**Курсовая работа состоит из 5 разделов/заданий по разделам.**

[1. изучение предметной области и постановка задачи (регрессии, классификации, кластеризации) 1](#_Toc135037085)

[2. сбор и обработка данных 1](#_Toc135037086)

[3. разведочный анализ данных (качественная визуализация данных и осмысление графиков, вычисление описательных статистик, и проч.) 3](#_Toc135037087)

[4. построение статистических моделей и алгоритмов их машинного 8](#_Toc135037088)

[обучения, включая разработку искусственных нейронных сетей 8](#_Toc135037089)

[5. интерпретация и использование модели машинного обучения (концепция продукта данных или принятие решения на основе обобщения данных) 19](#_Toc135037090)

# изучение предметной области и постановка задачи (регрессии, классификации, кластеризации)

Предметной областью является изучение семейного статуса молодых швейцарцев в течение 16 лет. Задачей может быть прогнозирование изменения семейного статуса на основе различных факторов, таких как пол, год рождения, родной язык, образование, профессия, а также оценка влияния семейного статуса на другие аспекты жизни, например, доход, здоровье или качество жизни. Для достижения данной задачи могут использоваться методы регрессии или классификации. Также возможны методы кластеризации, чтобы выявить группы людей с похожими характеристиками и семейным статусом.

# сбор и обработка данных

Датасет:

Формат: фрейм данных с 2000 строками, 16 переменными состояния, 1 переменной id и 7 ковариатами и 2 весовыми переменными.

Описание: набор данных содержит (в столбцах с 10 по 25) последовательности состояний семейной жизни в возрасте от 15 до 30 лет (длина последовательности равна 16). Последовательности представляют собой выборку из 2000 последовательностей, созданных на основе биографического обзора SHP. В него включены только лица, которым на момент проведения опроса было не менее 30 лет. Набор данных biofam описывает жизненный путь 2000 человек, родившихся в период с 1909 по 1972 год.

Состояния (a15-a30), пронумерованные от 0 до 7, определяются на основе комбинации пяти основных состояний:

0 = "Parent"

1 = "Left"

2 = "Married"

3 = "Left+Marr"

4 = "Child"

5 = "Left+Child"

6 = "Left+Marr+Child"

7 = "Divorced"

Содержит колонки:

sex (пол)

birthyr (год рождения)

nat\_1\_02 (первое гражданство)

plingu02 (язык анкеты)

p02r01 (религия)

p02r04 (религиозное участие)

cspfaj (социальный статус отца)

cspmoj (социальный статус матери)

Две дополнительные весовые переменные вставлены только в иллюстративных целях (поскольку biofam является подвыборкой исходных данных, эти весовые коэффициенты не адаптированы к фактическим данным).:

wp00tbgp (увеличение веса для населения Швейцарии)

wp00tbgs (веса, соответствующие размеру выборки)

Для начала импортируем необходимые библиотеки и загрузим датасет biofam:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read\_csv("biofam.csv", delimiter = ';')

Переименуем столбцы для удобства чтения:

df = df.rename(columns={'idhous': 'ID', 'sex': 'Gender', 'birthyr': 'BirthYear',

'nat\_1\_02': 'Nationality', 'plingu02': 'Language', 'p02r01': 'RelationshipStatus',

'p02r04': 'LivingArrangement', 'cspfaj': 'FatherOccupation', 'cspmoj': 'MotherOccupation',

'a15': 'Year15', 'a16': 'Year16', 'a17': 'Year17', 'a18': 'Year18', 'a19': 'Year19',

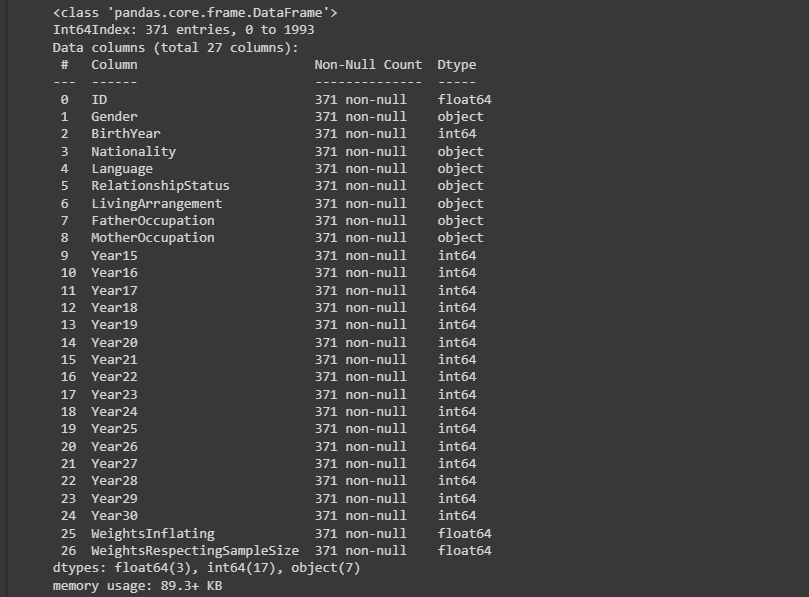
'a20': 'Year20', 'a21': 'Year21', 'a22': 'Year22', 'a23': 'Year23', 'a24': 'Year24',

'a25': 'Year25', 'a26': 'Year26', 'a27': 'Year27', 'a28': 'Year28', 'a29': 'Year29',

'a30': 'Year30', 'wp00tbgp': 'WeightsInflating', 'wp00tbgs': 'WeightsRespectingSampleSize'})

Убедимся, что все столбцы загрузились корректно и в нужном формате:

df.info()



Для удобства исследования удалим строки с пропущенными значениями:

df.dropna(inplace=True)

# 3. разведочный анализ данных (качественная визуализация данных и осмысление графиков, вычисление описательных статистик, и проч.)

Показательные статистики:

df.describe()



Качественная визуализация переменных:

sns.countplot(x='Gender', data=df)

plt.title('Gender Distribution')

plt.show()

sns.histplot(x='BirthYear', data=df, bins=10)

plt.title('Birth Year Distribution')

plt.show()

sns.countplot(x='Nationality', hue='Gender', data=df)

plt.title('Nationality and Gender Distribution')

plt.show()

sns.countplot(x='LivingArrangement', data=df)

plt.title('Living Arrangement Distribution')

plt.show()

sns.displot(x='WeightsInflating', data=df, kde=True)

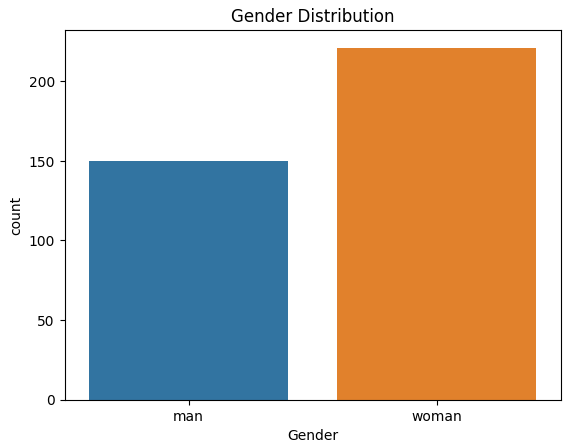
plt.title('Weights inflating to the Swiss population')

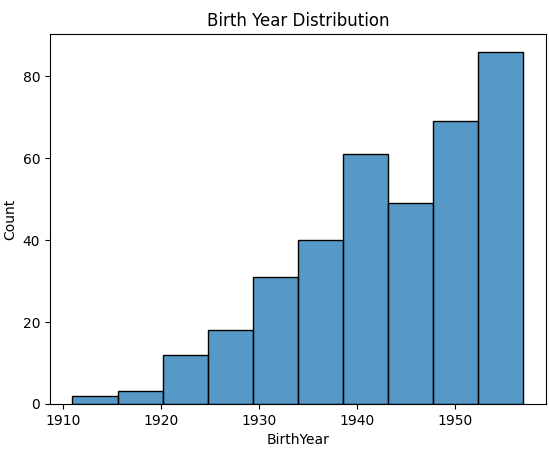
plt.show()

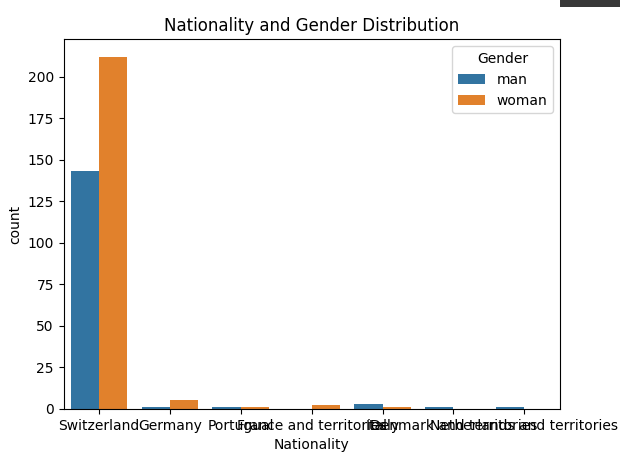
sns.displot(x='WeightsRespectingSampleSize', data=df, kde=True)

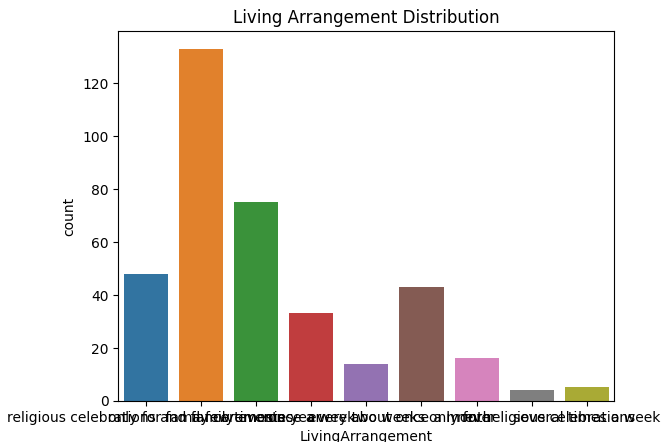
plt.title('Weights respecting sample size')

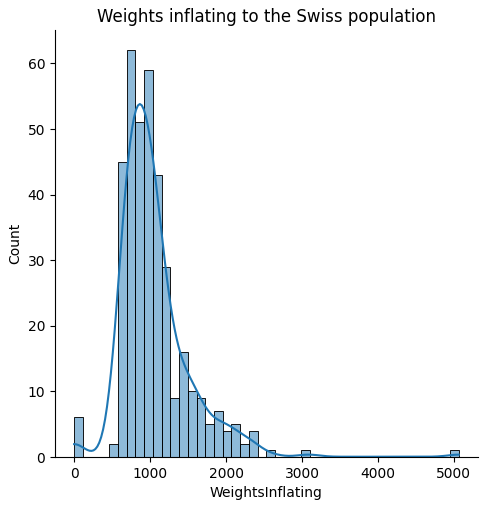
plt.show()

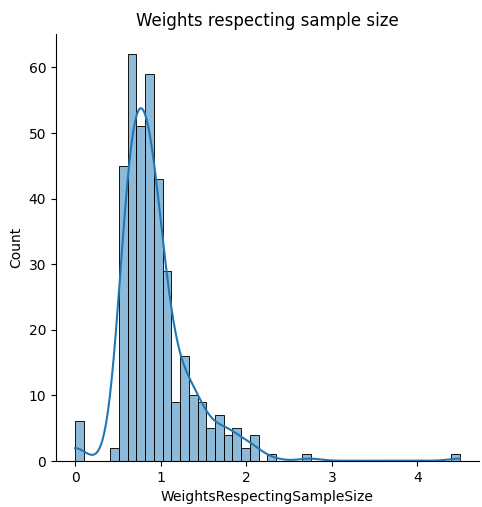










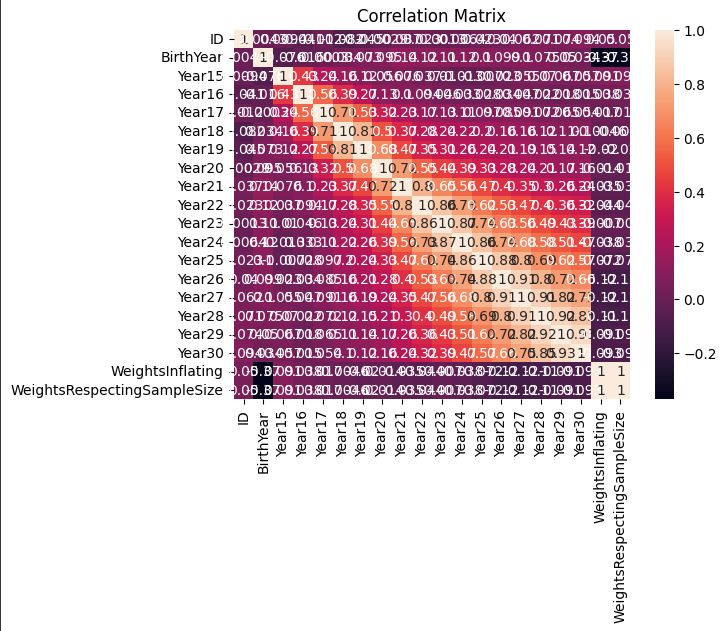


Корреляционная матрица:

sns.heatmap(df.corr(), annot=True)

plt.title('Correlation Matrix')

plt.show()



# 4. построение статистических моделей и алгоритмов их машинного

# обучения, включая разработку искусственных нейронных сетей

Модели линейной регрессии и случайного леса:

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from sklearn.model\_selection import learning\_curve

from sklearn.metrics import make\_scorer

from sklearn import linear\_model, ensemble

# Загрузка данных

training\_data = pd.read\_csv("biofam.csv", delimiter = ';')

# Проверка размерности и типов данных

training\_data.shape

training\_data.info()

# Удаление пустых значений

training\_data = training\_data.dropna()

# Преобразование категориальных признаков в бинарные

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(handle\_unknown='ignore')

training\_points\_encoded = encoder.fit\_transform(training\_data[['nat\_1\_02']])

# Объединение закодированных признаков с числовыми

training\_points = np.hstack((training\_data[['a16', 'a29']].values, training\_points\_encoded.toarray()))

# Определение целевой переменной

training\_values = training\_data['birthyr'].values

# Создание моделей линейной регрессии и случайного леса

linear\_regression\_model = linear\_model.LinearRegression()

random\_forest\_model = ensemble.RandomForestRegressor(n\_estimators=100)

# Обучение моделей на тренировочных данных

linear\_regression\_model.fit(training\_points, training\_values)

random\_forest\_model.fit(training\_points, training\_values)

# Оценка качества моделей на кросс-валидации

from sklearn.model\_selection import cross\_val\_score

linear\_regression\_scores = cross\_val\_score(linear\_regression\_model, training\_points, training\_values, cv=10, scoring='r2')

random\_forest\_scores = cross\_val\_score(random\_forest\_model, training\_points, training\_values, cv=10, scoring='r2')

print('Linear Regression CV scores:', linear\_regression\_scores)

print('Random Forest CV scores:', random\_forest\_scores)

print('Average Linear Regression CV score:', np.mean(linear\_regression\_scores))

print('Average Random Forest CV score:', np.mean(random\_forest\_scores))

# Построение графика предсказаний линейной регрессии и истинных значений

predicted\_values = linear\_regression\_model.predict(training\_points)

plt.scatter(training\_values, predicted\_values)

plt.xlabel("True Values")

plt.ylabel("Predictions")

plt.show()

# Построение графика важности признаков в модели случайного леса

importances = random\_forest\_model.feature\_importances\_

std = np.std([tree.feature\_importances\_ for tree in random\_forest\_model.estimators\_], axis=0)

indices = np.argsort(importances)[::-1]

plt.figure()

plt.title("Feature importances")

plt.bar(range(training\_points.shape[1]), importances[indices], yerr=std[indices])

plt.xticks(range(training\_points.shape[1]), indices)

plt.xlim([-1, training\_points.shape[1]])

plt.show()

Данный код выполняет задачу регрессии, а именно предсказание года рождения (признак 'birthyr') на основе других признаков из датасета "biofam". В данном случае используется модель линейной регрессии и модель случайного леса для обучения и предсказания. Также осуществляется оценка качества модели с помощью кросс-валидации и визуализация построенных моделей с помощью графиков.

Модель PCA:

# Загрузка необходимых библиотек и датасета

import pandas as pd

import numpy as np

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

data = pd.read\_csv("biofam.csv", delimiter = ';')

# Проведение предварительного анализа данных и очистки данных

data = data.replace('?', np.nan) # замена пропущенных значений на NaN

data = data.dropna() # удаление строк с пропущенными значениями

# Выбор переменных для PCA

variables = ['a16', 'a20', 'a22', 'a25', 'a27', 'a30']

# Нормализация данных

norm\_data = (data[variables] - data[variables].mean()) / data[variables].std()

# Выполнение PCA

pca = PCA()

pca.fit(norm\_data)

# Выделение главных компонент

components = pca.components\_

scores = pca.transform(norm\_data)

# Производим интерпретацию полученных результатов

print('Доля объясненной дисперсии:', pca.explained\_variance\_ratio\_)

print('Вклад каждой переменной:', components)

# Выбор двух коррелирующих параметров

corr\_variables = ['a16', 'a30']

# Выделение выбранных параметров

corr\_data = norm\_data[corr\_variables]

# Выполнение PCA для выбранных параметров

corr\_pca = PCA()

corr\_pca.fit(corr\_data)

# Выделение главных компонент для выбранных параметров

corr\_components = corr\_pca.components\_

corr\_scores = corr\_pca.transform(corr\_data)

# Производим интерпретацию полученных результатов

print('Доля объясненной дисперсии для выбранных параметров:', corr\_pca.explained\_variance\_ratio\_)

print('Вклад каждой переменной для выбранных параметров:', corr\_components)

# Сравнение результатов

print('Разница в доле объясненной дисперсии:', pca.explained\_variance\_ratio\_.sum() - corr\_pca.explained\_variance\_ratio\_.sum())

# График зависимости доли объясненной дисперсии и количества главных компонент

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

plt.xlabel('Number of Components')

plt.ylabel('Variance (%)')

plt.title('График зависимости доли объясненной дисперсии и количества главных компонент')

plt.show()

# График распределения главных компонент

plt.scatter(scores[:, 0], scores[:, 1])

plt.xlabel('1st Principal Component')

plt.ylabel('2nd Principal Component')

plt.title('График распределения главных компонент')

plt.show()

Данный код выполняет задачу анализа главных компонентов (PCA) на датасете "biofam". Он содержит следующие действия:

1. Загрузка необходимых библиотек и датасета.

2. Предварительный анализ данных и очистка данных. В данном случае, проводится замена пропущенных значений на NaN и удаление строк с пропущенными значениями.

3. Выбор переменных для PCA. В данном случае, выбираются переменные 'a16', 'a20', 'a22', 'a25', 'a27', и 'a30'.

4. Нормализация данных.

5. Выполнение PCA.

6. Выделение главных компонент и интерпретация результатов (доля объясненной дисперсии и вклад каждой переменной).

7. Выбор двух коррелирующих параметров ('a16' и 'a30').

8. Выделение выбранных параметров и выполнение PCA для этих параметров.

9. Выделение главных компонент для выбранных параметров и интерпретация результатов.

10. Сравнение результатов для полного набора переменных и выбранных параметров.

11. Построение графика зависимости доли объясненной дисперсии и количества главных компонент для полного набора переменных.

12. Построение графика распределения главных компонент для полного набора переменных.

Кроме того, данный код позволяет исследовать взаимосвязь между переменными в датасете и определить главные компоненты, которые наиболее влиятельны на общую переменную. Он также может помочь выделить наиболее значимые переменные для решения задачи машинного обучения, что может повысить точность и качество построенной модели.

Модель дерева решений:

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.tree import export\_graphviz

import graphviz

# Загрузка данных

df = pd.read\_csv("biofam.csv", delimiter = ';')

# Удаление столбца id

df.drop(columns=['idhous'], inplace=True)

df.drop(columns=['nat\_1\_02'], inplace=True)

df.drop(columns=['plingu02'], inplace=True)

df.drop(columns=['p02r01'], inplace=True)

df.drop(columns=['p02r04'], inplace=True)df.drop(columns=['cspfaj'], inplace=True)

df.drop(columns=['cspmoj'], inplace=True)

# Замена значений столбца sex

df['sex'].replace({0: 'woman', 1: 'man'}, inplace=True)

# Разделение на признаки и целевую переменную

X = df.drop(columns=['sex'])

y = df['sex']

# Создание модели дерева решений

clf = DecisionTreeClassifier(random\_state=42)

# Тренировка модели

clf.fit(X, y)

# Создание файла .dot для построения графа дерева решений

dot\_file = export\_graphviz(clf, out\_file='tree.dot', feature\_names=X.columns, class\_names=y.unique(), filled=True)

# Отображение графа дерева решений

with open('tree.dot') as f:

dot\_graph = f.read()

graphviz.Source(dot\_graph)

Данный код выполняет задачу классификации, а именно предсказание пола человека (признак 'sex') на основе других признаков из датасета biofam с использованием модели дерева решений.

1. Для этого вначале загружаются данные из файла 'biofam.csv' и удаляют столбцы, которые не будут участвовать в модели.

2. Заменяются категориальные значения столбца 'sex' на соответствующие значения ('man', 'woman').

3. Выделяются признаки и целевая переменная.

4. Создается модель дерева решений с параметром random\_state 42.

5. Модель обучается на тренировочных данных.

6. Создается файл .dot для построения графа дерева решений.

7. Визуализация полученного графа с помощью библиотеки graphviz.

8. Полученное дерево решений можно использовать для предсказания пола на новых данных.

Ансамблевый подход (бустинг):

import pandas as pd

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Загрузка данных

df = pd.read\_csv("biofam.csv", delimiter = ';')

# Удаление столбца id

df.drop(columns=['idhous'], inplace=True)

df.drop(columns=['nat\_1\_02'], inplace=True)

df.drop(columns=['plingu02'], inplace=True)

df.drop(columns=['p02r01'], inplace=True)

df.drop(columns=['p02r04'], inplace=True)

df.drop(columns=['cspfaj'], inplace=True)

df.drop(columns=['cspmoj'], inplace=True)

# Замена значений столбца sex

df['sex'].replace({0: 'woman', 1: 'man'}, inplace=True)

# Разделение на признаки и целевую переменную

X = df.drop(columns=['sex'])

y = df['sex']

# Разделение выборки на обучающую и тестовую

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Создание модели бустинга

model = GradientBoostingClassifier(random\_state=42)

# Обучение модели

model.fit(X\_train, y\_train)

# Предсказание на тестовой выборке

y\_pred = model.predict(X\_test)

# Оценка качества модели

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy: {:.2f}%".format(accuracy\*100))

# График важности признаков

importance = model.feature\_importances\_

plt.bar([x for x in X.columns], importance)

plt.xticks(rotation='vertical')

plt.show()

Данный код выполняет классификацию с использованием ансамблевого подхода – бустинга, позже производит интерпретацию полученного результата.

Кластеризация:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

# Загрузка данных

df = pd.read\_csv("biofam.csv", delimiter = ';')

# Удаление столбца id

df.drop(columns=['idhous'], inplace=True)

df.drop(columns=['nat\_1\_02'], inplace=True)

df.drop(columns=['plingu02'], inplace=True)

df.drop(columns=['p02r01'], inplace=True)

df.drop(columns=['p02r04'], inplace=True)

df.drop(columns=['cspfaj'], inplace=True)

df.drop(columns=['cspmoj'], inplace=True)

# Замена значений столбца sex

df['sex'].replace({0: 'woman', 1: 'man'}, inplace=True)

# Обработка пропущенных значений

df.replace(' ', np.nan, inplace=True)

df.dropna(inplace=True)

# Нормализация данных

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(df.drop(columns=['sex']))

# Кластеризация методом k-means

def kmeans\_f(X, n\_clusters):

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)

kmeans.fit(X)

return kmeans

# Вычисление индекса силуэта для каждого числа кластеров

def silhouette(X):

silhouette\_scores = []

for n\_clusters in range(2, 11):

kmeans = kmeans\_f(X, n\_clusters)

silhouette\_scores.append(silhouette\_score(X, kmeans.labels\_))

return silhouette\_scores

# Визуализация индекса силуэта в зависимости от числа кластеров

silhouette\_scores = silhouette(X\_scaled)

plt.plot(range(2, 11), silhouette\_scores)

plt.xlabel('Number of clusters')

plt.ylabel('Silhouette score')

plt.show()

# Кластеризация методом k-means

n\_clusters = 4

kmeans = kmeans\_f(X\_scaled, n\_clusters)

# Добавление меток кластеров в датафрейм

df['cluster'] = kmeans.labels\_

# Анализ кластеров

means = df.groupby(['cluster']).mean()

print(means)

# График анализа кластеров

fig, ax = plt.subplots(1, len(df.columns)-1, figsize=(20,5))

for i, col in enumerate(df.columns[:-1]):

for cluster in range(n\_clusters):

ax[i].hist(df[df['cluster'] == cluster][col], alpha=0.5, label=f'Cluster {cluster}')

ax[i].set\_title(col)

plt.legend()

plt.show()

Данный код выполняет кластеризацию данных с использованием k-means, оценивает качество кластеризации в зависимости от числа итераций и значения k, а также дает рекомендации по выбору числа кластеров.

Нейронная сеть:

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

import matplotlib.pyplot as plt

# Загрузка данных

data = pd.read\_csv("biofam.csv", delimiter = ';')

# преобразуем текстовые значения в числа

data['sex'] = data['sex'].apply(lambda x: 1 if x=='man' else 0)

data.drop(columns=['idhous'], inplace=True)

data.drop(columns=['nat\_1\_02'], inplace=True)

data.drop(columns=['plingu02'], inplace=True)

data.drop(columns=['p02r01'], inplace=True)

data.drop(columns=['p02r04'], inplace=True)

data.drop(columns=['cspfaj'], inplace=True)

data.drop(columns=['cspmoj'], inplace=True)

# Выборки для модели

X = data.iloc[:, :-1]

y = data.iloc[:, -1]

# Кодирование категориальных признаков

ct = ColumnTransformer([('encoder', OneHotEncoder(), [0, 2, 7])], remainder='passthrough')

X = ct.fit\_transform(X)

# Масштабирование данных

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Разделение данных на тренировочный и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Создание модели

model = tf.keras.models.Sequential([

tf.keras.layers.Dense(units=16, activation='relu', input\_shape=[X.shape[1]]),

tf.keras.layers.Dense(units=8, activation='relu'),

tf.keras.layers.Dense(units=1, activation='sigmoid')

])

# Компиляция модели

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Обучение модели

model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test))

# Оценка качества модели на тестовых данных

\_, accuracy = model.evaluate(X\_test, y\_test)

print('Accuracy: %.4f' % (accuracy\*100))

# Пример использования модели для предсказания класса

test\_sample = np.array([1, 1990, 1, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]).reshape(1, -1)

test\_sample = ct.transform(test\_sample) # преобразование категориальных признаков

test\_sample = scaler.transform(test\_sample) # масштабирование

predicted\_class = model.predict(test\_sample)[0][0]

# Вероятность принадлежности к классу 1

print('Predicted class probability:', predicted\_class)

# Округление вероятности и превращение в целочисленный класс

predicted\_class = int(np.round(predicted\_class))

print('Predicted class:', predicted\_class)

# График точности и функции потерь

history = model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test))

training\_loss = history.history['loss']

test\_loss = history.history['val\_loss']

epoch\_count = range(1, len(training\_loss) + 1)

plt.plot(epoch\_count, training\_loss, 'r--')

plt.plot(epoch\_count, test\_loss, 'b-')

plt.legend(['Training Loss', 'Test Loss'])

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.show()

training\_acc = history.history['accuracy']

test\_acc = history.history['val\_accuracy']

plt.plot(epoch\_count, training\_acc, 'r--')

plt.plot(epoch\_count, test\_acc, 'b-', alpha=0.7)

plt.legend(['Training Accuracy', 'Test Accuracy'])

plt.xlabel('Epoch')

plt.ylabel('Accuracy Score')

plt.show()

Описывается процесс обучения нейронной сети для прогнозирования семейного статуса швейцарцев на протяжении 16 лет (датасет "biofam" из пакета TraMineR языка R). Сначала выполняется предобработка данных: текстовые значения конвертируются в числа, из датасета удаляются ненужные столбцы, категориальные признаки кодируются с помощью OneHotEncoder, а данные масштабируются с помощью StandardScaler. Затем данные разбиваются на тренировочный и тестовый наборы. Создается модель нейронной сети с двумя скрытыми слоями. Компилируется и обучается созданная модель. Для оценки качества модели на тестовых данных вызывается метод evaluate. Затем производится прогноз на основе примера, соответствующего типу данных в датасете. Выводится вероятность принадлежности примера к классу 1 и округленный целочисленный класс. Наконец, строятся графики точности и функции потерь на тренировочном и тестовом наборах данных.

# 5. интерпретация и использование модели машинного обучения (концепция продукта данных или принятие решения на основе обобщения данных)