

**Московский авиационный институт
(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Машинное обучение»

Лабораторная работа № 1

Студент: Цапков Александр Максимович

Группа: М8О-307Б

Преподаватель: Ахмед Самир Халид

Дата: _____

Оценка: _____

Москва, 2021

Постановка задачи

Найти себе набор данных (датасет), для следующей лабораторной работы, и проанализировать его. Выявить проблемы набора данных, устранить их. Визуализировать зависимости, показать распределения некоторых признаков. Реализовать алгоритмы К ближайших соседа с использованием весов и Наивный Байесовский классификатор и сравнить с реализацией библиотеки sklearn.

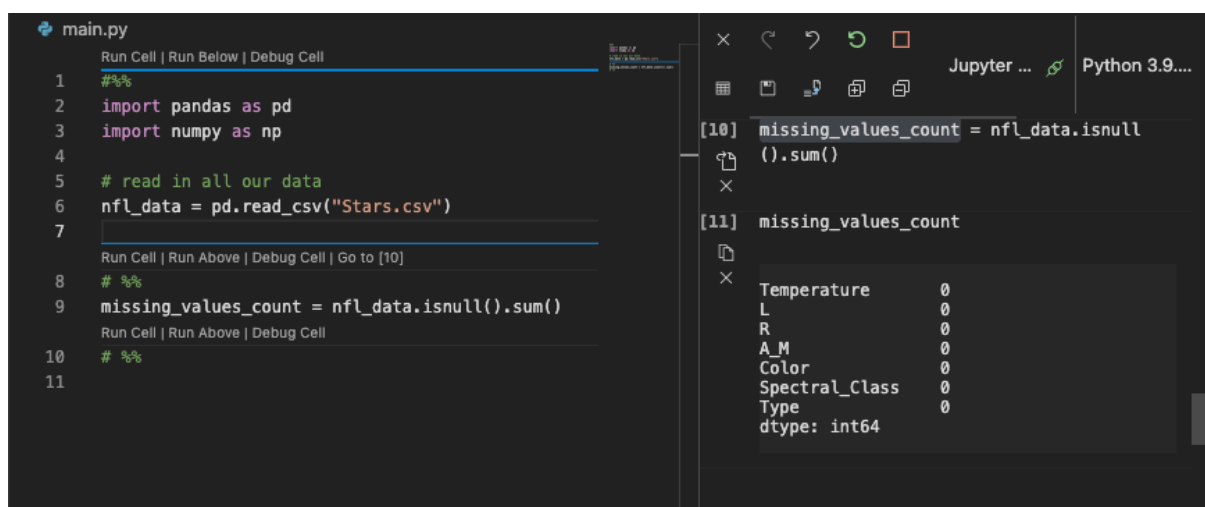
Датасет

На сайте Kaggle я выбрал интересный мне и подходящий для ЛР датасет: классификация типов звезд по их параметрам.

Подготовка датасета

До начала работы с самим датасетом мы должны его для начала подготовить. В процесс подготовки входят следующие пункты: обработка пропущенных значений, масштабирование (scaling) и парсинг.

Начнем с обработки пропущенных значений. Для этого загрузим наш датасет и посчитаем сколько в нем отсутствует значений:



```
main.py
1  #%%
2  import pandas as pd
3  import numpy as np
4
5  # read in all our data
6  nfl_data = pd.read_csv("Stars.csv")
7
8  # %%
9  missing_values_count = nfl_data.isnull().sum()
10 # %%
11
```

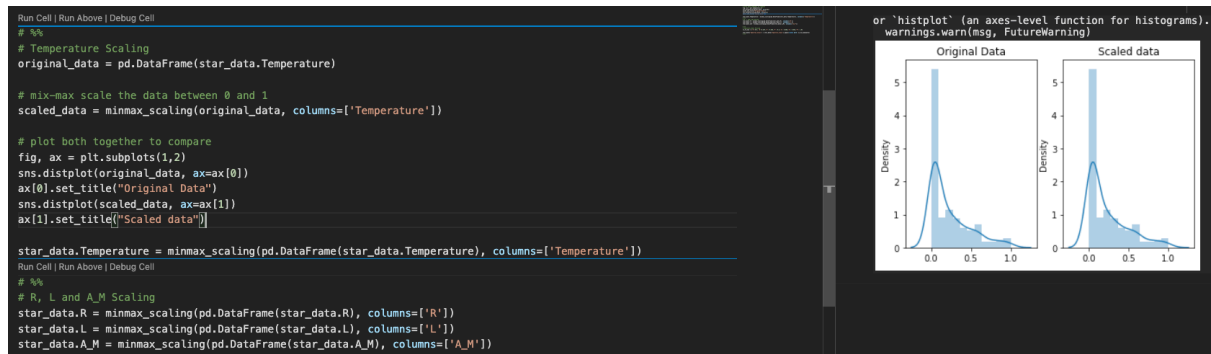
Run Cell | Run Above | Debug Cell | Go to [10]

```
[10] missing_values_count = nfl_data.isnull().sum()
[11] missing_values_count
```

Temperature	0
L	0
R	0
A_M	0
Color	0
Spectral_Class	0
Type	0
dtype:	int64

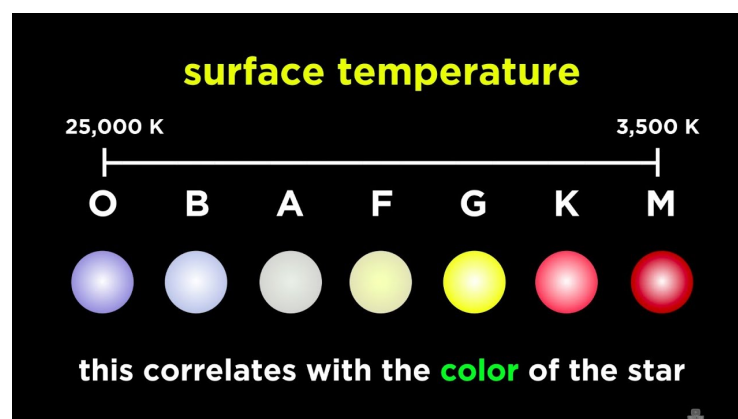
Как можно заметить, в выбранном мною датасете нет пропущенных значений, что говорит о его качестве.

Теперь приступим к масштабированию. Это нужно для однообразности данных и приведению все к одинаковому расстоянию для KNN алгоритма. Масштабирование не меняет функцию распределения данных, но при этом отображает их в нужный нам диапазон. Мы отобразим все числовые данные в диапазон от 0 до 1 с помощью функции `minmax_scaling`.



Переходим к последнему пункту подготовки данных — парсинг. В выбранном мною датасете имеются такие параметры как цвет звезды и ее спектральный класс. Это string значения и мы не можем использовать их в таком виде в KNN алгоритме. Цвет звезды напрямую связан со спектральным классом звезды, поэтому мы запарсим спектральный класс, а цвет звезды мы не будем использовать как значимый параметр.

Спектральный класс звезды связан с теми частотами которые излучает звезда. Частота излучения это numerical переменная, поэтому мы отобразим все спектральные классы в число от 0 до 1 в зависимости от тех частот которые соответствуют букве спектрального класса.



```
Run Cell | Run Above | Debug Cell | Go to [66]
# %%
# Spectral_Class Parsing
sc_to_num = {'O':0.0, 'B':0.167,'A': 0.333,'F': 0.5,'G': 0.666,'K': 0.833,'M': 1.0}

star_data['Spectral_Class'] = star_data['Spectral_Class'].apply(lambda mark: sc_to_num[mark])
Run Cell | Run Above | Debug Cell
# %%
```

0	1.000
1	1.000
2	1.000
3	1.000
4	1.000
...	...
235	0.000
236	0.000
237	0.333
238	0.333
239	0.000

Ну и конечно же мы должны разделить наши данные на тренировочные и тестовые. Для этого воспользуемся встроенной в pandas функцией. А так же удалим столбец с цветами.

```
# %%
# Splitting to Test and Lern sets
lables = star_data.pop('Type').values
star_data.pop('Color').values
from sklearn.model_selection import train_test_split
Star_train, Star_test, SLables_train, SLables_test = train_test_split(star_data, lables, test_size=0.33, random_state=0)
Run Cell | Run Above | Debug Cell
```

Алгоритмы К ближайших соседей

Для начала воспользуемся библиотекой sklearn для построения классификатора, и поле этого напишем свою реализацию.

Первым делом создадим классификатор с параметрами `n_neighbors = 5` и `weights = 'distance'`, что означает, что мы будем смотреть на 5 ближайших соседей с весами обратно пропорциональными расстоянию до них. После этого просто передаем нашему классификатору наши тренировочные данные и лэйблы к ним, а затем с помощью встроенного метода `score` проверяем точность. У меня она равна 98.75%.

```
Run Cell | Run Above | Debug Cell | Go to [141]
# %%
# sklearn KNN Classifier
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=5, weights = 'distance')
clf.fit(Star_train, SLables_train)
Run Cell | Run Above | Debug Cell | Go to [142]
# %%
# Scoring
clf.score(Star_test, SLables_test)
Run Cell | Run Above | Debug Cell
# %%
```

```
[141]: # sklearn KNN Classifier...
KNeighborsClassifier(weights='distance')

[142]: # Scoring...
0.9875
```

Теперь реализуем этот алгоритм сами. Для этого создадим класс `KNNClassifier` который обладает методами `fit`, `predict` и `score` как и классификатор в sklearn. Мой класс хранит `numpy array` со всеми координатами точек и список лэйблов. При использовании метода `predict_one` создается список дистанций до искомой точки и сортируется массив лэйблов по ключам-дистанциям. Затем для первых `n_neighbors`

точек с минимальной дистанцией до них высчитывается вес как сумма $1/\text{distance}$. Точность моего классификатора оказалась такой же как в sklearn — 98.75%

```
def predict_one(self, features):
    dist_vec = [0 for _ in range(len(self.data))]
    for k in range(len(self.data)):
        for i in range(len(self.data[k])):
            dist_vec[k] += pow(abs(self.data[k][i] - features[i]), 2)
        dist_vec[k] = math.sqrt(dist_vec[k])
    dist_vec, labl = zip(*sorted(zip(dist_vec, self.lables)))

    ans = {}
    for i in range(self.n_neighbors):
        if labl[i] not in ans:
            ans[labl[i]] = 1 / dist_vec[i]
        else:
            ans[labl[i]] += 1 / dist_vec[i]
    max_v = 0
    r_l = -1
    for k, v in ans.items():
        if v > max_v:
            max_v = v
            r_l = k
    return r_l
```

```
# %%
# sklearn KNN Classifier
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=5, weights = 'distance')
clf.fit(Star_train, SLabels_train)
Run Cell | Run Above | Debug Cell | Go to [24]

# %%
# Scoring
clf.score(Star_test, SLabels_test)
Run Cell | Run Above | Debug Cell | Go to [25]

# %%
# KNN Self Implimentation
import KnnClassifier
from KnnClassifier import KNNClassifier

sclf = KNNClassifier(5)
sclf.fit(Star_train, SLabels_train)
sclf.score(Star_test, SLabels_test)
```

```
[23] # sklearn KNN Classifier...
KNeighborsClassifier(weights='distance')

[24] # Scoring...
0.9875

[25] # KNN Self Implimentation...
0.9875
```

Наивный Байесовский классификатор

Для этого алгоритма тоже сначала воспользуемся реализацией из библиотеки `sklearn`. Так же как и с KNN создадим экземпляр класса классификатора, используем метод `fit` для обучения и `score` для проверки точности.

```
# %%  
# sklearn GaussianNB  
from sklearn.naive_bayes import GaussianNB  
gnb = GaussianNB()  
gnb.fit(Star_train, SLabels_train)  
gnb.score(Star_test, SLabels_test)
```

```
[14]: # sklearn GaussianNB...  
0.9875
```

Хоть это и наивный Байесовский классификатор, что означает предположение о независимости параметров (для найденного мною датасета это не так), однако точность очень высока — почти 99%

Теперь реализуем этот алгоритм сами. Для каждого параметра нам нужно уметь находить $P(C)$ и $P(x|C)$, где C — это тип, а x - параметр. Для этого при обучении мы вычислим матожидание и дисперсию для каждого параметра, будем считать распределение нормальным и находить по нему вероятности данной формулой:

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Код класса достаточно объемен для данного отчета, поэтому не буду приводить его здесь, расскажу только про методы. Класс `GNBClassifier` имеет такой же набор методов, как и `KNNClassifier`: `fit`, `predict_one`, `predict` и `score`. Точность моего классификатора на выбранном датасете составляет 97.5%

```
# %%  
# GaussianNB Self Implimentation  
from GNBClassifier import GNBClassifier  
  
sgnb = GNBClassifier()  
sgnb.fit(Star_train, SLabels_train)  
sgnb.score(Star_test, SLabels_test)
```

```
[14] # sklearn GaussianNB...  
0.9875
```

Приложение

1. main.py — файл с подготовкой датасета и применением классификаторов
2. GNBClassifier.py — реализация Наивного Байесовского классификатора
3. KnnClassifier.py — реализация KNN классификатора
4. Stars.csv — база данных звезд