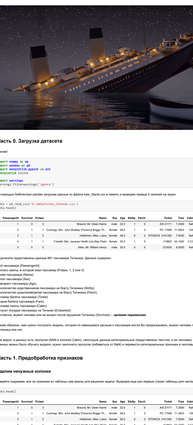
Практика в библиотеке sklearn, часть 2: Titanik

В этом задании мы применим полученные знания к датасету Titanik для его этого соревнования: https://www.kaggle.com/c/titanic

Данные основаны с сайта на оценке, кто умер живой и кто умер с тем или другим.

Почитать о том, что такое титаник: https://www.kaggle.com/c/titanic/data#start-0



Часть 0. Загрузка датасета

Начнем!

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

С помощью библиотеки pandas загрузим данные из файла train_titanik.csv в память и выведем первые 5 записей на экран:

```
data = pd.read_csv('D:/data/train_titanik.csv')
data.head()
```

	Passengerid	Survived	Pclass	Name	Sex	Age	Sibsp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

В данном задании мы будем работать с данными ПО titanicскому Титанику. Данные содержат:

- id пассажира (Passengerid)
- класс каюты, в которой жил пассажир (Pclass: 1, 2 или 3)
- имя пассажира (Name)
- пол пассажира (Sex)
- возраст пассажира (Age)
- количество родственников пассажира по линии Sibsp (Sibsp)
- количество родственников пассажира по линии Parch (Parch)
- номер билета пассажира (Ticket)
- цена билета пассажира (Fare)
- номер каюты пассажира (Cabin)
- порт посадки пассажира на Титанике (Embarked)
- спасен/не спасен или он выжил после крушения Титаника (Survived) – **целевая переменная**

Таким образом, наша задача построить модель, которая по имеющимся данным о пассажире могла бы предсказывать, выжил человек после крушения Титаника или нет.

Как видно, в данных есть пропуски (NAN в колонке Cabin), категориальные данные категориальные (представлены текстом, а не числами). Чтобы мы этим данным можем было обучать модели, нужно заполнить пропуски (избавиться от NaN) и перевести категориальные признаки в числовые.

Часть 1. Предобработка признаков

Удалим ненужные колонки

Давайте подумаем, все ли признаки из таблицы нам нужны для решения задачи. Выведем еще раз первые строки таблицы для напоминания:

```
data.head()
```

	Passengerid	Survived	Pclass	Name	Sex	Age	Sibsp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Мы, как люди, знаем, что вероятность за Титаник не зависит от имени пассажира и номера билета. Поэтому давайте удалим эти колонки. Колонку Passengerid тоже не оставляем, так как она была добавлена в данные для удобства и, очевидно, вычисляемость пассажира не зависит от его ID в этом датасете.

```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html
data = data.drop(['Passengerid', 'Name', 'Ticket'], axis=1)
```

Посмотрим на данные

Для начала удалим строчки колонку целевой переменной от датасета и разделим датасет на тренировочный и тестовый. Тренировочное datasetа будем обучать МЛМ, на тестовом – тестировать.

```
X = data['Survived']
# https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.drop.html
data = data.drop('Survived', axis=1)
```

Давайте посмотрим, какие в данных есть колонки и какого типа данные в них записаны:

```
print(data.columns)
print(data.dtypes)
```

```
Index(['Pclass', 'Sex', 'Age', 'Sibsp', 'Parch', 'Fare', 'Cabin', 'Embarked'], dtype='object')
Pclass       int64
Sex         object
Age        float64
Sibsp        int64
Parch        int64
Fare       float64
Cabin       object
Embarked    object
dtype: object
```

И самим посмотрим, в каких столбцах данные содержатся NaN:

```
data.columns[pd.isnull(data).any()].tolist()
```

```
['Age', 'Cabin', 'Embarked']
```

Очень часто бывает полезно посмотреть на связь разных признаков между собой на картинке. Это позволяет выявить выбросы (элементы, которые сильно отличаются от остальных), понять, какие признаки похожи друг на друга, а какие – нет. Для этого в библиотеке pandas есть специальная функция scatter_matrix, которая отображает все данные в виде кривой-какой-нибудь картинки (найти нужную по названию, наверно), также можно воспользоваться значением s (=1)

```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.plotting.scatter_matrix.html
pd.scatter_matrix(data, figsize=(20, 20), s=1)
```

```
<matplotlib.collections.PathCollection at 0x7fa1e4e6d1d0>
```

Заполнение пропусков в данных

Начнем с того, что заполним NaN в колонках. Где-то есть, скажем-наверх, значениями. Для того, чтобы лучше понять, какие значения для пропусков выбрать, можно посмотреть на данные с этим кривой-какой-нибудь статистики (найти нужную по названию, наверно), также можно воспользоваться значением s (=1)

1. Age

Возраст человека может быть любым числом от 0 до 147. Поэтому, наверно, что для колонки Age заполнить на колонки-тип строк с каждым отличным от всех значением. Большой смысл имеет найти значение колонки.

```
data['Age'].mean(), data['Age'].median()
```

```
(29.699117647058826, 28.0)
```

```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html
data['Age'] = data['Age'].fillna(28)
```

2. Embarked

Колонка Embarked, наоборот, содержит категориальное значение, которой имеет много разных значений в датасете. Поэтому для него как-раз имеет смысл найти значение с наибольшим количеством повторений.

```
data['Embarked'].value_counts()
```

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

Заполним отсутствующие значения самым часто встречающимся значением S:

```
data['Embarked'] = data['Embarked'].fillna('S')
```

Cabin

И вот тут данность с колонкой Cabin, сразу напишется. Она категориальная, и значений в ней много разных.

Давайте посмотрим, сколько всего неповторяющихся значений в колонке Cabin:

```
data['Cabin'].value_counts().sort_values
```

```
<bound method Series.sort_values of C23 C25 C27        4
G6             4
B96 B98        4
C22 C26        3
F33            3
              ..
E34            1
C7             1
C54            1
E36            1
C148           1
Name: Cabin, Length: 147, dtype: int64>
```

Категорий слишком много, давайте это колонку тоже удалим:

```
data = data.drop('Cabin', axis=1)
```

Отлично, мы разобрались со всеми пропусками. Они категориальные, и значений в них много разных.

```
# Вычисл ьзовать, чтоо в данных осталось пропусков
assert not data.isnull().values.any()
```

Перевод категориальных признаков в числа

Модели машинного обучения (как родная изначальная) умеют работать только с числовыми признаками. Поэтому все нечисловые признаки нужно превратить в числовые.

Еще раз выведем колонки и их типы:

```
print(data.columns)
print(data.dtypes)
```

```
Index(['Pclass', 'Sex', 'Age', 'Sibsp', 'Parch', 'Fare', 'Embarked'], dtype='object')
Pclass       int64
Sex         object
Age        float64
Sibsp        int64
Parch        int64
Fare       float64
Embarked    object
dtype: object
```

У нас всего две категориальных колонки (Sex и Embarked).

```
data['Sex'].describe()
```

```
count    891
unique     2
top     male
freq     577
Name: Sex, dtype: object
```

1. Sex

В колонке Sex всего два различных значения (male и female). Давайте закодируем эти значения цифрами 0 и 1 и заменим значения в колонке на 0 и 1:

```
data['Sex'] = data['Sex'].astype('category')
data['Sex'] = data['Sex'].cat.codes
```

```
data.head()
```

	Pclass	Sex	Age	Sibsp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S
0	3	1	22.0	1	0	7.2500	0	0	1
1	1	0	38.0	1	0	71.2833	1	0	0
2	3	0	26.0	0	0	7.9250	0	0	1
3	1	0	35.0	1	0	53.1000	0	0	1
4	3	1	35.0	0	0	8.0500	0	0	1

2. Embarked

```
data['Embarked'].describe()
```

```
count     891
unique      3
top         S
freq      644
Name: Embarked, dtype: object
```

В колонке Embarked всего 3 уникальных значения. Мы можем поступить с ними так же, как и с колонкой Sex – заменить значения на цифры 0, 1 и 2. Давайте сделаем по-другому, применим к этой колонке технологию one-hot encoding. Что получилось? – создали по самим для кого.

Результат этого действия создадим так столбца, каждый с названием одного из значений колонки Embarked. Если в исходном столбце значение было равно, то в соответствующем столбце будет стоять 1, иначе – 0. Если не совсем понятно, посмотрите на пример ниже.

В Pandas для этого есть специальная функция:

```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html
data = pd.get_dummies(data, columns=['Embarked'])
```

```
data.head()
```

	Pclass	Sex	Age	Sibsp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S
0	3	1	22.0	1	0	7.2500	0	0	1
1	1	0	38.0	1	0	71.2833	1	0	0
2	3	0	26.0	0	0	7.9250	0	0	1
3	1	0	35.0	1	0	53.1000	0	0	1
4	3	1	35.0	0	0	8.0500	0	0	1

Отлично! Мы замещение предобработку признаков делается.

Осталось разбить данные на тренировочный и контрольный датасет и можно обучать модель.

Разбиение данных на train и val

```
from sklearn.model_selection import train_test_split
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
train_X, val_X, train_y, val_y = train_test_split(data, X, test_size=0.3)
```

Часть 2. Обучение модели.

```
from sklearn.neighbors import KNeighborsClassifier
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
knn = KNeighborsClassifier(n_neighbors=3)
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(train_data, train_y)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

```
from sklearn.metrics import accuracy_score
```

```
predicted = knn.predict(val_data)
predicted
```

```
array([1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```