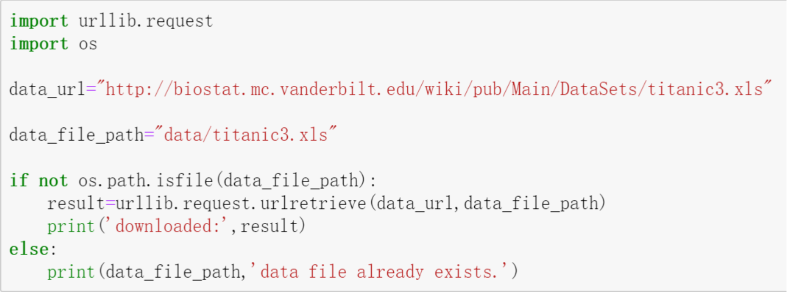
预测泰坦尼克号旅客生存概率

数据准备

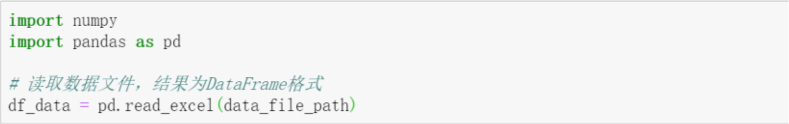
数据准备

下载泰坦尼克号旅客的数据集



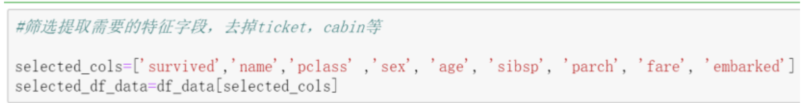
```python
import urllib.request
import os

data_url="http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls"

data_file_path="data/titanic3.xls"

if not os.path.isfile(data_file_path):
    result=urllib.request.urlretrieve(data_url,data_file_path)
    print('downloaded:',result)
else:
    print(data_file_path,'data file already exists.')
```

读取数据



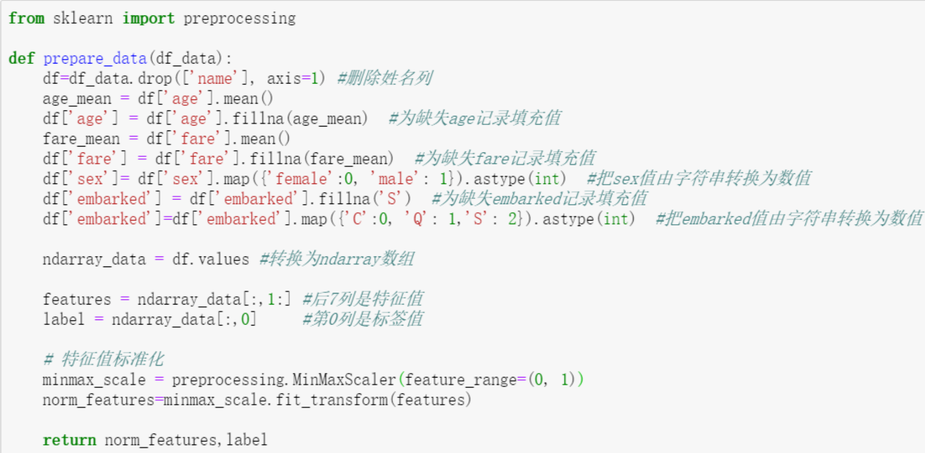
```python
import numpy
import pandas as pd

# 读取数据文件，结果为DataFrame格式
df_data = pd.read_excel(data_file_path)
```

筛选提取需要的特征字段



```python
#筛选提取需要的特征字段，去掉ticket，cabin等

selected_cols=['survived','name','pclass' ,'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked']
selected_df_data=df_data[selected_cols]
```

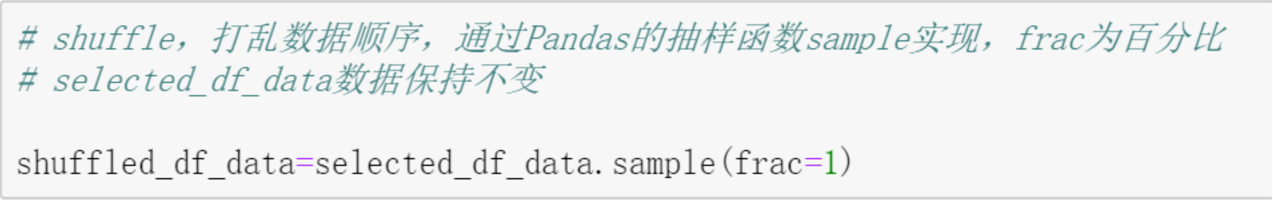
定义数据预处理函数



```python
from sklearn import preprocessing

def prepare_data(df_data):
    df=df_data.drop(['name'], axis=1) #删除姓名列
    age_mean = df['age'].mean()
    df['age'] = df['age'].fillna(age_mean)  #为缺失age记录填充值
    fare_mean = df['fare'].mean()
    df['fare'] = df['fare'].fillna(fare_mean)  #为缺失fare记录填充值
    df['sex']= df['sex'].map({'female':0, 'male': 1}).astype(int)  #把sex值由字符串转换为数值
    df['embarked'] = df['embarked'].fillna('S')  #为缺失embarked记录填充值
    df['embarked']=df['embarked'].map({'C':0, 'Q': 1,'S': 2}).astype(int)  #把embarked值由字符串转换为数值

    ndarray_data = df.values #转换为ndarray数组

    features = ndarray_data[:,1:] #后7列是特征值
    label = ndarray_data[:,0]     #第0列是标签值

    # 特征值标准化
    minmax_scale = preprocessing.MinMaxScaler(feature_range=(0, 1))
    norm_features=minmax_scale.fit_transform(features)

    return norm_features,label
```

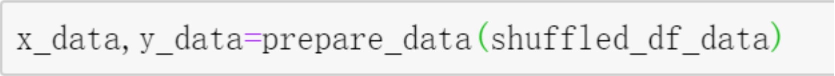
数据准备

shuffle，打乱数据顺序，为后面训练做准备



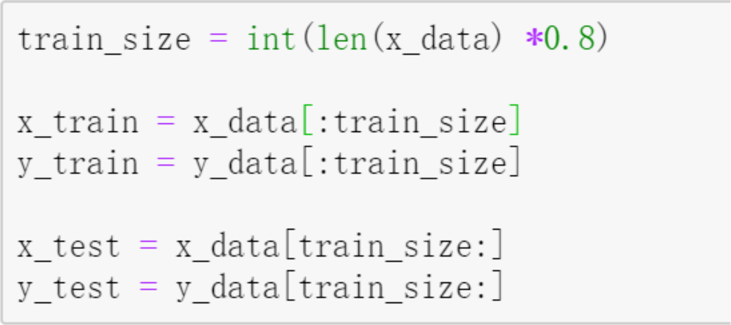
```
# shuffle，打乱数据顺序，通过Pandas的抽样函数sample实现，frac为百分比
# selected_df_data数据保持不变

shuffled_df_data=selected_df_data.sample(frac=1)
```

得到处理后的数据集



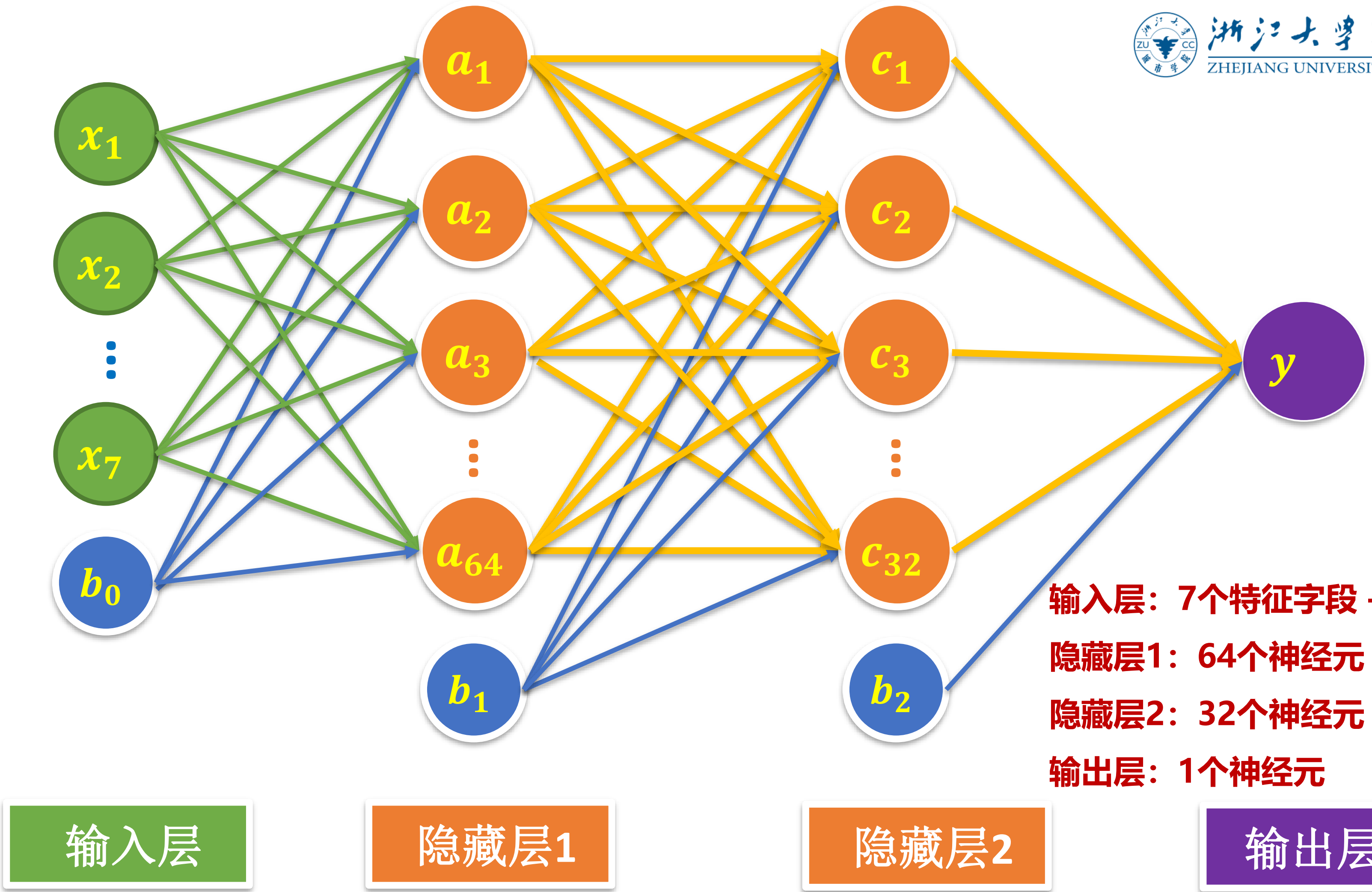
```
x_data,y_data=prepare_data(shuffled_df_data)
```

划分训练集和测试集



```
train_size = int(len(x_data) *0.8)

x_train = x_data[:train_size]
y_train = y_data[:train_size]

x_test = x_data[train_size:]
y_test = y_data[train_size:]
```

建立多层神经网络模型



输入层：7个特征字段 -> 7个神经元

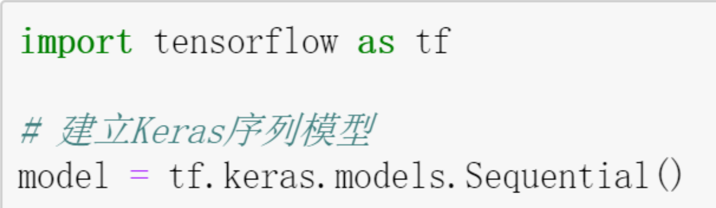
隐藏层1：64个神经元

隐藏层2：32个神经元

输出层：1个神经元

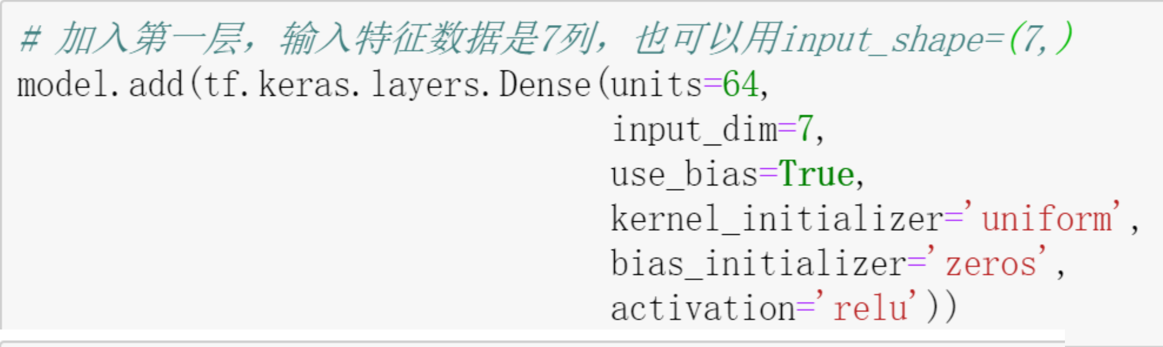
建立模型结构

1



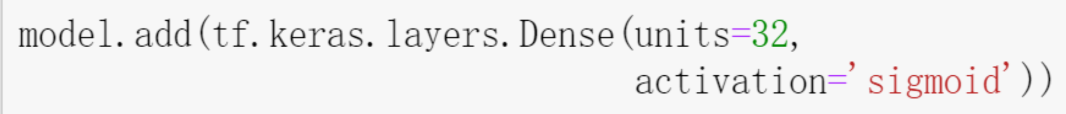
```
import tensorflow as tf

# 建立Keras序列模型
model = tf.keras.models.Sequential()
```

2



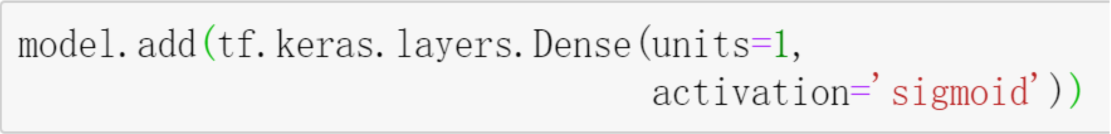
```
# 加入第一层，输入特征数据是7列，也可以用input_shape=(7,)
model.add(tf.keras.layers.Dense(units=64,
                                input_dim=7,
                                use_bias=True,
                                kernel_initializer=’uniform’,
                                bias_initializer=’zeros’,
                                activation=’relu’))
```

3

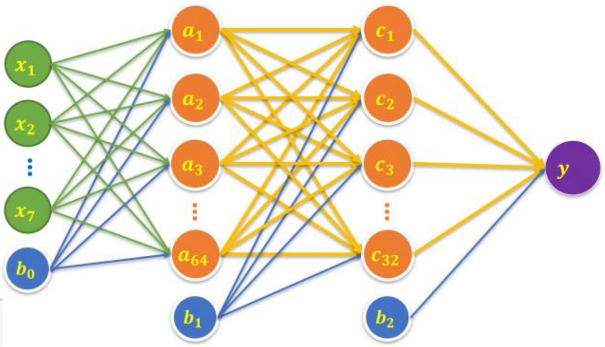


```
model.add(tf.keras.layers.Dense(units=32,
                                 activation=’sigmoid’))
```

4

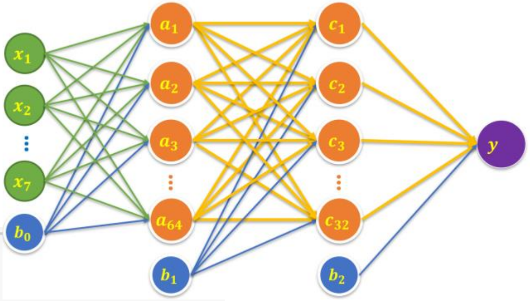


```
model.add(tf.keras.layers.Dense(units=1,
                                 activation=’sigmoid’))
```

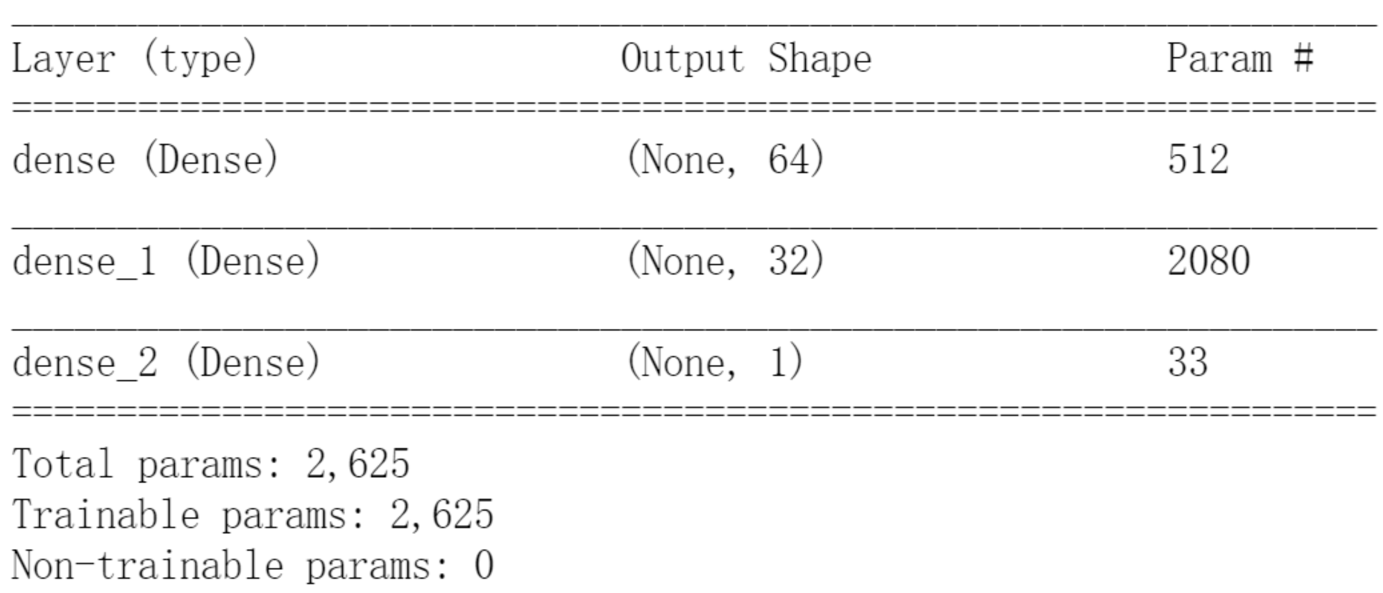


模型设置

模型结构

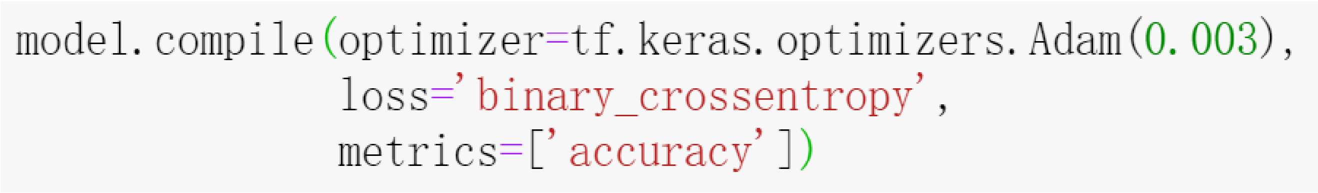


```
model.summary()
```

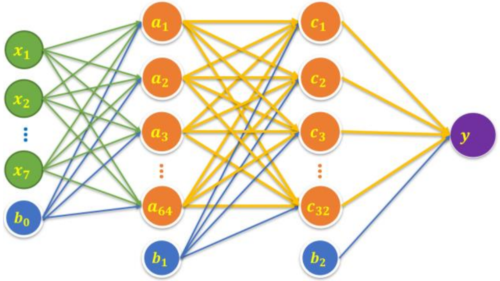


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                512
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,625
Trainable params: 2,625
Non-trainable params: 0
```

模型设置



```
model.compile(optimizer=tf.keras.optimizers.Adam(0.003),
              loss=’binary_crossentropy’,
              metrics=[’accuracy’])
```

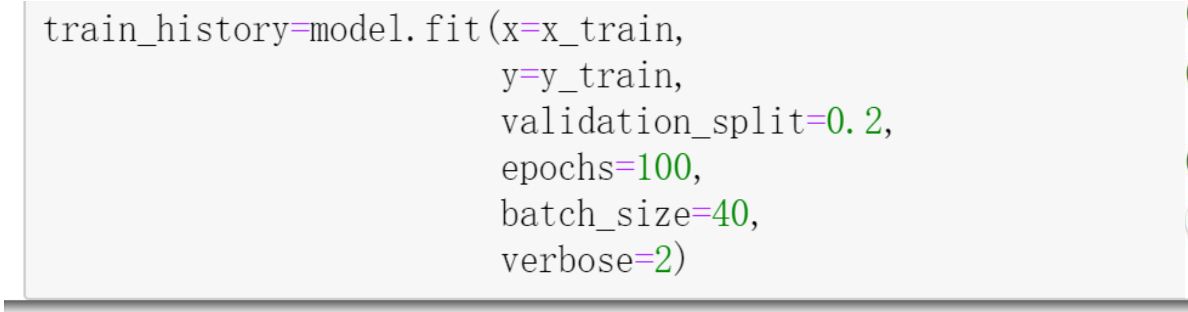


- optimizer可以是优化器的名字，如'adam'，也可以是优化器的实例
- loss是损失函数名
  - 用sigmoid作为激活函数，一般损失函数选用binary_crossentropy
  - 用softmax作为激活函数，一般损失函数选用categorical_crossentropy
- metrics 模型要训练和评估的度量值

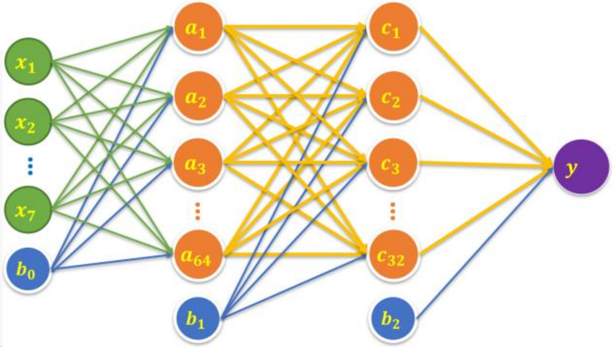
更多详细信息查阅API文档
https://www.tensorflow.org/versions/r1.10/api_docs/python/tf/keras/Model

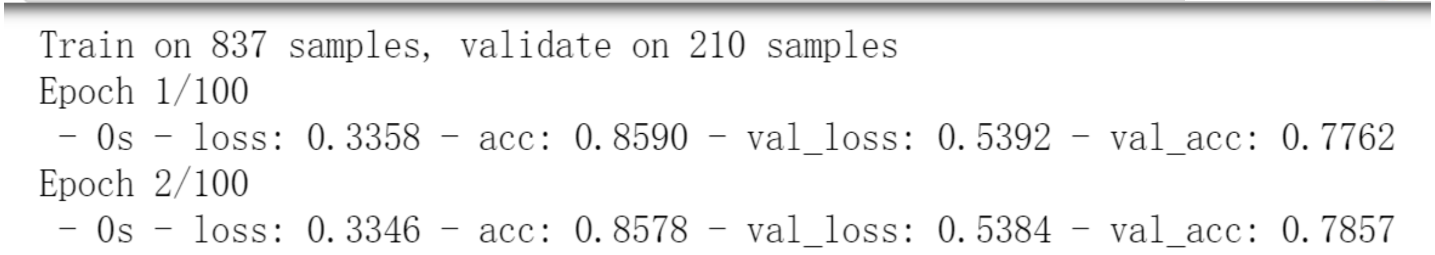
模型训练

模型训练



```
train_history=model.fit(x=x_train,
                        y=y_train,
                        validation_split=0.2,
                        epochs=100,
                        batch_size=40,
                        verbose=2)
```



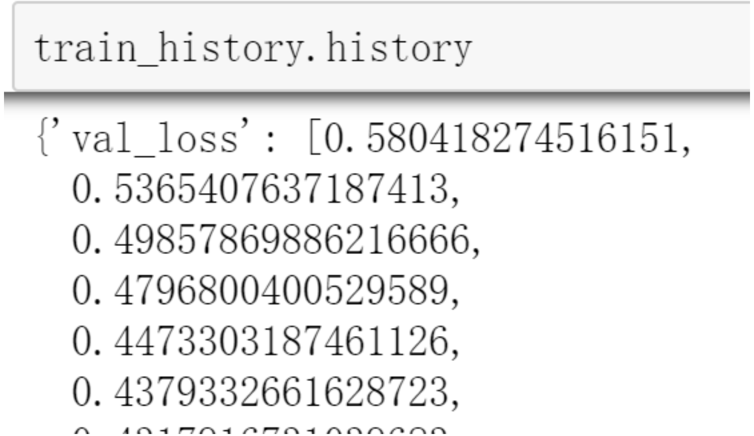


```
Train on 837 samples, validate on 210 samples
Epoch 1/100
 - 0s - loss: 0.3358 - acc: 0.8590 - val_loss: 0.5392 - val_acc: 0.7762
Epoch 2/100
 - 0s - loss: 0.3346 - acc: 0.8578 - val_loss: 0.5384 - val_acc: 0.7857
```

- x:输入的特征数据　　y:标签数据
- validation_split:验证集所占的比例
- verbose: 训练过程显示模式

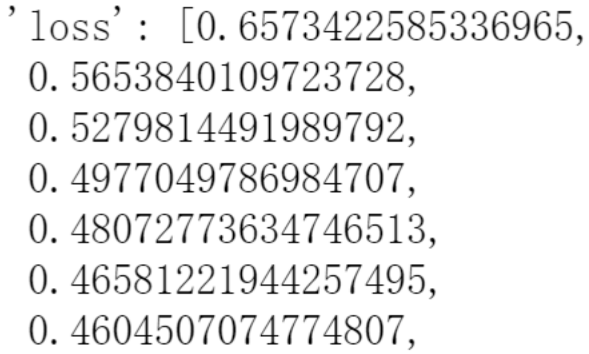
  取值 0：不显示，1：带进度条模式，2：每epoch显示一行
- 返回值：过程历史对象，包括训练过程的loss和acc数据，以及验证过程的（如果有）

模型训练

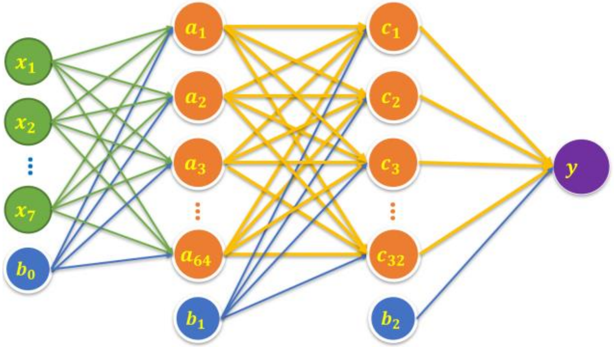


```
train_history.history
```

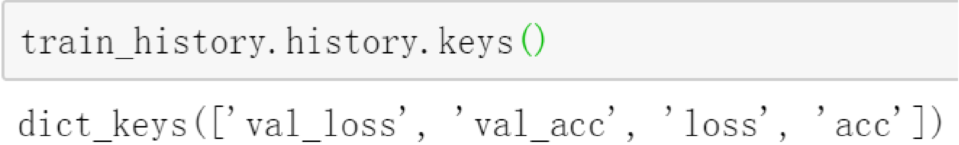
```
{’val_loss’: [0. 580418274516151,
  0. 5365407637187413,
  0. 49857869886216666,
  0. 4796800400529589,
  0. 447330318746 1126,
  0. 437933266162 8723,
  0. 4217916721020628
```



```
 ’loss’: [0. 6573422585336965,
  0. 5653840109723728,
  0. 5279814491989792,
  0. 4977049786984707,
  0. 4807277363474 6513,
  0. 4658122194425 7495,
  0. 4604507074774807,
```



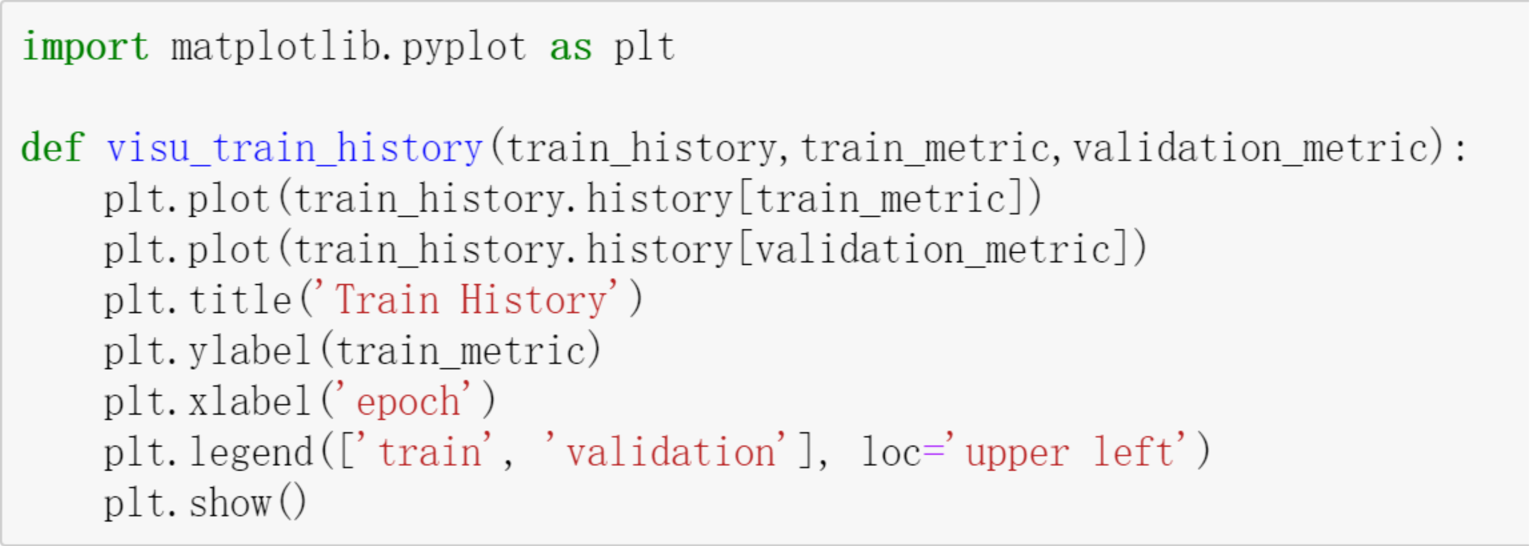
训练过程的历史数据：字典模式存储



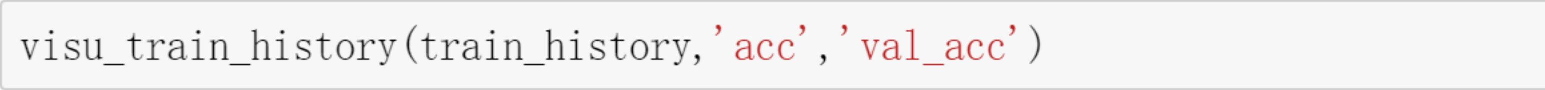
```
train_history.history.keys()
```

```
dict_keys([’val_loss’, ’val_acc’, ’loss’, ’acc’])
```

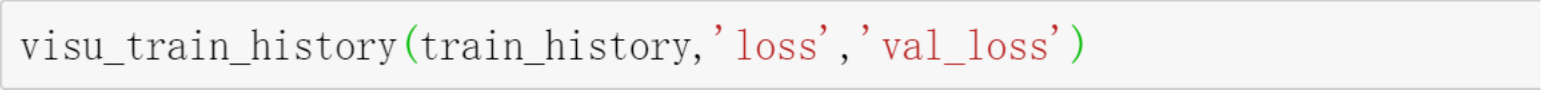
模型训练过程可视化



```
import matplotlib.pyplot as plt

def visu_train_history(train_history,train_metric,validation_metric):
    plt.plot(train_history.history[train_metric])
    plt.plot(train_history.history[validation_metric])
    plt.title(’Train History’)
    plt.ylabel(train_metric)
    plt.xlabel(’epoch’)
    plt.legend([’train’, ’validation’], loc=’upper left’)
    plt.show()
```

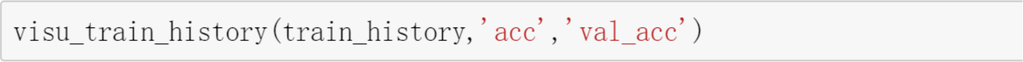


```
visu_train_history(train_history,’acc’,’val_acc’)
```

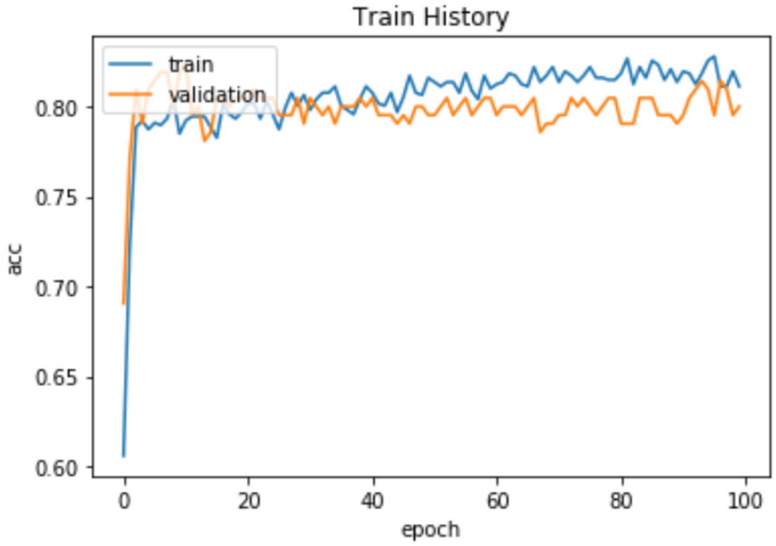


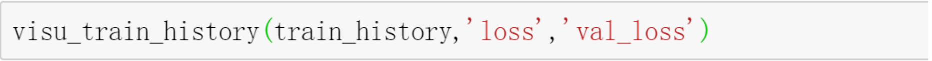
```
visu_train_history(train_history,’loss’,’val_loss’)
```

模型训练过程可视化

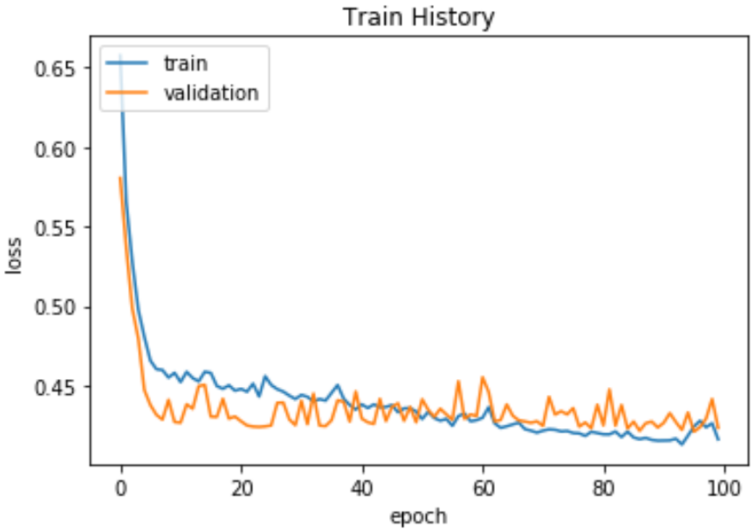


```
visu_train_history(train_history,'acc','val_acc')
```



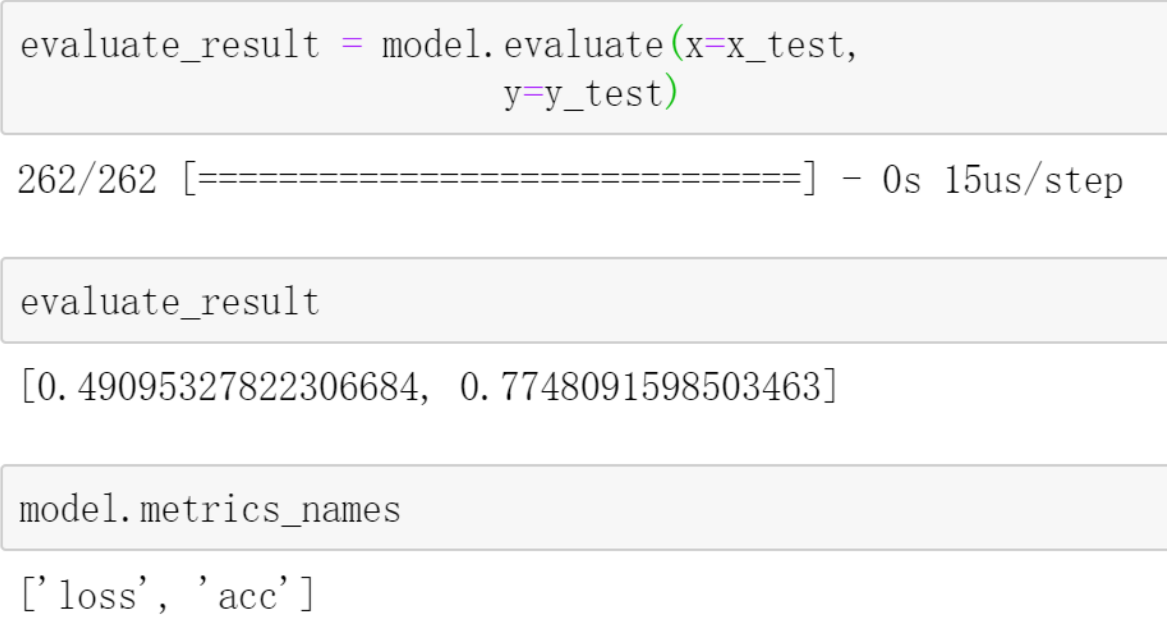


```
visu_train_history(train_history,'loss','val_loss')
```



模型评估

模型评估



```
evaluate_result = model.evaluate(x=x_test,
                                 y=y_test)
```

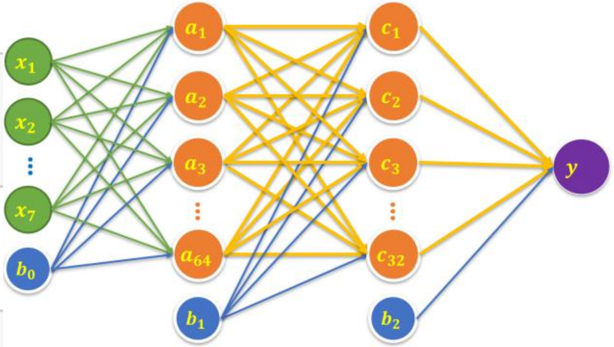
```
262/262 [==============================] - 0s 15us/step
```

```
evaluate_result
```

```
[0.49095327822306684, 0.7748091598503463]
```

```
model.metrics_names
```

```
['loss', 'acc']
```



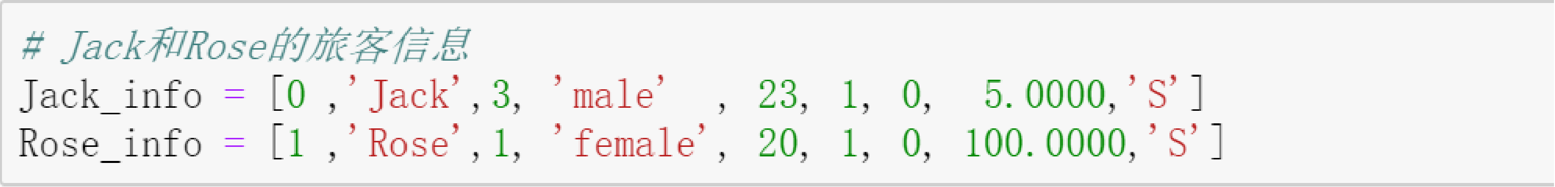
- **model.metrics_names:** 评估结果返回值的标签

模型应用

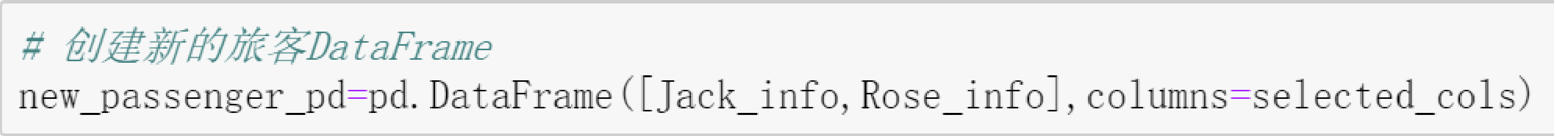
应用模型进行预测

加入Jack & Rose的数据

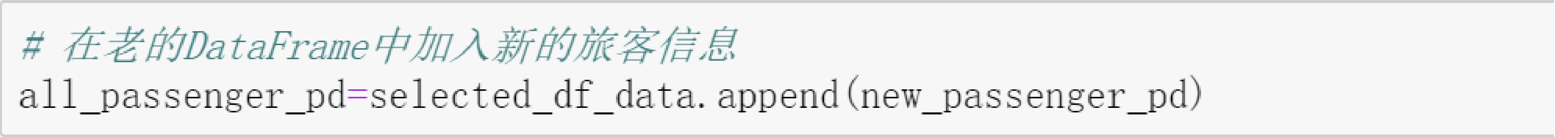
Jack： 3等舱，男性，票价5， 年龄 23； Rose：头等舱，女性，票价100，年龄20



```
# Jack和Rose的旅客信息
Jack_info = [0 ,’Jack’,3, ’male’   , 23, 1, 0,   5.0000,’S’]
Rose_info = [1 ,’Rose’,1, ’female’, 20, 1, 0, 100.0000,’S’]
```



```
# 创建新的旅客DataFrame
new_passenger_pd=pd.DataFrame([Jack_info,Rose_info],columns=selected_cols)
```

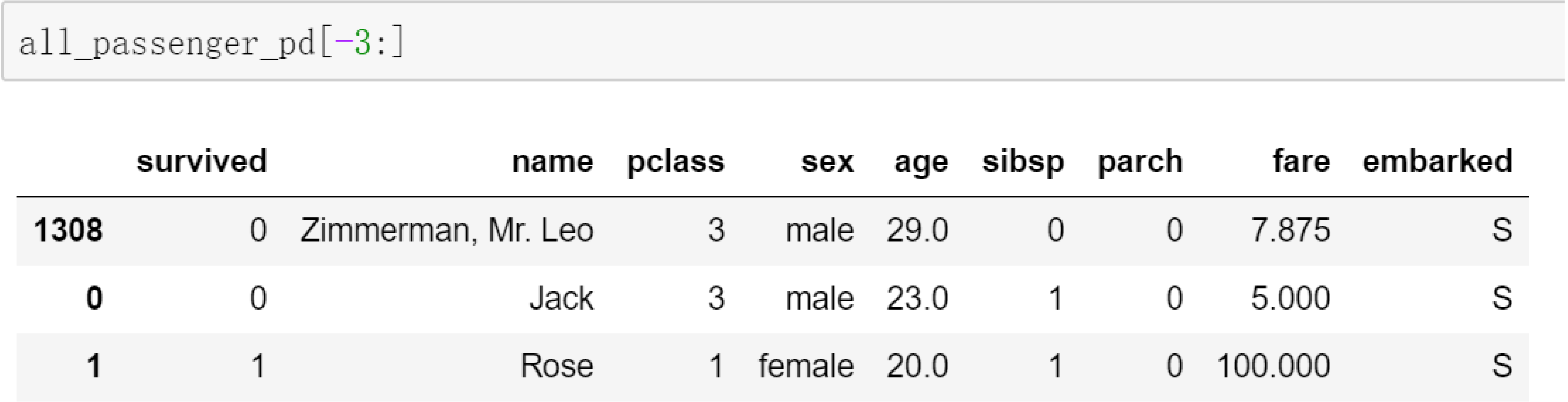


```
# 在老的DataFrame中加入新的旅客信息
all_passenger_pd=selected_df_data.append(new_passenger_pd)
```

应用模型进行预测

加入Jack & Rose的数据

Jack： 3等舱，男性，票价5， 年龄 23； Rose：头等舱，女性，票价100，年龄20

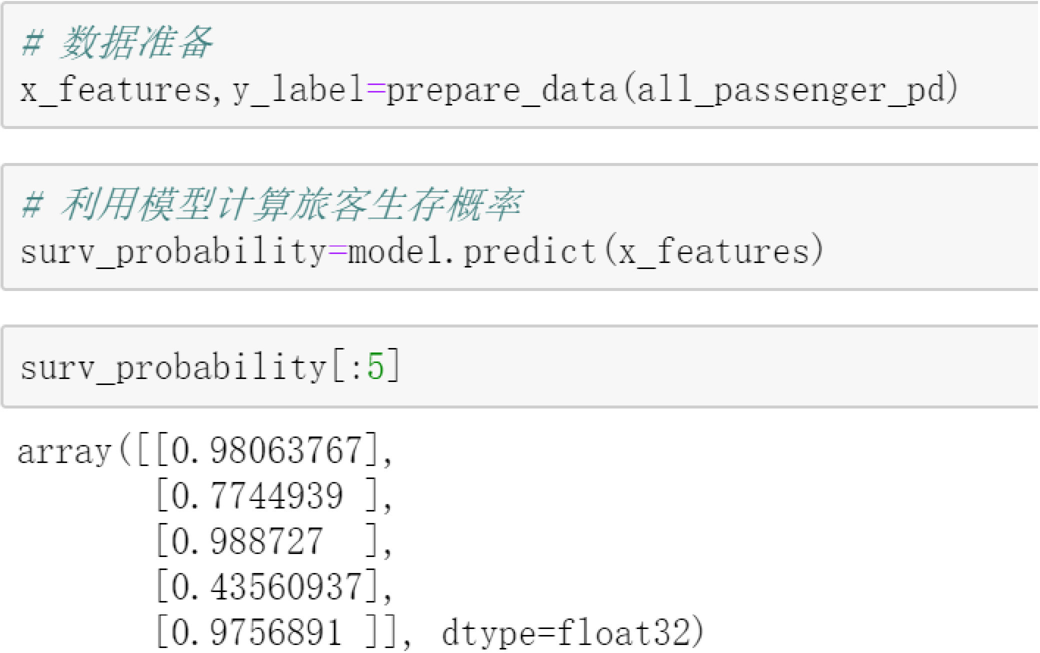


```
all_passenger_pd[-3:]
```

	survived	name	pclass	sex	age	sibsp	parch	fare	embarked
1308	0	Zimmerman, Mr. Leo	3	male	29.0	0	0	7.875	S
0	0	Jack	3	male	23.0	1	0	5.000	S
1	1	Rose	1	female	20.0	1	0	100.000	S

执行预测

再次执行数据预处理，然后使用model.predict执行预测



```
# 数据准备
x_features,y_label=prepare_data(all_passenger_pd)
```

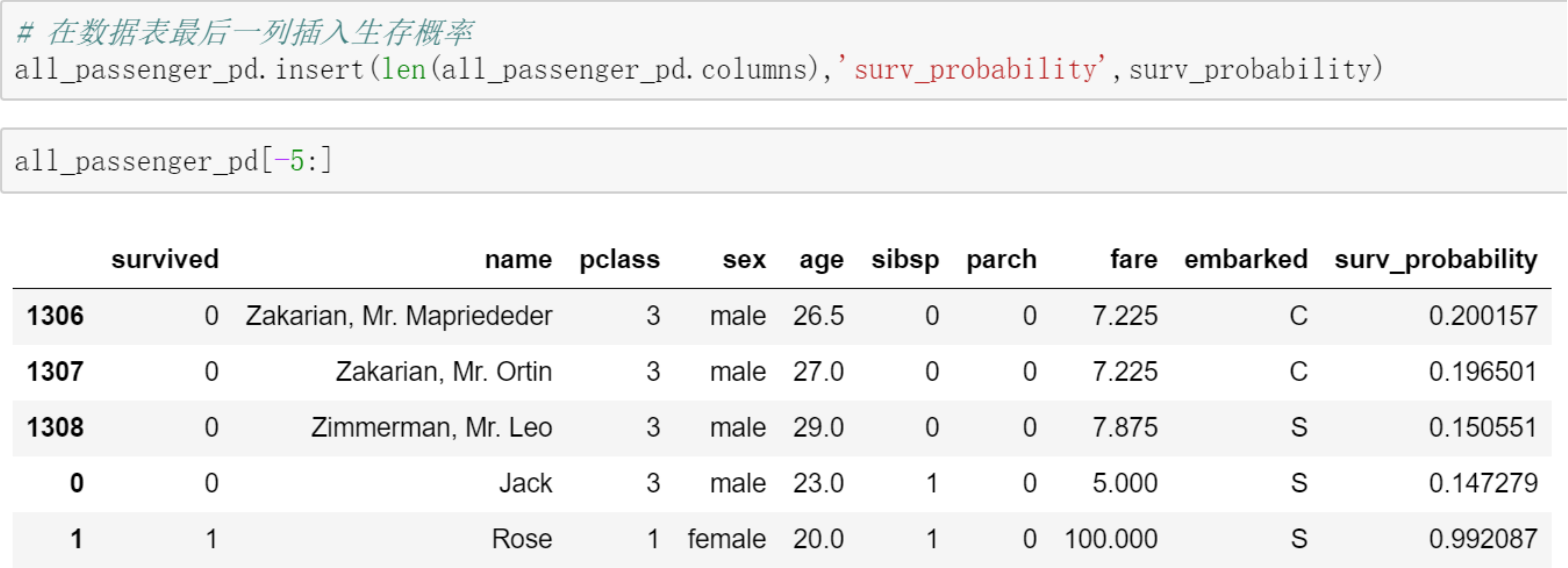
```
# 利用模型计算旅客生存概率
surv_probability=model.predict(x_features)
```

```
surv_probability[:5]
```

```
array([[0.98063767],
       [0.7744939 ],
       [0.988727  ],
       [0.43560937],
       [0.9756891 ]], dtype=float32)
```

查看预测结果

合并数据，查看预测结果



```
# 在数据表最后一列插入生存概率
all_passenger_pd.insert(len(all_passenger_pd.columns),'surv_probability',surv_probability)
```

```
all_passenger_pd[-5:]
```

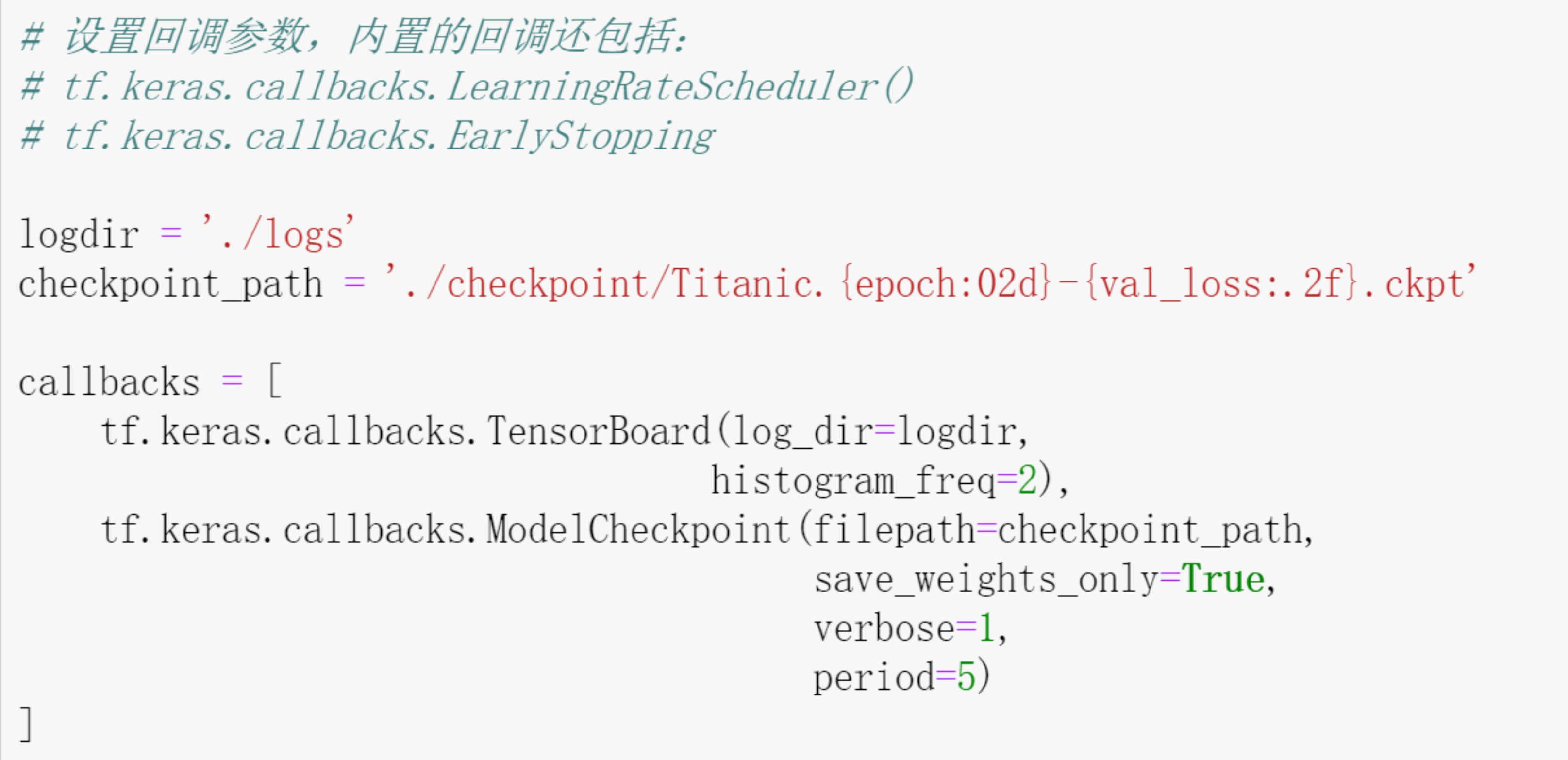
	survived	name	pclass	sex	age	sibsp	parch	fare	embarked	surv_probability
1306	0	Zakarian, Mr. Mapriededer	3	male	26.5	0	0	7.225	C	0.200157
1307	0	Zakarian, Mr. Ortin	3	male	27.0	0	0	7.225	C	0.196501
1308	0	Zimmerman, Mr. Leo	3	male	29.0	0	0	7.875	S	0.150551
0	0	Jack	3	male	23.0	1	0	5.000	S	0.147279
1	1	Rose	1	female	20.0	1	0	100.000	S	0.992087

模型训练日志记录、模型存储的实现

–

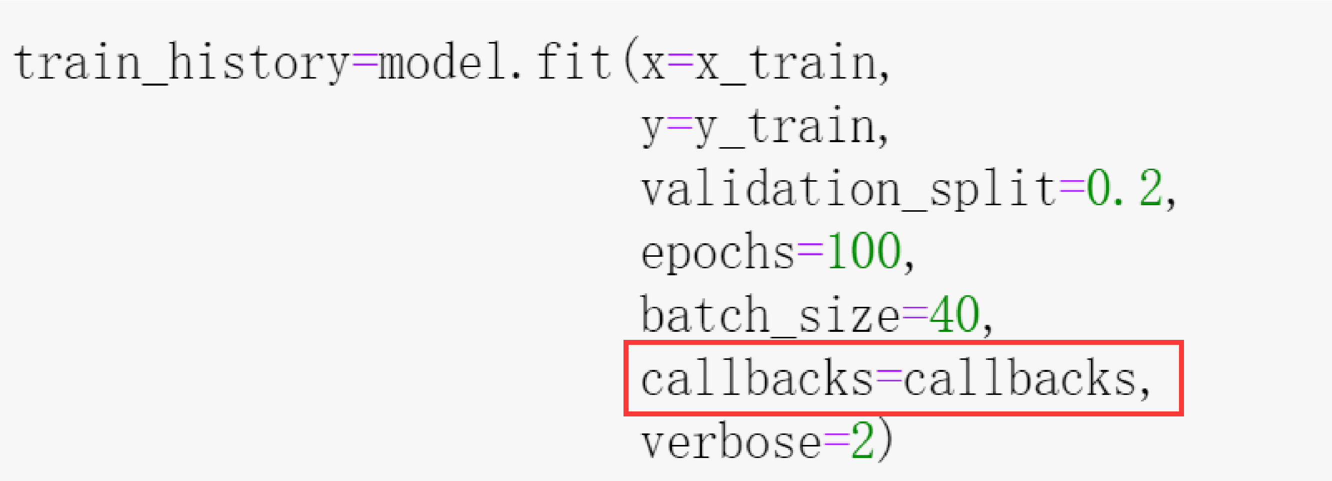
模型训练过程中的回调

模型回调参数设置



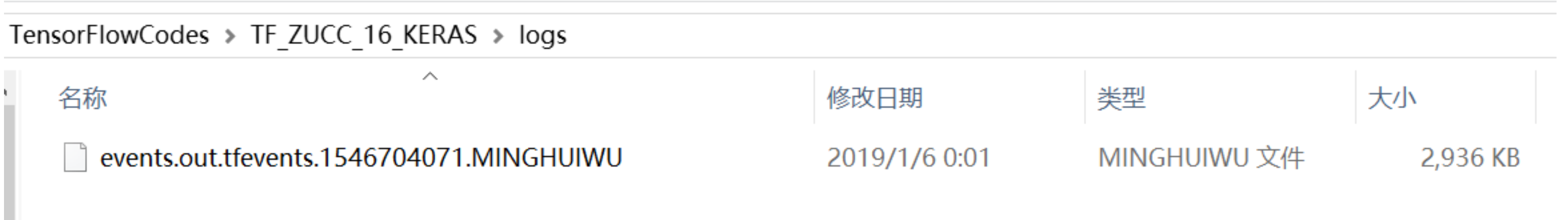
```
# 设置回调参数，内置的回调还包括:
# tf.keras.callbacks.LearningRateScheduler()
# tf.keras.callbacks.EarlyStopping

logdir = ’./logs’
checkpoint_path = ’./checkpoint/Titanic.{epoch:02d}-{val_loss:.2f}.ckpt’

callbacks = [
    tf.keras.callbacks.TensorBoard(log_dir=logdir,
                                   histogram_freq=2),
    tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                       save_weights_only=True,
                                       verbose=1,
                                       period=5)
]
```

模型训练



```
train_history=model.fit(x=x_train,
                        y=y_train,
                        validation_split=0.2,
                        epochs=100,
                        batch_size=40,
                        callbacks=callbacks,
                        verbose=2)
```

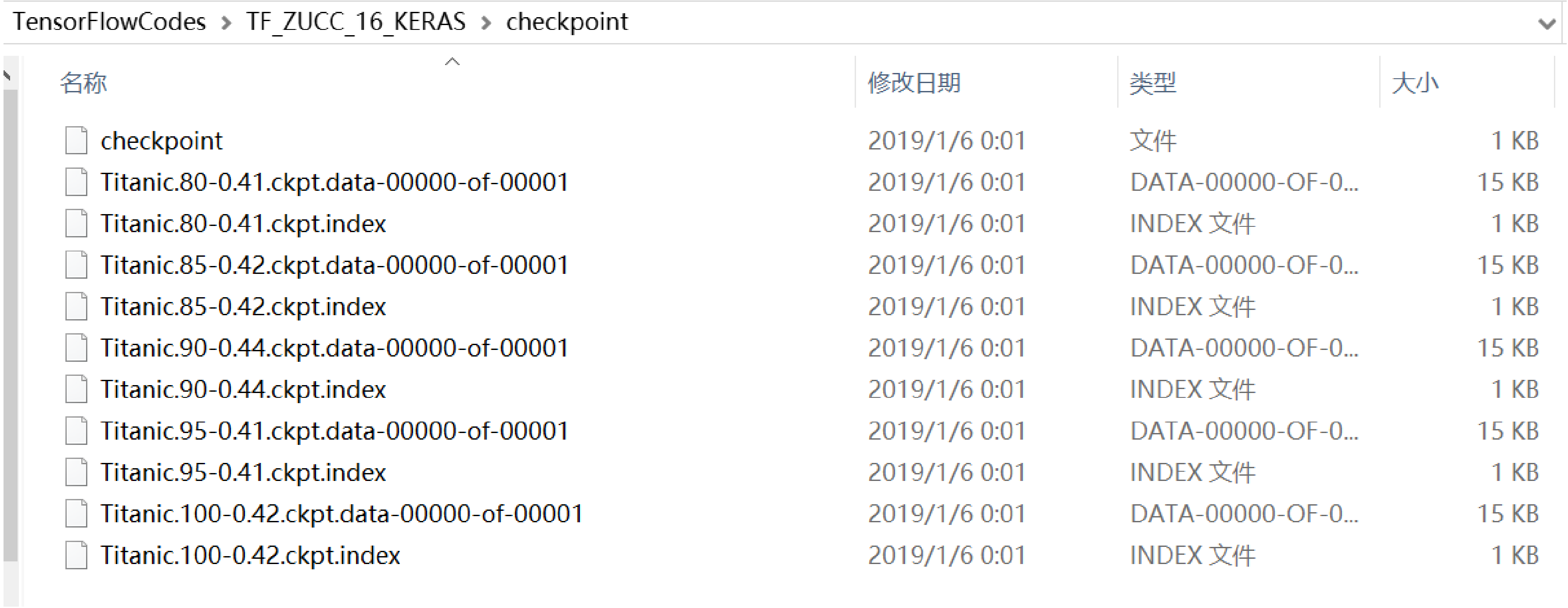
查看模型训练日志文件



TensorFlowCodes > TF_ZUCC_16_KERAS > logs

名称	修改日期	类型	大小
events.out.tfevents.1546704071.MINGHUIWU	2019/1/6 0:01	MINGHUIWU 文件	2,936 KB

查看模型存储文件



TensorFlowCodes › TF_ZUCC_16_KERAS › checkpoint

名称	修改日期	类型	大小
checkpoint	2019/1/6 0:01	文件	1 KB
Titanic.80-0.41.ckpt.data-00000-of-00001	2019/1/6 0:01	DATA-00000-OF-0...	15 KB
Titanic.80-0.41.ckpt.index	2019/1/6 0:01	INDEX 文件	1 KB
Titanic.85-0.42.ckpt.data-00000-of-00001	2019/1/6 0:01	DATA-00000-OF-0...	15 KB
Titanic.85-0.42.ckpt.index	2019/1/6 0:01	INDEX 文件	1 KB
Titanic.90-0.44.ckpt.data-00000-of-00001	2019/1/6 0:01	DATA-00000-OF-0...	15 KB
Titanic.90-0.44.ckpt.index	2019/1/6 0:01	INDEX 文件	1 KB
Titanic.95-0.41.ckpt.data-00000-of-00001	2019/1/6 0:01	DATA-00000-OF-0...	15 KB
Titanic.95-0.41.ckpt.index	2019/1/6 0:01	INDEX 文件	1 KB
Titanic.100-0.42.ckpt.data-00000-of-00001	2019/1/6 0:01	DATA-00000-OF-0...	15 KB
Titanic.100-0.42.ckpt.index	2019/1/6 0:01	INDEX 文件	1 KB