data.dropna(inplace=True)  
 print("Пустые значения был удалены.")  
  
# Категориальных признаков нет, поэтому ничего с ними не делаем  
  
# Стандартизация (масштабирование) признаков  
from sklearn.preprocessing import StandardScaler  
  
scaler = StandardScaler()  
X = data.drop("Wine", axis=1)  
X = pd.DataFrame(scaler.fit\_transform(X), columns=X.columns)  
y = data["Wine"]

В данных нет пустых значений.

Получите и визуализируйте (графически) статистику по датасету (включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили), постройте 3d-визуализацию признаков.



X = data.drop(columns=["Wine"]) # X содержит все столбцы данных, кроме "Wine"  
y = data["Wine"] # y содержит столбец "Wine"  
# Нормализация  
X = (X - X.mean()) / (X.max() - X.min())  
  
# Выбор 3 случайных столбцов после нормализации  
r\_columns = list(X.columns)  
random.shuffle(r\_columns)  
r\_columns = r\_columns[:3]  
  
colors = {1: "red", 2: "green", 3: "blue"}  
df = X[r\_columns].join(y)  
# выбранные столбцы с целевой переменной  
print(df.columns)  
ax = plt.axes(projection="3d")  
  
for i in list(set(y)):  
 values = df.loc[df["Wine"] == i]  
 ax.scatter(values[df.columns[0]], values[df.columns[1]], values[df.columns[2]], color=colors[i])  
  
plt.show()

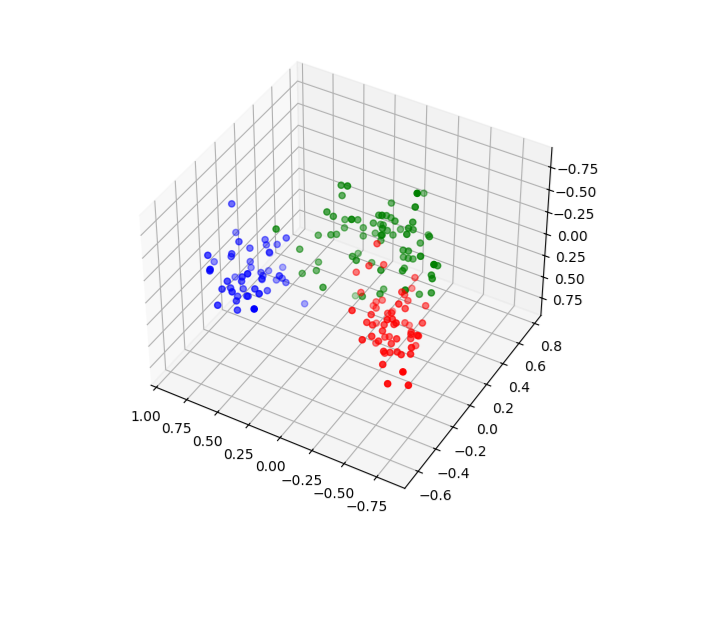


Index(['Ash', 'Proline', 'OD280/OD315 of diluted wines', 'Wine'], dtype='object')

Теперь посмотрим на тот же график, но сформируем новые признаки методом главных компонент.

# Преобразование данных в матрицу и вычисление ковариационной матрицы  
X\_mat = np.asarray(X)  
cov = np.matmul(X\_mat.T, X\_mat)  
# Вычисление собственных значений и собственных векторов  
eig\_vals, eig\_vectors = np.linalg.eig(cov)  
eig\_vectors = eig\_vectors.T  
# сортировка в порядке убывания  
eig = [[eig\_vals[i], eig\_vectors[i]] for i in range(len(eig\_vals))]  
eig.sort(key = lambda t: t[0], reverse = True)  
# первые три вектора  
pca = [eig[i][1] for i in range(3)]  
X\_reduced = np.matmul(pca, X\_mat.T).T  
X\_reduced = pd.DataFrame(X\_reduced, columns=["pca1", "pca2", "pca3"])  
# Объединение преобразованных данных с целевой переменной  
df = X\_reduced.join(y)  
  
print(df.columns)  
  
fig = plt.figure(figsize=(20, 20))  
  
ax = fig.add\_subplot(1, 3, 2, projection="3d")  
  
for i in list(set(y)):  
 values = df.loc[df["Wine"] == i]  
 ax.scatter(values[df.columns[0]], values[df.columns[1]], values[df.columns[2]], color = colors[i])  
  
ax.view\_init(-140, 60)  
  
plt.show()

Index(['pca1', 'pca2', 'pca3', 'Wine'], dtype='object')



Реализуйте метод k-ближайших соседей без использования сторонних библиотек, кроме NumPy и Pandas.

def euclidean\_distance(x1, x2):  
 return np.sqrt(np.sum((x1 - x2) \*\* 2))  
  
# X: признаки  
# y: метки классов  
# query\_point: точка для которой выполняется классификация  
# k: количество ближайших соседей, которые будут учтены  
def k\_nearest\_neighbors(X, y, query\_point, k):  
 # Вычисляем расстояние между точками в многомерном пространстве  
 distances = [euclidean\_distance(query\_point, x) for x in X]  
  
 # Получаем точки с наименьшими расстояниями  
 k\_indices = np.argsort(distances)[:k]  
 k\_nearest\_labels = [y[i] for i in k\_indices]  
 # Берем самые часто встречающиеся  
 most\_common = np.bincount(k\_nearest\_labels).argmax()  
  
 return most\_common

Постройте две модели k-NN с различными наборами признаков:

* Модель 1: Признаки случайно отбираются .
* Модель 2: Фиксированный набор признаков, который выбирается заранее.
* # Функция для получения матриц ошибок  
  def get\_confusion\_matrices(k\_values, X\_model, y):  
   confusion\_matrices\_model = []  
   X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_model, y, test\_size=0.2, random\_state=42)  
   # Для каждой точки в тестовом наборе выполняется классификация  
   for k in k\_values:  
   y\_pred\_model = [k\_nearest\_neighbors(X\_train.values, y\_train.values, x, k) for x in X\_test.values]  
   confusion\_matrix = np.zeros((3, 3), dtype=int)  
   # Строится матрица ошибок, которая считает, сколько точек было правильно и неправильно классифицировано для каждого класса  
   for i in range(len(y\_test)):  
   confusion\_matrix[y\_test.iloc[i] - 1][y\_pred\_model[i] - 1] += 1  
   confusion\_matrices\_model.append(confusion\_matrix)  
    
   return confusion\_matrices\_model

Для каждой модели проведите оценку на тестовом наборе данных при разных значениях k. Выберите несколько различных значений k, например, k=3, k=5, k=10, и т. д. Постройте матрицу ошибок.

k\_values = [3, 5, 10]  
  
# Модель 1  
random\_feature\_indices = np.random.choice(X.shape[1], size=3, replace=False)  
X\_model1 = X.iloc[:, random\_feature\_indices]  
  
# Модель 2  
fixed\_feature\_indices = [0, 1, 5]  
X\_model2 = X.iloc[:, fixed\_feature\_indices]  
# Выбираются случайные признаков из матрицы признаков X.  
confusion\_matrices\_model1 = get\_confusion\_matrices(k\_values, X\_model1, y)  
# Фиксированный набор признаков  
confusion\_matrices\_model2 = get\_confusion\_matrices(k\_values, X\_model2, y)  
  
# Вывод матриц ошибок  
for k, confusion\_matrix in zip(k\_values, confusion\_matrices\_model1):  
 print(f"Матрицы ошибок для Модели 1 с k={k}:\n", confusion\_matrix)  
  
print("\n---------------------------------\n")  
  
for k, confusion\_matrix in zip(k\_values, confusion\_matrices\_model2):  
 print(f"Матрицы ошибок для Модели 2 с k={k}:\n", confusion\_matrix)

Матрицы ошибок для Модели 1 с k=3:

[[14 0 0]

[ 2 12 0]

[ 2 1 5]]

Матрицы ошибок для Модели 1 с k=5:

[[14 0 0]

[ 2 12 0]

[ 2 1 5]]

Матрицы ошибок для Модели 1 с k=10:

[[14 0 0]

[ 2 12 0]

[ 2 1 5]]

---------------------------------

Матрицы ошибок для Модели 2 с k=3:

[[14 0 0]

[ 1 11 2]

[ 0 0 8]]

Матрицы ошибок для Модели 2 с k=5:

[[14 0 0]

[ 1 11 2]

[ 0 0 8]]

Матрицы ошибок для Модели 2 с k=10:

[[14 0 0]

[ 1 11 2]

[ 0 0 8]]