data-science_part2_basics_stack1

November 18, 2022

1 Part 2: Basics

1.1 Load Titanic data

- Downloaded from here.
- Description on Kaggle

[44]:
```
# import `pandas` which is one of the main libraries for data analytics in␣
↪Python:
# website: https://pandas.pydata.org/
import pandas as pd
```

[50]:
```
# load titanic data
# we set the index of the data to `PassengerId`
data_titanic = pd.read_csv("titanic.csv", index_col="PassengerId")
data_titanic
```

[50]:

PassengerId	Survived	Pclass	Name	Sex	Age
1	0	3	Braund, Mr. Owen Harris	male	22.0
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0
3	1	3	Heikkinen, Miss. Laina	female	26.0
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
5	0	3	Allen, Mr. William Henry	male	35.0
…	…	…			
887	0	2			
888	1	1			
889	0	3			
890	1	1			
891	0	3			

```
…                                                  …     …    …
887                                 Montvila, Rev. Juozas    male  27.0
888                          Graham, Miss. Margaret Edith  female  19.0
889              Johnston, Miss. Catherine Helen "Carrie"  female   NaN
890                                 Behr, Mr. Karl Howell    male  26.0
891                                   Dooley, Mr. Patrick    male  32.0

             SibSp  Parch            Ticket     Fare Cabin Embarked
PassengerId
1                1      0         A/5 21171   7.2500   NaN        S
2                1      0          PC 17599  71.2833   C85        C
3                0      0  STON/O2. 3101282   7.9250   NaN        S
4                1      0            113803  53.1000  C123        S
5                0      0            373450   8.0500   NaN        S
…              …    …                …       …    …      …
887              0      0            211536  13.0000   NaN        S
888              0      0            112053  30.0000   B42        S
889              1      2        W./C. 6607  23.4500   NaN        S
890              0      0            111369  30.0000  C148        C
891              0      0            370376   7.7500   NaN        Q

[891 rows x 11 columns]
```

1.2 Predict survival (Survived)

[52]:
```
# define the target (what we want to predict)
y = data_titanic["Survived"]
y
```

[52]:
```
PassengerId
1      0
2      1
3      1
4      1
5      0
      ..
887    0
888    1
889    0
890    1
891    0
Name: Survived, Length: 891, dtype: int64
```

[53]:
```
# define the set of input features (the data we want to use to predict survival)
X = data_titanic.drop(columns="Survived")
```

[57]:
```
# import a linear regression from scikit-learn
# scikit-learn is the main library for machine learning in Python
# website: https://scikit-learn.org
from sklearn.linear_model import LogisticRegression
```

[56]:
```
# define our model (many different models possible, we heard about k-nearest␣
 ↪neighbors and decision trees)
model = LogisticRegression()
```

[58]:
```
# try fit the model
model.fit(X, y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [58], line 2
      1 # try fit the model
----> 2 model.fit(X, y)

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪linear_model/_logistic.py:1508, in LogisticRegression.fit(self, X, y,␣
 ↪sample_weight)
   1505 else:
   1506     _dtype = [np.float64, np.float32]
-> 1508 X, y = self._validate_data(
   1509     X,
   1510     y,
   1511     accept_sparse="csr",
   1512     dtype=_dtype,
   1513     order="C",
   1514     accept_large_sparse=solver not in ["liblinear", "sag", "saga"],
   1515 )
   1516 check_classification_targets(y)
   1517 self.classes_ = np.unique(y)

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪base.py:581, in BaseEstimator._validate_data(self, X, y, reset,␣
 ↪validate_separately, **check_params)
    579         y = check_array(y, **check_y_params)
    580     else:
--> 581         X, y = check_X_y(X, y, **check_params)
    582     out = X, y
    584 if not no_val_X and check_params.get("ensure_2d", True):

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪utils/validation.py:964, in check_X_y(X, y, accept_sparse,␣
 ↪accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d,␣
 ↪allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric,␣
 ↪estimator)
```

```
    961 if y is None:
    962     raise ValueError("y cannot be None")
--> 964 X = check_array(
    965     X,
    966     accept_sparse=accept_sparse,
    967     accept_large_sparse=accept_large_sparse,
    968     dtype=dtype,
    969     order=order,
    970     copy=copy,
    971     force_all_finite=force_all_finite,
    972     ensure_2d=ensure_2d,
    973     allow_nd=allow_nd,
    974     ensure_min_samples=ensure_min_samples,
    975     ensure_min_features=ensure_min_features,
    976     estimator=estimator,
    977 )
    979 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric)
    981 check_consistent_length(X, y)

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪utils/validation.py:746, in check_array(array, accept_sparse,␣
 ↪accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d,␣
 ↪allow_nd, ensure_min_samples, ensure_min_features, estimator)
    744         array = array.astype(dtype, casting="unsafe", copy=False)
    745     else:
--> 746         array = np.asarray(array, order=order, dtype=dtype)
    747 except ComplexWarning as complex_warning:
    748     raise ValueError(
    749         "Complex data not supported\n{}\n".format(array)
    750     ) from complex_warning

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/pandas
 ↪core/generic.py:2064, in NDFrame.__array__(self, dtype)
   2063 def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064     return np.asarray(self._values, dtype=dtype)

ValueError: could not convert string to float: 'Braund, Mr. Owen Harris'
```

Fitting the model failed with the error `ValueError: could not convert string to float: 'Braund, Mr. Owen Harris'`. This is, because `LogisticRegression` can only handle numbers. But the titanic dataset contains many variables that are not numbers, such as `Name` which containts the string `'Braund, Mr. Owen Harris'`.

[62]:
```
# select columns that are numbers
# For this, we use `numpy` the 'fundamental package for scientific computing␣
 ↪with Python'.
# website: https://numpy.org/
```

```
import numpy as np
X.select_dtypes(np.number).columns
```

[62]:
```
Index(['Pclass', 'Age', 'SibSp', 'Parch', 'Fare'], dtype='object')
```

[63]:
```
# define a new set of input features that only contains numeric features
X_numeric = X[X.select_dtypes(np.number).columns]
X_numeric
```

[63]:

PassengerId	Pclass	Age	SibSp	Parch	Fare
1	3	22.0	1	0	7.2500
2	1	38.0	1	0	71.2833
3	3	26.0	0	0	7.9250
4	1	35.0	1	0	53.1000
5	3	35.0	0	0	8.0500
…	…	…	…	…	…
887	2	27.0	0	0	13.0000
888	1	19.0	0	0	30.0000
889	3	NaN	1	2	23.4500
890	1	26.0	0	0	30.0000
891	3	32.0	0	0	7.7500

[891 rows x 5 columns]

[65]:
```
# try to fit our model again
model.fit(X_numeric, y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [65], line 2
      1 # try to fit our model again
----> 2 model.fit(X_numeric, y)

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪linear_model/_logistic.py:1508, in LogisticRegression.fit(self, X, y,␣
 ↪sample_weight)
   1505 else:
   1506     _dtype = [np.float64, np.float32]
-> 1508 X, y = self._validate_data(
   1509     X,
   1510     y,
   1511     accept_sparse="csr",
   1512     dtype=_dtype,
   1513     order="C",
   1514     accept_large_sparse=solver not in ["liblinear", "sag", "saga"],
   1515 )
```

```
   1516 check_classification_targets(y)
   1517 self.classes_ = np.unique(y)

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪base.py:581, in BaseEstimator._validate_data(self, X, y, reset,␣
 ↪validate_separately, **check_params)
    579         y = check_array(y, **check_y_params)
    580     else:
--> 581         X, y = check_X_y(X, y, **check_params)
    582     out = X, y
    584 if not no_val_X and check_params.get("ensure_2d", True):

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪utils/validation.py:964, in check_X_y(X, y, accept_sparse,␣
 ↪accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d,␣
 ↪allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric,␣
 ↪estimator)
    961 if y is None:
    962     raise ValueError("y cannot be None")
--> 964 X = check_array(
    965     X,
    966     accept_sparse=accept_sparse,
    967     accept_large_sparse=accept_large_sparse,
    968     dtype=dtype,
    969     order=order,
    970     copy=copy,
    971     force_all_finite=force_all_finite,
    972     ensure_2d=ensure_2d,
    973     allow_nd=allow_nd,
    974     ensure_min_samples=ensure_min_samples,
    975     ensure_min_features=ensure_min_features,
    976     estimator=estimator,
    977 )
    979 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric)
    981 check_consistent_length(X, y)

File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪utils/validation.py:800, in check_array(array, accept_sparse,␣
 ↪accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d,␣
 ↪allow_nd, ensure_min_samples, ensure_min_features, estimator)
    794         raise ValueError(
    795             "Found array with dim %d. %s expected <= 2."
    796             % (array.ndim, estimator_name)
    797         )
    799     if force_all_finite:
--> 800         ␣
 ↪_assert_all_finite(array, allow_nan=force_all_finite == "allow-nan")
    802 if ensure_min_samples > 0:
    803     n_samples = _num_samples(array)
```

```
File ~/miniconda3/envs/teaching-datascience/lib/python3.10/site-packages/sklear /
 ↪utils/validation.py:114, in _assert_all_finite(X, allow_nan, msg_dtype)
    107     if (
    108         allow_nan
    109         and np.isinf(X).any()
    110         or not allow_nan
    111         and not np.isfinite(X).all()
    112     ):
    113         type_err = "infinity" if allow_nan else "NaN, infinity"
--> 114         raise ValueError(
    115             msg_err.format(
    116                 type_err, msg_dtype if msg_dtype is not None else X.dty e
    117             )
    118         )
    119 # for object dtype data, we only check for NaNs (GH-13254)
    120 elif X.dtype == np.dtype("object") and not allow_nan:

ValueError: Input contains NaN, infinity or a value too large for␣
 ↪dtype('float64').
```

This time, we see the error `ValueError: Input contains NaN, infinity or a value too large for dtype('float64')`. Which means that there are columns that contain empty fields. Let's have a look at them.

[69]:
```
# we see that the column `Age` contains `177` empty values
X_numeric.isna().sum()
```

[69]:
```
Pclass      0
Age       177
SibSp       0
Parch       0
Fare        0
dtype: int64
```

[75]:
```
# for now let's drop all rows with NAs
# IMPORTANT: this looses A LOT of data which should only be done as a last␣
 ↪resort in practice.
# We will learn about feature imputation which would be an alternative.
idx_notna = ~ X_numeric["Age"].isna()
X_numeric_notna = X_numeric[idx_notna]
y_notna = y[idx_notna]
```

[78]:
```
# fit the model AGAIN
model.fit(X_numeric_notna, y_notna)
```

[78]:
```
LogisticRegression()
```

[80]:
```
# predict
model.predict(X_numeric_notna)
```

[80]:
```
array([0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0])
```

[ ]: