МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ

ТЕХНОЛОГИЙ И ТЕЛЕКОММУНИКАЦИЙ

КАФЕДРА ПРИКЛАДНОЙ ИНФОРМАТИКИ

**Лабораторная работа №6**

по дисциплине

«Искусственный интеллект и машинное обучение»

**Выполнил:**

Сердюков Никита Анатольевич

Студент 2 курса группы \_ПИН-б-о-22-1

Направления подготовки

09.03.03 Прикладная информатика

очной формы обучения

Ставрополь, 2023 г.

Тема: Линейная регрессия.

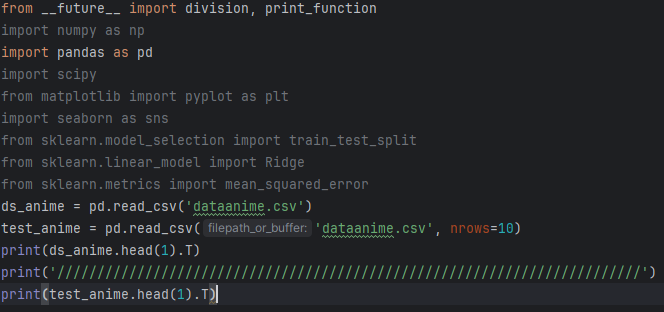
Цель работы: изучение принципов построения информационных систем с использованием линейных методов машинного обучения.

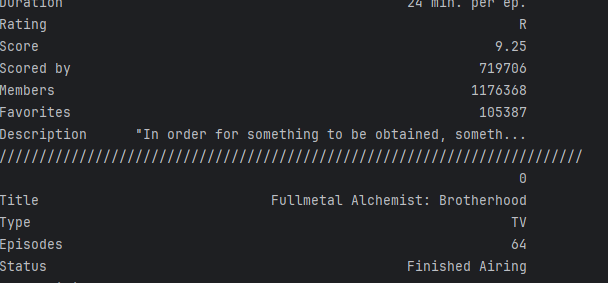
Выполнение работы:

Датасет: Anime Data

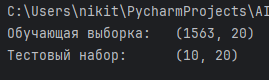
Описание датасета: Японская анимация, известная как аниме, в наши дни получила международное распространение. В этом наборе данных представлены данные об аниме, взятые из MyAnimeList.

1. Загрузка данных

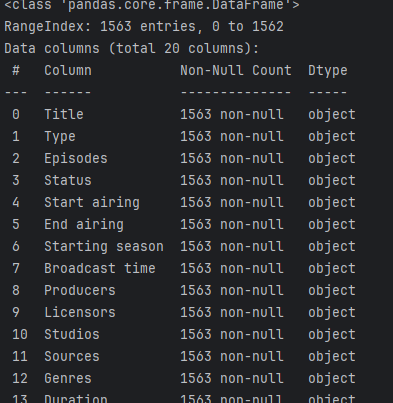






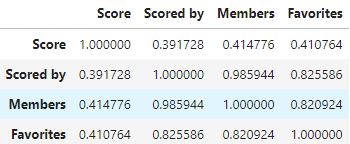


1. Тестовый набор состоит из первых десяти элементов.

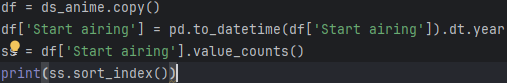


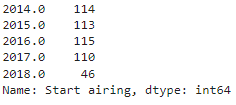
1. Корреляция признаков



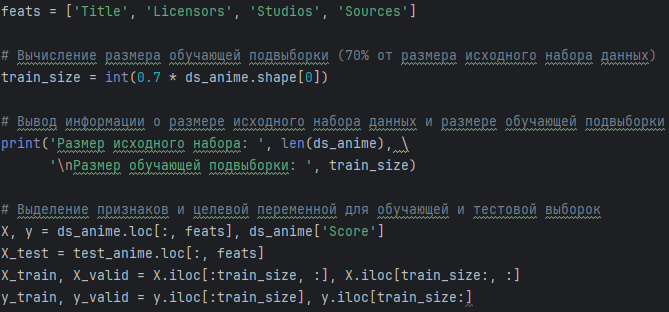


1. Количество аниме выпущенное за различные года.



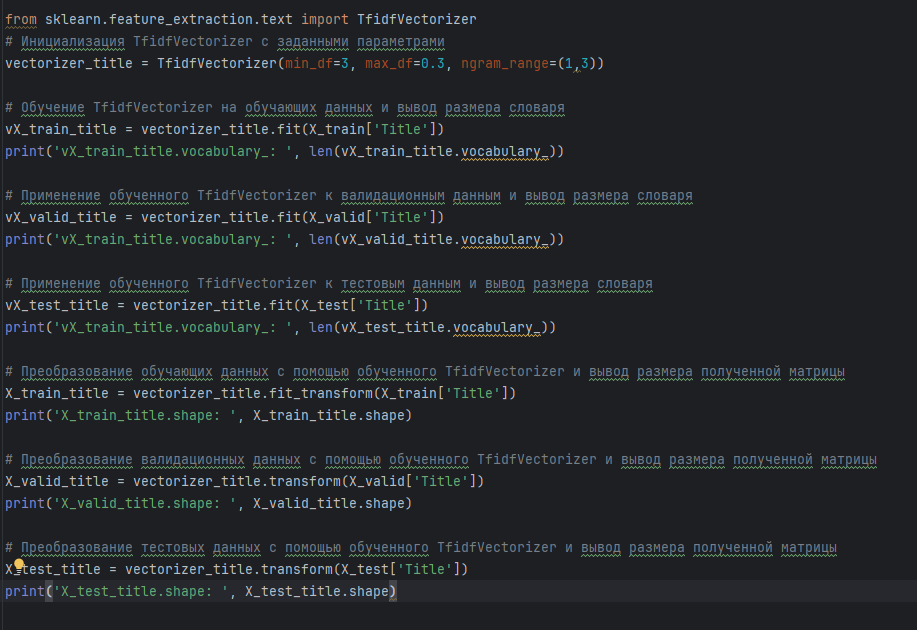


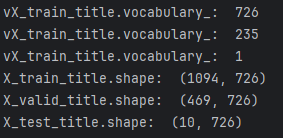
1. Работа с TfidfVectorizer



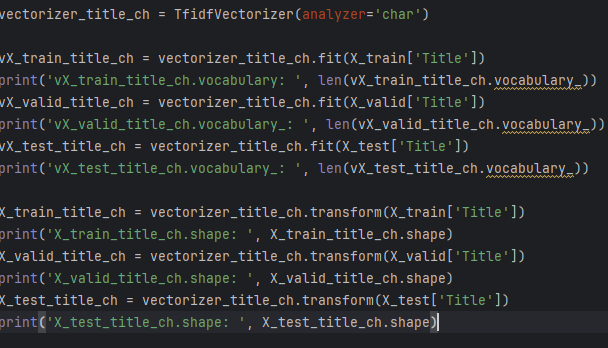


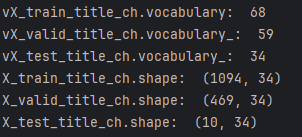
1. Создадим три словаря и три матрицы TF-IDF



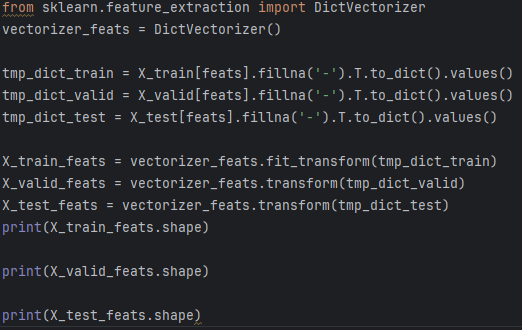


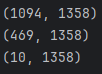
1. создание матриц и словарей, но с символами в качестве параметров



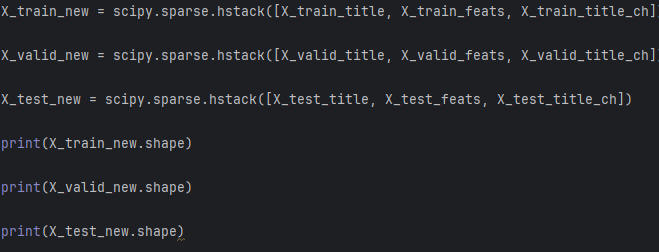


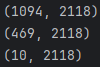
1. Создание матриц с помощью DictVectorizer, для пропущенных значений



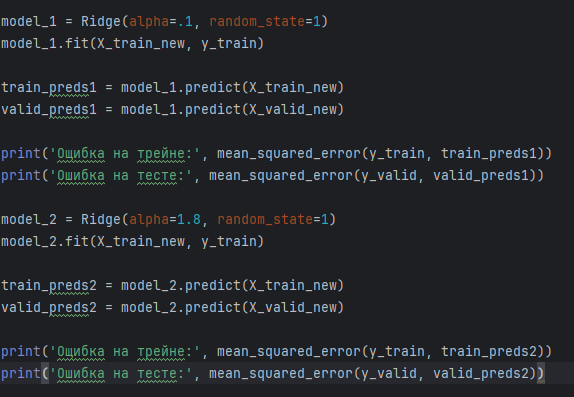


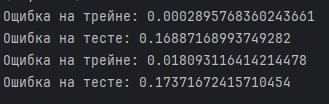
1. Объединение полученных матриц





1. Сравниваем модели





Листинг:

from \_\_future\_\_ import division, print\_function  
import numpy as np  
import pandas as pd  
import scipy  
from matplotlib import pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import Ridge  
from sklearn.metrics import mean\_squared\_error  
ds\_anime = pd.read\_csv('dataanime.csv')  
test\_anime = pd.read\_csv('dataanime.csv', nrows=10)  
  
feats = ['Title', 'Licensors', 'Studios', 'Sources']  
  
# Вычисление размера обучающей подвыборки (70% от размера исходного набора данных)  
train\_size = int(0.7 \* ds\_anime.shape[0])  
  
# Вывод информации о размере исходного набора данных и размере обучающей подвыборки  
print('Размер исходного набора: ', len(ds\_anime), \  
 '\nРазмер обучающей подвыборки: ', train\_size)  
  
# Выделение признаков и целевой переменной для обучающей и тестовой выборок  
X, y = ds\_anime.loc[:, feats], ds\_anime['Score']  
X\_test = test\_anime.loc[:, feats]  
X\_train, X\_valid = X.iloc[:train\_size, :], X.iloc[train\_size:, :]  
y\_train, y\_valid = y.iloc[:train\_size], y.iloc[train\_size:]  
  
from sklearn.feature\_extraction.text import TfidfVectorizer  
# Инициализация TfidfVectorizer с заданными параметрами  
vectorizer\_title = TfidfVectorizer(min\_df=3, max\_df=0.3, ngram\_range=(1,3))  
# Обучение TfidfVectorizer на обучающих данных и вывод размера словаря  
vX\_train\_title = vectorizer\_title.fit(X\_train['Title'])  
print('vX\_train\_title.vocabulary\_: ', len(vX\_train\_title.vocabulary\_))  
  
# Применение обученного TfidfVectorizer к валидационным данным и вывод размера словаря  
vX\_valid\_title = vectorizer\_title.fit(X\_valid['Title'])  
print('vX\_train\_title.vocabulary\_: ', len(vX\_valid\_title.vocabulary\_))  
  
# Применение обученного TfidfVectorizer к тестовым данным и вывод размера словаря  
vX\_test\_title = vectorizer\_title.fit(X\_test['Title'])  
print('vX\_train\_title.vocabulary\_: ', len(vX\_test\_title.vocabulary\_))  
  
# Преобразование обучающих данных с помощью обученного TfidfVectorizer и вывод размера полученной матрицы  
X\_train\_title = vectorizer\_title.fit\_transform(X\_train['Title'])  
print('X\_train\_title.shape: ', X\_train\_title.shape)  
  
# Преобразование валидационных данных с помощью обученного TfidfVectorizer и вывод размера полученной матрицы  
X\_valid\_title = vectorizer\_title.transform(X\_valid['Title'])  
print('X\_valid\_title.shape: ', X\_valid\_title.shape)  
  
# Преобразование тестовых данных с помощью обученного TfidfVectorizer и вывод размера полученной матрицы  
X\_test\_title = vectorizer\_title.transform(X\_test['Title'])  
print('X\_test\_title.shape: ', X\_test\_title.shape)  
  
vectorizer\_title\_ch = TfidfVectorizer(analyzer='char')  
  
vX\_train\_title\_ch = vectorizer\_title\_ch.fit(X\_train['Title'])  
print('vX\_train\_title\_ch.vocabulary: ', len(vX\_train\_title\_ch.vocabulary\_))  
vX\_valid\_title\_ch = vectorizer\_title\_ch.fit(X\_valid['Title'])  
print('vX\_valid\_title\_ch.vocabulary\_: ', len(vX\_valid\_title\_ch.vocabulary\_))  
vX\_test\_title\_ch = vectorizer\_title\_ch.fit(X\_test['Title'])  
print('vX\_test\_title\_ch.vocabulary\_: ', len(vX\_test\_title\_ch.vocabulary\_))  
  
X\_train\_title\_ch = vectorizer\_title\_ch.transform(X\_train['Title'])  
print('X\_train\_title\_ch.shape: ', X\_train\_title\_ch.shape)  
X\_valid\_title\_ch = vectorizer\_title\_ch.transform(X\_valid['Title'])  
print('X\_valid\_title\_ch.shape: ', X\_valid\_title\_ch.shape)  
X\_test\_title\_ch = vectorizer\_title\_ch.transform(X\_test['Title'])  
print('X\_test\_title\_ch.shape: ', X\_test\_title\_ch.shape)  
  
from sklearn.feature\_extraction import DictVectorizer  
vectorizer\_feats = DictVectorizer()  
  
tmp\_dict\_train = X\_train[feats].fillna('-').T.to\_dict().values()  
tmp\_dict\_valid = X\_valid[feats].fillna('-').T.to\_dict().values()  
tmp\_dict\_test = X\_test[feats].fillna('-').T.to\_dict().values()  
  
X\_train\_feats = vectorizer\_feats.fit\_transform(tmp\_dict\_train)  
X\_valid\_feats = vectorizer\_feats.transform(tmp\_dict\_valid)  
X\_test\_feats = vectorizer\_feats.transform(tmp\_dict\_test)  
print(X\_train\_feats.shape)  
  
print(X\_valid\_feats.shape)  
  
print(X\_test\_feats.shape)  
X\_train\_new = scipy.sparse.hstack([X\_train\_title, X\_train\_feats, X\_train\_title\_ch])  
  
X\_valid\_new = scipy.sparse.hstack([X\_valid\_title, X\_valid\_feats, X\_valid\_title\_ch])  
  
X\_test\_new = scipy.sparse.hstack([X\_test\_title, X\_test\_feats, X\_test\_title\_ch])  
  
print(X\_train\_new.shape)  
  
print(X\_valid\_new.shape)  
  
print(X\_test\_new.shape)  
  
model\_1 = Ridge(alpha=.1, random\_state=1)  
model\_1.fit(X\_train\_new, y\_train)  
  
train\_preds1 = model\_1.predict(X\_train\_new)  
valid\_preds1 = model\_1.predict(X\_valid\_new)  
  
print('Ощибка на трейне:', mean\_squared\_error(y\_train, train\_preds1))  
print('Ошибка на тесте:', mean\_squared\_error(y\_valid, valid\_preds1))  
  
model\_2 = Ridge(alpha=1.8, random\_state=1)  
model\_2.fit(X\_train\_new, y\_train)  
  
train\_preds2 = model\_2.predict(X\_train\_new)  
valid\_preds2 = model\_2.predict(X\_valid\_new)  
  
print('Ощибка на трейне:', mean\_squared\_error(y\_train, train\_preds2))  
print('Ошибка на тесте:', mean\_squared\_error(y\_valid, valid\_preds2))

Вывод: Результаты показывают, что model\_1 показывает лучшие результаты по сравнению с model\_2, так как достигает более низких значений среднеквадратичной ошибки MSE как на обучающем, так и на валидационном наборе данных. Кроме того, model\_1 требует больше времени на обучение по сравнению с model\_2.