Introduction to TensorFlow

Fully Connected Deep Networks with TensorFlow

郭耀仁

大綱

- 關於 Fully Connected Deep Networks
- 取得資料
- Benchmark
- 建構 TensorFlow 計算圖形
- 訓練
- 隨堂練習

關於 Fully Connected Deep Networks

Fully Connected Deep Networks 多層感知器

單一個感知器（Perceptron）的公式：

$$y = \sigma(WX + b)$$

σ 即所謂的激活函數（activation function）

激活函數用來將線性關係映射至非線性關係

常用的激活函數：

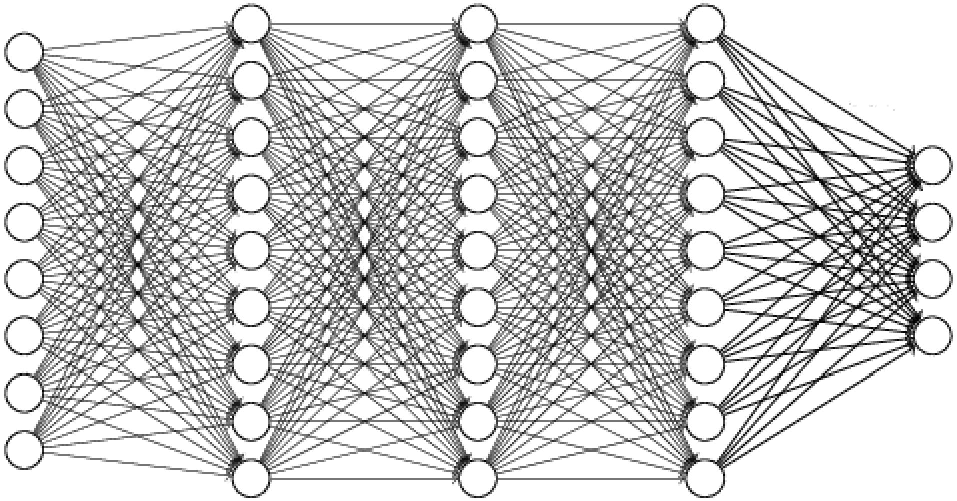
- Sigmoid
- TanH
- ReLU

連結多個感知器建構出多層感知器，透過 propagations 求出係數

- 輸入層
- 隱藏層
- 輸出層

什麼是 propagations（傳播）？

- Forward Propagation: 第一個隨機猜測
- Backward Propagation： 訓練



Source: https://math.stackexchange.com/questions/2048722/a-name-for-layered-directed-graph-as-in-a-fully-connected-neural-network (https://math.stackexchange.com/questions/2048722/a-name-for-layered-directed-graph-as-in-a-fully-connected-neural-network)

動手做一個 Fully Connected Deep Networks

先做 Forward Propagation

In [1]:

```
import numpy as np

class Neural_Network:
    def __init__(self, input_size, hidden_size, output_size):
        self._input_size = input_size
        self._hidden_size = hidden_size
        self._output_size = output_size
        self.W0 = np.random.randn(self._input_size, self._hidden_size)
        self.W1 = np.random.randn(self._hidden_size, self._output_size)
    def forward(self, X):
        self.z0 = np.dot(X, self.W0)
        self.z1 = self.sigmoid(self.z0)
        self.z2 = np.dot(self.z1, self.W1)
        output = self.sigmoid(self.z2)
        return output
    def sigmoid(self, x):
        return 1/(1 + np.exp(-x))
```

給定的 X 與 y

- 神經網路重視資料的標準化（Standard Scaling）
  - 使用了 Activation Function 的緣故
  - Standard Scaler
  - MinMax Scaler

In [2]:

```python
import numpy as np

def min_max_scaler(x):
    x_min = x.min(axis=0)
    x_max = x.max(axis=0)
    return (x - x_min) / (x_max - x_min)
```

In [3]:

```python
X = np.linspace(1, 30, 30).reshape(-1, 1)
y = 2*X + 3
print(X.shape)
print(y.shape)
print("======")
print(X[:5])
print("======")
print(y[:5])
```

```
(30, 1)
(30, 1)
======
[[1.]
 [2.]
 [3.]
 [4.]
 [5.]]
======
[[ 5.]
 [ 7.]
 [ 9.]
 [11.]
 [13.]]
```

In [4]:
```
X_scaled = min_max_scaler(X)
y_scaled = min_max_scaler(y)
print(X_scaled[:5])
print("======")
print(y_scaled[:5])
```

```
[[0.        ]
 [0.03448276]
 [0.06896552]
 [0.10344828]
 [0.13793103]]
======
[[0.        ]
 [0.03448276]
 [0.06896552]
 [0.10344828]
 [0.13793103]]
```

In [5]:

```python
NN = Neural_Network(1, 4, 1)
y_hat = NN.forward(X_scaled)

print("y_hat:")
print(y_hat.ravel())
print("======")
print("y:")
print(y_scaled.ravel())
```

```
y_hat:
[0.4742546  0.47088419 0.46752178 0.46417296 0.46084327 0.45753813
 0.45426283 0.45102252 0.44782214 0.44466646 0.44156002 0.43850709
 0.43551173 0.4325777  0.42970851 0.42690734 0.42417713 0.42152049
 0.41893976 0.41643696 0.41401385 0.41167187 0.40941222 0.40723581
 0.4051433  0.40313508 0.40121133 0.39937198 0.39761678 0.39594525]
======
y:
[0.         0.03448276 0.06896552 0.10344828 0.13793103 0.17241379
 0.20689655 0.24137931 0.27586207 0.31034483 0.34482759 0.37931034
 0.4137931  0.44827586 0.48275862 0.51724138 0.55172414 0.5862069
 0.62068966 0.65517241 0.68965517 0.72413793 0.75862069 0.79310345
 0.82758621 0.86206897 0.89655172 0.93103448 0.96551724 1.        ]
```

再做 Back Propagation

- 計算輸出層的 Loss
- 將 Loss 輸入 sigmoid 導函數，計算 W_1 的 Loss 比例
- W_1 的 Loss 再輸入 sigmoid 導函數，計算 W_0 的 Loss 比例
- 依據梯度遞減調整 W_1 與 W_0

In [6]:

```
import numpy as np

class Neural_Network:
    def __init__(self, input_size, hidden_size, output_size):
        self._input_size = input_size
        self._hidden_size = hidden_size
        self._output_size = output_size
        self.W0 = np.random.randn(self._input_size, self._hidden_size)
        self.W1 = np.random.randn(self._hidden_size, self._output_size)
    def forward(self, X):
        self.z0 = np.dot(X, self.W0)
        self.z1 = self.sigmoid(self.z0)
        self.z2 = np.dot(self.z1, self.W1)
        output = self.sigmoid(self.z2)
        return output
    def sigmoid(self, x):
        return 1/(1 + np.exp(-x))
    def sigmoid_derivative(self, x):
        return x*(1-x)
    def backward(self, X, y, output):
        self.error = y - output
        self.output_delta = self.error * self.sigmoid_derivative(output)
        self.z1_error = np.dot(self.output_delta, self.W1.T)
        self.z1_delta = self.z1_error * self.sigmoid_derivative(self.z1)
        self.W0 += np.dot(X.T, self.z1_delta)
        self.W1 += np.dot(self.z1.T, self.output_delta)
    def train(self, X, y):
        output = self.forward(X)
        self.backward(X, y, output)
    def get_weights(self):
        return self.W0, self.W1
    def predict(self, X_test):
        return self.forward(X_test)
```

In [7]:

```
NN = Neural_Network(1, 4, 1)
for i in range(10000):
    for x_, y_ in zip(X_scaled, y_scaled):
        NN.train(x_.reshape(-1, 1), y_.reshape(-1, 1))
    loss = np.mean(((y_scaled - NN.forward(X_scaled))**2).sum())
    if i % 1000 == 0:
        print("Epoch: {}, Loss: {}".format(i, loss))
```

```
Epoch: 0, Loss: 3.2321717227043365
Epoch: 1000, Loss: 0.0381216257861632l
Epoch: 2000, Loss: 0.036089745001124524
Epoch: 3000, Loss: 0.03529837545376917
Epoch: 4000, Loss: 0.03490766381521602
Epoch: 5000, Loss: 0.034702067612926185
Epoch: 6000, Loss: 0.03459881476904855
Epoch: 7000, Loss: 0.03455858545240497
Epoch: 8000, Loss: 0.03456000817600267
Epoch: 9000, Loss: 0.03459032993787807
```

In [8]:
```python
weight_0, weight_1 = NN.get_weights()
print(weight_0)
print(weight_1)
```

```
[[-0.89827987  0.90333068 -0.89808679 -0.9035581 ]]
[[-4.30576809]
 [ 9.52137165]
 [-3.89353405]
 [-6.67238022]]
```

In [9]:
```
X_test = X_scaled[:3]
y_hat_scaled = NN.predict(X_test)
y_hat = y.min() + (y.max() - y.min())*y_hat_scaled
print(y_hat)
print(y[:3])
```

```
[[ 8.73841305]
 [ 9.45883124]
 [10.30406586]]
[[5.]
 [7.]
 [9.]]
```

取得資料

簡單、作為測試目的即可

Scikit-Learn Breast Cancer 資料集

In [10]:
```python
from sklearn.datasets import load_breast_cancer

breast_cancer = load_breast_cancer()
print(breast_cancer.feature_names)
print(breast_cancer.DESCR)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
```

- fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

- class:
  - WDBC-Malignant
  - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54

```
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
```

```
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extractio
n
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis a
nd
     prognosis via linear programming. Operations Research, 43(4), pages 570-5
77,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniq
ues
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77
(1994)
     163-171.
```

In [11]:

```
X_arr = breast_cancer.data
y_arr = breast_cancer.target
print(X_arr.shape)
print(y_arr.shape)
```

```
(569, 30)
(569,)
```

In [12]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_arr, y_arr, test_size=0.3, r
andom_state=123)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(398, 30)
(171, 30)
(398,)
(171,)
```

Benchmark

In [13]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

clf = LogisticRegression(solver="liblinear")
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
```

In [14]:

```python
print(clf.coef_)
print(clf.intercept_)
print(acc)
```

```
[[ 1.93788057e+00  1.94992830e-01 -2.93534745e-03 -9.97160018e-03
  -1.31053304e-01 -3.94404828e-01 -5.87789964e-01 -3.08240545e-01
  -2.34487470e-01 -2.08028628e-02 -1.37757133e-03  1.25279091e+00
   8.06843035e-03 -8.58257925e-02 -1.44335700e-02 -4.29282110e-04
  -4.53672393e-02 -3.74242144e-02 -3.87957326e-02  6.72139227e-03
   1.00102118e+00 -3.88323187e-01 -1.28745720e-01 -1.78899170e-02
  -2.50197148e-01 -1.09822573e+00 -1.45824679e+00 -6.35170781e-01
  -7.02595973e-01 -9.59181366e-02]]
[0.41185323]
0.9824561403508771
```

Benchmark 完整程式碼

In [15]:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

breast_cancer = load_breast_cancer()
X_arr = breast_cancer.data
y_arr = breast_cancer.target
X_train, X_test, y_train, y_test = train_test_split(X_arr, y_arr, test_size=0.3, r
andom_state=123)
clf = LogisticRegression(solver="liblinear")
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
```

建構 TensorFlow 計算圖形

準備 Placeholders 作為 Input layers

In [16]:

```
import tensorflow as tf

n_features = X_train.shape[1]
with tf.name_scope("input-layer"):
    X = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
```

Placeholders 不指定外觀的好處

- 多層感知器的應用常會使用 **Mini-batching** 的技巧
- 訓練結束之後可以餵入 X_test 獲得 y_pred_arr

準備 Variables 作為 Hidden layers

In [17]:

```
import numpy as np

n_classes = np.unique(y_train).size
n_neurons = 2**4
with tf.name_scope("hidden-layer"):
    W = tf.Variable(tf.random_normal((n_features, n_neurons))) # (30, 16)
    b = tf.Variable(tf.random_normal((n_neurons,)))            # (16,)
    X_hidden = tf.nn.relu(tf.matmul(X, W) + b)                 # (398, 30) x (30,
 16) + (16,)
```

```
WARNING:tensorflow:From /Users/kuoyaojen/anaconda3/envs/tensorflow/lib/python
3.6/site-packages/tensorflow/python/framework/op_def_library.py:263: colocate_
with (from tensorflow.python.framework.ops) is deprecated and will be removed
in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

如何決定 hidden-layer 的層數與神經元數量?

- 絕大多數情境：使用一層 hidden-layer 就足夠
- 神經元數量：2^n

準備 Variables 作為 Output layers

In [18]:

```
with tf.name_scope("output-layer"):
    W = tf.Variable(tf.random_normal((n_neurons, 1))) # (16, 1)
    b = tf.Variable(tf.random_normal((1,)))           # (1,)
    y_logit = tf.squeeze(tf.add(tf.matmul(X_hidden, W), b))  # (398, 16) x (16, 1)
+ (1,)
    y_one_prob = tf.sigmoid(y_logit)
    y_pred = tf.round(y_one_prob)
```

寫下成本函數的公式

In [19]:

```
with tf.name_scope("loss"):
    entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit, labels=y)
    loss = tf.reduce_sum(entropy)
```

宣告 Optimizer 與學習速率

In [20]:
```
learning_rate = 0.0001
with tf.name_scope("optimizer"):
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

```
WARNING:tensorflow:From /Users/kuoyaojen/anaconda3/envs/tensorflow/lib/python
3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future versi
on.
Instructions for updating:
Use tf.cast instead.
```

建構 TensorFlow 計算圖形完整程式碼

In [21]:

```
import tensorflow as tf
import numpy as np

tf.reset_default_graph()
n_features = X_train.shape[1]
n_classes = np.unique(y_train).size
n_neurons = 2**4
learning_rate = 0.0001

with tf.name_scope("input-layer"):
    X = tf.placeholder(tf.float32)
    y = tf.placeholder(tf.float32)
with tf.name_scope("hidden-layer"):
    W = tf.Variable(tf.random_normal((n_features, n_neurons))) # (30, 16)
    b = tf.Variable(tf.random_normal((n_neurons,)))            # (16,)
    X_hidden = tf.nn.relu(tf.matmul(X, W) + b)                 # (398, 30) x (30,
 16) + (16,)
with tf.name_scope("output-layer"):
    W = tf.Variable(tf.random_normal((n_neurons, 1))) # (16, 1)
    b = tf.Variable(tf.random_normal((1,)))           # (1,)
    y_logit = tf.squeeze(tf.add(tf.matmul(X_hidden, W), b))  # (398, 16) x (16, 1)
+ (1,)
    y_one_prob = tf.sigmoid(y_logit)
    y_pred = tf.round(y_one_prob)
with tf.name_scope("loss"):
    entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_logit, labels=y)
    loss = tf.reduce_sum(entropy)
with tf.name_scope("optimizer"):
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

訓練

In [22]:

```
n_steps = 50000
file_writer_path = "./graphs/fully-connected-deep-networks"
loss_history = []

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer()) # 初始化所有的變數張量!
    train_writer = tf.summary.FileWriter(file_writer_path, tf.get_default_graph())
    for i in range(n_steps):
        feed_dict = {
            X: X_train,
            y: y_train
        }
        _, loss_ = sess.run([optimizer, loss], feed_dict=feed_dict)
        loss_history.append(loss_)
        if i % 1000 == 0:
            print("step {}, loss: {}".format(i, loss_))
    w_final, b_final = sess.run([W, b])
    y_pred_arr = sess.run(y_pred, feed_dict={X: X_test})
```

```
step 0, loss: 539639.75
step 1000, loss: 274796.96875
step 2000, loss: 37507.2578125
step 3000, loss: 5371.13427734375
step 4000, loss: 5262.8720703125
step 5000, loss: 5168.35546875
step 6000, loss: 5027.861328125
step 7000, loss: 4896.12841796875
step 8000, loss: 3382.2109375
step 9000, loss: 2716.5947265625
step 10000, loss: 2126.1630859375
step 11000, loss: 1527.1834716796875
step 12000, loss: 977.72314453125
step 13000, loss: 590.98681640625
step 14000, loss: 299.52642822265625
step 15000, loss: 104.7797622680664
step 16000, loss: 68.5936279296875
step 17000, loss: 59.4394416809082
```

In [23]:

```
import matplotlib.pyplot as plt

plt.plot(range(n_steps), loss_history)
plt.title("Loss Summary")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

In [24]:

```
acc = accuracy_score(y_test, y_pred_arr)
print(w_final)
print(b_final)
print(acc)
```

```
[[ 0.26682755]
 [-0.53566235]
 [-0.10864915]
 [ 1.4385455 ]
 [-0.6868179 ]
 [-1.319403  ]
 [ 0.5819109 ]
 [ 0.06051218]
 [ 0.2184967 ]
 [ 0.6089343 ]
 [-0.143741  ]
 [-1.5354853 ]
 [-0.6753923 ]
 [-1.8530135 ]
 [ 1.4804659 ]
 [ 0.6578026 ]]
[2.2221074]
0.9766081871345029
```

隨堂練習

使用 titanic 資料集並利用 TensorFlow 建立一個 Fully Connected Deep Networks Classifier

In [1]:

```
import pandas as pd

train_url = "https://s3-ap-northeast-1.amazonaws.com/kaggle-getting-started/titanic/train.csv"
train = pd.read_csv(train_url)
```

使用一個 Linear Regressor 填補 `Age` 變數的遺漏值

In [2]:

```
train.describe()
```

Out[2]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [17]:
```
from sklearn.linear_model import LinearRegression
import numpy as np

is_nan = np.isnan(train["Age"])
not_nan = ~(np.isnan(train["Age"]))
X_train = train[not_nan].loc[:, ["Pclass", "SibSp", "Parch", "Fare"]].values
y_train = train[not_nan]["Age"].values
regressor = LinearRegression()
regressor.fit(X_train, y_train)
X_test = train[is_nan].loc[:, ["Pclass", "SibSp", "Parch", "Fare"]].values
y_hat = regressor.predict(X_test)
```

In [ ]: