Предобработка данных

Здравствуйте, уважаемые слушатели! Тема нашей лекции – Предобработка данных. План лекции:

1. Линейная алгебра для предобработки данных
2. Оценка набора данных
3. Входные переменные и целевая переменная
4. Обработка отсутствующих и аномальных значений
5. Преобразование дубликатов и кодирование значений
6. Корреляция значений
7. Нормализация данных

1. Вступительное слово

Часто, когда речь идет о задачах машинного обучения, предполагаются, что рассматриваются идеально «чистые» данные, которые не имеют шумов, пропущенных значений и т.п. Однако учитывая тот факт, что в реальном мире практически нет идеальных данных, которые не требуют дополнительных манипуляции, поэтому возникает необходимость предварительной обработки данных. Данный процесс позволяет подать на вход алгоритмов ML более понятные данные для анализа.

Предварительная обработка данных включает в себя: очистку шумов; обработку отсутствующих значений; поиск и удаление дубликатов; кодирование данных; нормирование или нормализацию данных; «сглаживание»; разделение данных на входные и целевые переменные; разделение данных на тренировочный и тестовый наборы.

Следует отметить, что существует множество процессов предобработки данных и выбор напрямую зависят от предметной области и качества имеющихся данных. Зачастую перечисленному выше набору добавляются дополнительные этапы, позволяющие в конечном счете повысить качество интерпретации данных.

2. Линейная алгебра

Прежде чем приступить к работе, давайте вспомним некоторые понятия из линейной алгебры, так как линейная алгебра является основополагающей в ML. Линейная алгебра будет использована не только для предобработки данных, а также в оценке моделей и преобразований данных.

Как упоминалось выше, мы будем работать с данными. А данные представляют собой массивы. Эти массивы могут быть разной размерности, будут как одномерными в виде вектора (например: целевые переменные), так и двумерными в виде матрицы (например: входные переменные).

Матрица – это прямоугольный массив чисел, обычно записываемый между квадратными скобками. Например, матрица A1 и матрица B1

$$A1 = \begin{bmatrix} 131.1 \\ 242.5 \\ 354.9 \\ 567.7 \end{bmatrix}$$

$$B1 = \begin{bmatrix} 123 \\ 245 \end{bmatrix}$$

Важным атрибутом матрицы является ее размерность, означающая количество строк и столбцов. Например, у матрицы A1 размерность 4x3, B1 имеет размерность 2x3. Размерность или размер матрицы указывается в виде строк и столбцов. То есть матрица, имеющая m строк и n столбцов, называется матрицей размера m x n. Матрица называется квадратной матрицей

(square) если m=n, то есть число столбцов равно числу строк. Иногда говорят, что матрица «толстая», если число столбцов больше числа строк или «тонкая», когда число столбцов меньше сила строк. Матрицы экивалентны если они имеют одинаковую размерность и одинаковые элементы. Обращение к элементам матриц осуществляется с помощью указания строки и столбца. Элементы матрицы могут также называться коэффициентами матрицы. Например, коэффициент A1(1,1) равен 1. Элемент A1(2,3) равен 2.5. Положительные числа в скобках – индексы, указывающие расположение элемента.

Векторы и скаляры

Матрица, которая состоит только из одной колонки, называется вектором колонкой или просто вектором. Аналогично, матрица, состоящая из одной строки, называется вектором строкой. Размерность вектора колонки из n элементов n x 1, вектора строки из m элементов 1 x m. Вектор колонка

$$vc = \begin{bmatrix} 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

Имеет размерность 4 x 1. Вектор строка vr имеет размерность 1 x 3

$$vr = \begin{bmatrix} 1179.99 \end{bmatrix}$$

Матрица размером 1x 1 являющаяся просто значением часто называется скаляром.

Вектора, матрицы часто обозначаются строчными и заглавными буквами, соответственно. Например, A — это матрица, в то время как v – вектор. Хотя существуют и другие общепринятые стандарты обозначений векторов и матриц, мы будем придерживатся упомянутой нотации.

Нулевая и единичная матрицы

Нулевая матрица содержит только нули в качестве элементов. Единичная матрица – это квадратная матрица, которая содержит единицы на главной диагонали. То есть, все элементы матрицы в которых индексы совпадают равны 1, остальные элементы равны 0

$$I(i, j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Примеры единичных матриц

$$I_1 = \begin{bmatrix} 100 \\ 010 \\ 001 \end{bmatrix}$$

$$I_2 = \begin{bmatrix} 10 \\ 01 \end{bmatrix}$$

Операции с матрицами

Рассмотрим часто используемые операции с матрицами: сложение, умножение, транспонирование

Сложение и вычитание возможно только для матриц одинакового размера. Например

$$\begin{matrix}12\\34\\56\end{matrix} + \begin{matrix}21\\11\\45\end{matrix} = \begin{matrix}33\\45\\911\end{matrix}$$

Операция сложения матриц коммутативна (A+B=B+A) и ассоциативна (A+B)+C=A+(B+C)=A+B+C.

Операция умножения на скаляр выполняется поэлементно, например

$$-2 * \begin{matrix}21\\11\\45\end{matrix} = \begin{matrix}-4-2\\-2-2\\-8-10\end{matrix}$$

Умножение матриц и векторов

Умножение возможно только для матриц с согласованной размерностью. Например, если размерность матрицы K обозначить как R_K^{4x3} , матрицы G как R_G^{2x3}, то умножение этих матриц невозможно. Если же мы имеем две матрицы размерностью R_C^{4x3} и R_D^{3x4}, то результат умножения

M1=C*D будет иметь размерность R_{M1}^{4x4}

Другими словами, при умножении матрицы A на матрицу B, число колонок матрицы A должно быть равно числу строк матрицы B. В случае, упомянутых матриц C и D, можно выполнить умножение и в другом порядке M2 = DC, но при этом размерность M2 будет равна 3x3 (R_{M2}^{3x3}). Отсюда естественным образом следует, что, $M1 \neq M2$ Иными словами, операция умножения для матриц не коммутативна, то есть

$$AB \neq BA$$

Однако свойство ассоциативности сохраняется $(AB)C = A(BC)$.

Собственно умножение выполняется следующим образом. Пусть мы хотим получить произведение C=AB. Каждый элемент матрицы C вычисляется следующим образом

$$C(i, j) = \sum_{k=1}^{P} a_{ik}\, b_{kj}$$

Рассмотрим пример.

Пусть

$$A = \begin{matrix}12\\34\\56\end{matrix}$$

$$B = \begin{matrix}123\\456\end{matrix}$$

Тогда

$$M1 = AB = \begin{matrix}91215\\192633\\294051\end{matrix}$$

и

$$M2 = BA = \begin{matrix}2228\\4964\end{matrix}$$

Элементы матрицы M2 рассчитываются следующим образом:

M2(1,1)= B(1,1)*A(1,1)+B(1,2)*A(2,1)+B(1,3)*A(3,1)= 1*1+2*3+3*5=22.
M2(1,2)= B(1,1)*A(1,2)+B(1,2)*A(2,2)+B(1,3)*A(3,2)= 1*2+2*4+3*6=28.
M2(2,1)= B(2,1)*A(1,1)+B(2,2)*A(2,1)+B(2,3)*A(3,1)= 4*1+5*3+6*5=49.
M2(2,1)= B(2,1)*A(1,2)+B(2,2)*A(2,2)+B(2,3)*A(3,2)= 4*2+5*4+6*6=64.

Возможно умножение матрицы на вектор. Размерность также должна быть согласована, например, если мы имеем матрицу C размерностью R_C^{4x3} и вектор vr размерностью R_{vr}^{3x1}, то результат умножения C*vr будет также вектором размерностью R_{Cvr}^{4x1}. В случае, умножения векторов.

Интересный случай умножения вектора строки на вектор столбец. Результатом такого умножения является скаляр размерностью 1 x 1.

Возведение матриц в целую степень возможно для квадратных матриц, однако извлечение корня или возведение в степень меньшую единицы возможно не всегда также как и возведение в нулевую степень, которое возможно только для инвертируемых матриц.

Матрица A инвертируема или не сингулярна, если найдется такая матрица K такая, что KA=I. Матрица K называется обратной матрицей и обозначается A^{-1}. Если матрица не имеет обратной она называется сингулярной или необратимой. Необходимым, но не достаточным условием инвертируемости является условие квадратности матриц. Другими словами, не все квадратные матрицы имеют обратную матрицу. Например, не имеет обратной нулевая матрица. Матрица вида

$$\begin{bmatrix} 1 & -1 \\ -2 & 2 \end{bmatrix}$$

также сингулярна и не имеет обратной.

Транспонирование матриц

Если A – матрица размера m x n, то транспонированная матрица A` или $A^T(i, j) = A(j, i)$ для всех i и j. Например,

$$B = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 5 \end{bmatrix}$$

$$B^T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \end{bmatrix}$$

Массив A размерности mxn выражается следующим образом:

Если представить, что каждая строка массива — это объект, а столбцы — это признаки, то можно с уверенностью можно сказать смотря на данную матрицу, что мы имеем m количества объектов, каждая из которых выражена из n количества признаков. Признаки можно поделить на несколько классов или типов, каждый из которых имеет свои особенности и обрабатываются по-разному.

- Бинарные признаки - тип признаков, которые принимают значения из Dj = {0, 1};
- Вещественные признаки - тип признаков, которые принимают вещественные числа Dj = R;
- Категориальные признаки - тип признаков, особенностью которых является невозможность сравнения «больше-меньше» значений, Dj — неупорядоченное множество;

- Порядковые признаки - частный случай категориальных признаков, но в данном случае Dj — упорядоченное множество;
- Множествозначные признаки - тип признаков, в котором есть подмножество некоторого множества признаков.

В некоторых литературах их делят на два: качественные и количественные.

Но не стоить забывать о целевых переменных, которая играет важную роль в контролируемых алгоритмах ML. Но об этом будет рассказано более развернуто в следующих занятиях.

3. Оценка набора данных

В предыдущем уроке были даны краткие описания предназначения некоторых библиотек, среди которых был pandas, который сейчас будет использован для загрузки набора данных и первичного ознакомления с набором данных. Для этого воспользуемся следующим кодом. Мы использовали датасет survey.csv, который можно скачать по ссылке написанный внизу https://www.dropbox.com/sh/f9wx01fuliqg39c/AAC61D8lPJ2-58ZR88hcu_y3a?dl=0.

```
import pandas as pd
df = pd.read_csv("survey.csv")
df.head()
```

В результате мы получим таблицу, которая показывает, что данные весьма далеки от идеальных. В колонке Gender ссодержатся не стандартизованные значения, колонки state, self_employed, comments содержат много неопределенных (пропущенных) значений, колонка Age, как будет видно в дальнейшем, содержит отрицательные и слишком большие значения.

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employees	...
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes	Often	6-25	...
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No	Rarely	More than 1000	...
2	2014-08-27 11:29:44	32	Male	Canada	NaN	NaN	No	No	Rarely	6-25	...
3	2014-08-27 11:29:46	31	Male	United Kingdom	NaN	NaN	Yes	Yes	Often	26-100	...
4	2014-08-27 11:30:22	31	Male	United States	TX	NaN	No	No	Never	100-500	...

Хоть нам уже ясно что эти данные не идеальные, нужно убедить в этом посмотрев на статистику. Для этого применяем следующее:

df.shape - информация о размерности дата фрейма;
Out: (1259, 27)

df.info() - информация о размерности данных и как данные индексируются, количество not-a-number элементов;
Out: <class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 27 columns):
 # Column Non-Null Count Dtype
--- ------ -------------- -----

```
 0   Timestamp                  1259 non-null   object
 1   Age                        1259 non-null   int64
 2   Gender                     1259 non-null   object
…
```

df.describe() - статистики: count,mean, std, min, 25%-50%-75% percentile, max;
Out:

```
                Age
count  1.259000e+03
mean   7.942815e+07
std    2.818299e+09
min   -1.726000e+03
25%    2.700000e+01
50%    3.100000e+01
75%    3.600000e+01
max    1.000000e+11
```

Как вы заметили функция describe() работает с численными типами данных. Соответственно вывел статистику только для признака «Age» тип которого является int.

df.nunique() - уникальных значений для каждого столбца дата фрейма;
Out:

```
Timestamp                  1246
Age                          53
Gender                       49
Country                      48
state                        45
self_employed                 2
family_history                2
treatment                     2
work_interfere                4
no_employees                  6
```

Как мы видим, столбец Gender содержит 49 вариантов, хотя на самом деле ожидалось всего 2, а среднее значение возраста в колонке Age – около 79 миллионов лет, что совсем неуместно. Для того чтобы разобраться с подобными странными результатами, желательно иметь перечень уникальных значений каждой колонки данных. Для этого выполним следующее:

```
feature_names = df.columns.tolist()
for column in feature_names:
    print(column)
    print(df[column].value_counts(dropna=False))
```

Мы увидим, что столбец Age содержит ошибочные значения, например, отрицательный возраст или возраст, сравнимый с возрастом Вселенной, а также возраст детей, которые еще не работают и не могут высказать мнение о своем рабочем месте:

```
Age
 99999999999     1
```

```
 5            1
-1            1
 11           1
 8            1
-29           1
-1726         1
 329          1
```

Колонка Gender содержит много разных способов записи пола:

```
Gender
Male                615
male                206
Female              121
M                   116
female              62
F                   38
m                   34
f                   15
Make                4
Male                3
Woman               3
```

4. Входные и целевые переменные

Целевая переменная, в нашем случае это «treatment», которая содержит два значения: Yes или No. Исходя из примерно равного распределения Yes и No, можем считать корпус сбалансированным.

```
treatment
Yes    637
No     622
```

Колонку treatment исключаем из набора входных переменных:

```
features = df.drop('treatment', 1)
```

и формируем отдельную колонку labels

```
labels = df['treatment']
```

Итак, мы разбили набор данных на входные значения features и целевую колонку labels. Целевая переменная помечает каждую строку входных данных значением Yes или No. Забегая перед, нужно сказать что данная обработка нужна для задачи классификации. А для нормальной работы классификатора необходимо привести исходные данные в порядок, исключив или изменив аномальные значения, устранив дубликаты и перекодировав некоторые колонки, содержащие категориальные значения с помощью one-hot-encoding.

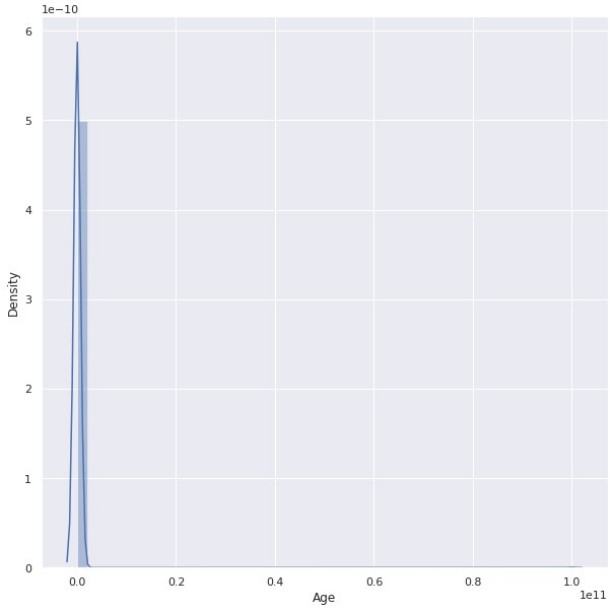
5. Обработка отсутствующих и аномальных значений.

Один из простых способов обработки отсутствующих и аномальных значений – просто удалить строки с такими значениями. Однако в этом случае мы теряем данные: до удаления отсутствующих значений размер дата фрейма (1259, 26), а после (86, 26).

Как мы видим, такие потери слишком велики и могут повлиять на качество классификации. Хотя не существуют общепринятых рекомендаций, тем не менее в некоторых случаях пропущенные данные могут быть заполнены, например, средними или медианными значениями. Посмотрим на колонку Age, визуализировав распределение данных возраста, например, так:

```
%matplotlib inline
import seaborn as sns
sns.set(color_codes=True)
plot = sns.distplot(features.Age.dropna())
plot.figure.set_size_inches(10,10)
```

и получим картинку, далекую от реального распределения возрастов.



Значения, выходящие за разумные рамки, можно преобразовать в формат not-a-number-np.nan:

```
features.loc[(features['Age'] < 18) | (features['Age'] >72), 'Age'] = np.nan
```

а затем, используя SimpleImputer [1], заменить эти значения средними значениями по колонке:

```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
```

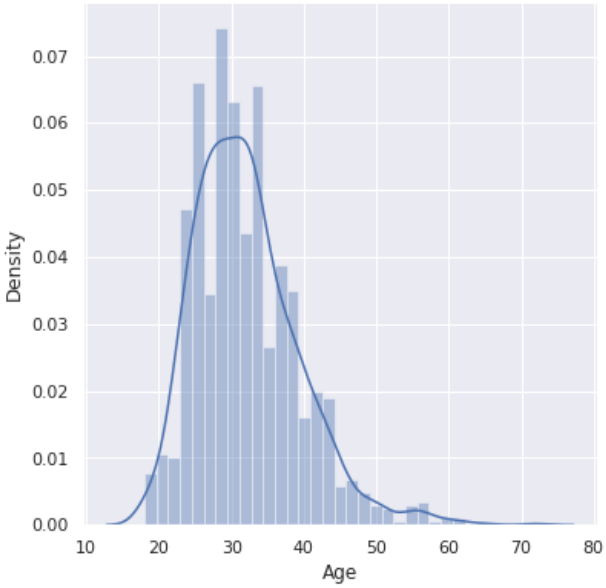
Фрагмент кода, выполняющий указанные операции, показан ниже:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
features.loc[(features['Age'] < 18) | (features['Age'] >72), 'Age'] = np.nan
Age1=imputer.fit_transform(features[['Age']])
```

1

```
features['Age']=Age1
```

Теперь дата фрейм features содержит новую колонку Age1, значения в которой не меньше 18 и не больше 72. Визуализировав значения в колонке Age1 тем же фрагментом кода, получим разумное распределение, показанное на графике.



Но на практических задачах не всегда целесообразно использовать такие манипуляции (средними, модой или же медианными) значениями для заполнения пропусков. Это обсусливается тем, что данные могут быть сильно искажены, так как данный метод не учитывает тип переменных в столбце. Для предотвращения таких ситуации существует множество и других методов заполнения. Среди них можно отметить: KNNImputer, IterativeImputer и т. п.

6. Преобразование дубликатов и кодирование значений

Алгоритмы машинного обучения, как правило, «ожидают» на входе числовые значения. Один из вариантов – кодирование категориальных значений целыми числами. Для этого можно использовать готовую функцию LabelEncoder() из билиотеки sklearn.preprocessing.

Например, для колонки 'leave':

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(features['leave'])
label_encoder.transform(features['leave'])
```

Для упрощения многократной обработки создадим функцию:

```
def convertToNumbers (df,feature):
    label_encoder = preprocessing.LabelEncoder()
    label_encoder.fit(df[feature])
    df[feature]=label_encoder.transform(df[feature])
    return df
```

Данная функция будет конвертировать нечисловые значения в целое число. После ее применения в цикле:

```
feature_names=['Country','Gender','state','self_employed','family_history','work_interfere','no_employees','remote_work']
for f in feature_names:
    features=convertToNumbers (features,f)
features
```

получим:

	Timestamp	Age	Gender	Country	state	self_employed	family_history	work_interfere	no_employees	remote_work	...	leave	mental_health_cons
0	2014-08-27 11:29:31	37.0	0	45	10	0	0	1	4	0	...	2	
1	2014-08-27 11:29:37	44.0	1	45	11	0	0	2	5	0	...	0	
2	2014-08-27 11:29:44	32.0	1	7	29	0	0	2	4	0	...	1	
3	2014-08-27 11:29:46	31.0	1	44	29	0	1	1	2	0	...	1	
4	2014-08-27 11:30:22	31.0	1	45	38	0	0	0	1	1	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
1254	2015-09-12 11:17:21	26.0	1	44	29	0	0	3	2	0	...	2	
1255	2015-09-26 01:07:35	32.0	1	45	10	0	1	1	2	1	...	1	
1256	2015-11-07 12:36:58	34.0	1	45	2	0	1	3	5	0	...	1	
1257	2015-11-30 21:25:06	46.0	0	45	22	0	0	3	1	1	...	0	
1258	2016-02-01 23:04:31	25.0	1	45	10	0	1	3	2	0	...	0	

1259 rows × 26 columns

Проблема при таком конвертировании заключается в том, что мы задаем порядок на категориальных значениях. Например, для колонки Gender 0 означает Female, 1 – Male, 2 – Other. Или, например, страна с кодом 45 будет значить больше, чем страна с кодом 7, то есть мы неявным образом ранжируем данные. Чтобы избежать возникновения такого порядка, применяется преобразование значений в унитарный код (one-hot-encoding).

Примечание. Унитарный код – это код, при котором признак (свойство) объекта кодируется двоичным вектором, содержащим только одну единицу. Длина вектора равна количеству вариантов данного свойства.

Например, для колонки Gender, значения которой равнозначны, данные можно преобразовать так:

Female	Male	Other
1	0	0
0	1	0
0	1	0
0	1	0
0	1	0

Для подобного преобразования можно использовать pd.get_dummies():

```
import pandas as pd
pd.get_dummies(features['Gender'])
```

Для того чтобы заменить исходные колонки набора данных на некоторое количество новых, можно воспользоваться функцией:

```
def convertToOHE (df,feature):
    ohe=pd.get_dummies(df[feature])
    df=df.drop(feature, 1)
    df=df.join(ohe)
    return df
features =  convertToOHE (features,'Gender')
```

Функция получает на вход data frame, в котором удаляет исходную колонку и заменяет ее несколькими колонками one-hot-encoding.

7. Корреляция значений

Теперь у нас почти все готово. Удалим колонки, которые не удастся использовать в дальнейшем:

```
features=features.drop(['Timestamp','comments'],1)
```

Чтобы лучше понять связь между признаками нужно оценить взаимную корреляцию исходных данных, воспользовавшись следующим фрагментом кода:

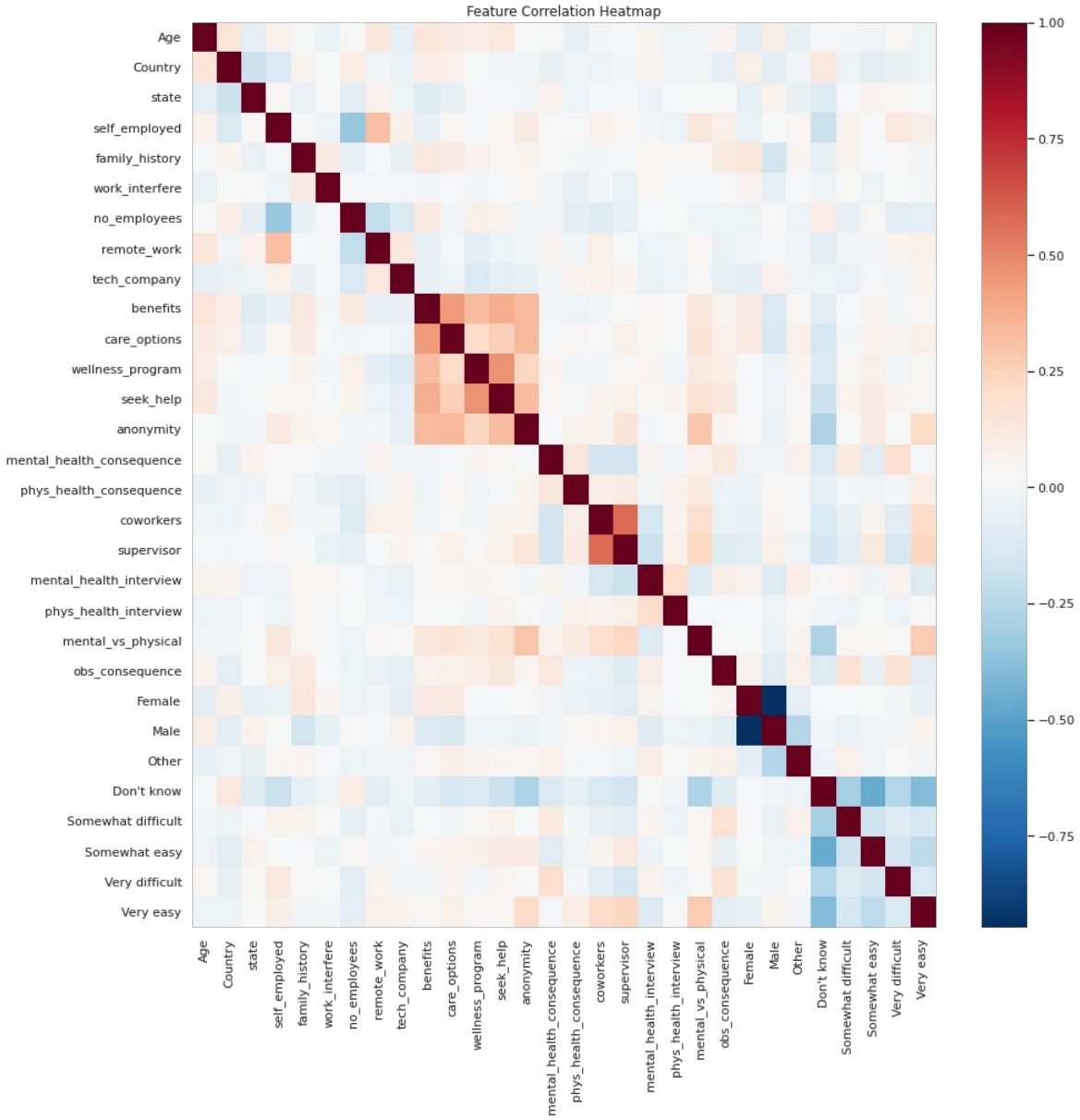
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

def show_heatmap(data):
    plt.figure(figsize=(15, 15))
    pearson_corr = features.corr(method='pearson')
    sns.heatmap(pearson_corr, annot=False, cmap='RdBu_r')

    plt.title("Feature Correlation Heatmap", fontsize=12)
    plt.show()

show_heatmap(features)
```

Нужно отметить, что для выявления взаимосвязи между признаками была использована коэффициент корреляции Пирсона, который в Python используется по умолчанию. Также можно использовать коэффициент корреляции Спирмена.



Видно, что данные довольно слабо коррелированы, за исключением очевидной и явно выраженной корреляции между Male и Female.

Целевая колонка treatment не коррелирует значительно с какой-либо колонкой. Это говорит о том, что при обучении модели целевая колонка не определяется какой-то одной колонкой исходных данных, а зависит от распределения данных. В общем-то это говорит о том, что все данные нужны для корректного решения задачи. Второй вывод, который мы можем сделать на основании данной таблицы, заключается в том, что исходные данные относительно корректны, то есть не происходит так называемой утечки данных (data leakage). В машинном обучении данный термин означает, что во входных данных уже содержится ответ на задачу. В этом случае наблюдалась бы сильная корреляция между одним из входных свойств и целевой колонкой. Если бы это было так, то созданная модель получилась бы примитивной и, по существу, не нужной, то есть задачу можно было бы решать, просто анализируя значение одной колонки входных данных.

8. Нормализация данных

Если мы не нормализуем данные, то входные переменные, которые изменяются в более широком диапазоне значений, станут для алгоритма «более значимыми». Для того

чтобы этого не произошло, входные значения приводят к одному масштабу. Таким образом, нормализация данных позволяет исключить зависимость алгоритма от абсолютных величин входных переменных. Посмотрим на пример, у нас имеется набор данных с различными единицами: температура в градусах Кельвина, относительная влажность и день года. Ожидаемые диапазоны для каждой входной переменной:

- Температура: от 270 K до 305 K
- Относительная влажность: от 0 до 1 (т.е. влажность 30%, равная 0,3)
- День года: от 0 до 365

Когда мы смотрим на эти значения, мы интуитивно соотносим их, основываясь на своем опыте. Например, увеличение влажности на 0,5% более значимо, чем изменение температуры на полградуса. Однако алгоритм машинного обучения не имеет такого опыта. Поэтому для него температура станет намного более важным параметром, чем относительная влажность. Нормализация данных позволяет всем признакам вносить вклад на основании их важности в задаче машинного обучения, а не масштаба представления входных данных.

Вместе с тем не все алгоритмы одинаково чувствительны к нормализованным значениям. В таблице приведен неполный список алгоритмов, которые нуждаются в нормализации входных значений для достижения хороших показателей качества, и алгоритмов, которые хорошо работают и без нормализации данных.

Алгоритмы, которые требуют нормализованных данных	Алгоритмы, которые не нуждаются в нормализации
Логистическая регрессия	Деревья принятия решений (и случайные леса)
Машины опорных векторов (SVM)	Градиентный бустинг (например, XGBoost)
Нейронные сети всех видов	Наивный Байес (Naïve Bayes)
Метод главных компонент (PCA)	
k-NN	

Наиболее распространенным способом нормализации является линейная нормировка. Линейная нормировка способствует тому, чтобы каждая компонента входного вектора находилась на заданном отрезке, например, от 0 до 1 или от –1 до 1. При известном диапазоне изменения входной переменной можно использовать простейший вид преобразования:

$$p = \frac{(x - x_{min})(max - min)}{(x_{max} - x_{min})} + min,$$

где [*min*, *max*] – задаваемый диапазон изменения входных переменных; [x_{min} , x_{max}] – изначальный диапазон изменения значений входной переменной; *p* – преобразованное входное значение.

Такой способ трансформации входных значений выполняет MinMaxScaler из библиотеки scikit-learn. Во многих случаях можно воспользоваться сокращенным выражением, которое приводит любое значение к диапазону от 0 до 1:

$$p = \frac{(x - x_{min})}{(x_{max} - x_{min})}.$$

При кодировании количественных переменных необходимо в общем случае учесть содержательное значение признака, его расположение на интервале значений, точность измерения. Преобразование может быть выполнено с помощью двух выражений:

$$x_i = \frac{x_i - M(x_i)}{q(x_i)}$$

или

$$x_i = \frac{x_i - M(x_i)}{max x_i - M(x_i) \vee},$$

где

x_i – *i*-я координата входного вектора *X;*

$M(x_i) = \frac{1}{n}\sum_{i=1}^{n}$ x_i – выборочная оценка математического ожидания x_i (среднее значение);

$q(x_i) = \sqrt{}$

– выборочная оценка среднеквадратического отклонения.

Применение данных масштабирующих утилит на практике:

```
from sklearn.preprocessing import StandardScaler
scale_features_std = StandardScaler()
features_StandardScaler = scale_features_std.fit_transform(features)
features_StandardScaler
```

или

```
from sklearn.preprocessing import MinMaxScaler
scale_features_mm = MinMaxScaler()
features_MinMaxScaler  = scale_features_mm.fit_transform(features)
features_MinMaxScaler
```

В результате мы получим массив числовых значений:

```
array([[0.35185185, 0.95744681, 0.22222222, ..., 1.        , 0.        ,
        0.        ],
       [0.48148148, 0.95744681, 0.24444444, ..., 0.        , 0.        ,
        0.        ],
       [0.25925926, 0.14893617, 0.64444444, ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.2962963 , 0.95744681, 0.04444444, ..., 0.        , 0.        ,
        0.        ],
       [0.51851852, 0.95744681, 0.48888889, ..., 0.        , 0.        ,
        0.        ],
       [0.12962963, 0.95744681, 0.22222222, ..., 0.        , 0.        ,
        0.        ]])
```

На этом наш урок подходит к концу.