lab04

May 9, 2024

0.1 1. Load the titanic dataset in the individual job from Tensorflow Datasets, including the independent features (age, fare) and class label (embarked) specified in the job. Leave in the set features that take numeric values.

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import tensorflow_datasets as tfds
```

[2]:
```python
# loading titanic dataset
ds = tfds.load("titanic", split='train')
print(ds)
```

```
WARNING:absl:You use TensorFlow DType <dtype: 'string'> in tfds.features This
will soon be deprecated in favor of NumPy DTypes. In the meantime it was
converted to object.
WARNING:absl:You use TensorFlow DType <dtype: 'float32'> in tfds.features This
will soon be deprecated in favor of NumPy DTypes. In the meantime it was
converted to float32.
WARNING:absl:You use TensorFlow DType <dtype: 'int32'> in tfds.features This
will soon be deprecated in favor of NumPy DTypes. In the meantime it was
converted to int32.
```

```
<_PrefetchDataset element_spec={'age': TensorSpec(shape=(), dtype=tf.float32,
name=None), 'boat': TensorSpec(shape=(), dtype=tf.string, name=None), 'body':
TensorSpec(shape=(), dtype=tf.int32, name=None), 'cabin': TensorSpec(shape=(),
dtype=tf.string, name=None), 'embarked': TensorSpec(shape=(), dtype=tf.int64,
name=None), 'fare': TensorSpec(shape=(), dtype=tf.float32, name=None),
'home.dest': TensorSpec(shape=(), dtype=tf.string, name=None), 'name':
TensorSpec(shape=(), dtype=tf.string, name=None), 'parch': TensorSpec(shape=(),
dtype=tf.int32, name=None), 'pclass': TensorSpec(shape=(), dtype=tf.int64,
name=None), 'sex': TensorSpec(shape=(), dtype=tf.int64, name=None), 'sibsp':
TensorSpec(shape=(), dtype=tf.int32, name=None), 'survived':
TensorSpec(shape=(), dtype=tf.int64, name=None), 'ticket': TensorSpec(shape=(),
dtype=tf.string, name=None)}>
```

[3]:
```python
# Convert tf.data.titanic to a panda dataframe
df = tfds.as_dataframe(ds)
```

```python
df.head()
```

[3]:
```
    age        boat  body       cabin  embarked     fare  \
0  30.0  b'Unknown'    -1  b'Unknown'         2  13.0000
1  37.0  b'Unknown'    98  b'Unknown'         2   7.9250
2  28.0        b'9'    -1  b'Unknown'         2  13.0000
3  18.0  b'Unknown'    -1  b'Unknown'         2  73.5000
4  -1.0  b'Unknown'    -1  b'Unknown'         0   7.8958

                             home.dest                              name  \
0                         b'Sarnia, ON'       b'McCrie, Mr. James Matthew'
1  b'Ruotsinphytaa, Finland New York, NY'  b'Gustafsson, Mr. Anders Vilhelm'
2                            b'Spain'       b'Reynaldo, Ms. Encarnacion'
3                 b'Lyndhurst, England'       b'Davies, Mr. Charles Henry'
4                          b'Unknown'          b'Gheorgheff, Mr. Stanio'

   parch  pclass  sex  sibsp  survived           ticket
0      0       1    0      0         0        b'233478'
1      0       2    0      2         0       b'3101276'
2      0       1    1      0         1        b'230434'
3      0       1    0      0         0  b'S.O.C. 14879'
4      0       2    0      0         0        b'349254'
```

[4]:
```python
#Let's drop columns which don't take numeric values
df = df.drop(columns=['boat', 'cabin', 'home.dest', 'name', 'ticket'])
df.head()
```

[4]:
```
    age  body  embarked     fare  parch  pclass  sex  sibsp  survived
0  30.0    -1         2  13.0000      0       1    0      0         0
1  37.0    98         2   7.9250      0       2    0      2         0
2  28.0    -1         2  13.0000      0       1    1      0         1
3  18.0    -1         2  73.5000      0       1    0      0         0
4  -1.0    -1         0   7.8958      0       2    0      0         0
```

[5]:
```python
#Let's see which columns have NaN values
# count NaN values in each column
print(df.isnull().sum())
```

```
age         0
body        0
embarked    0
fare        0
parch       0
pclass      0
sex         0
sibsp       0
survived    0
dtype: int64
```

0.2 2. Visualize points in a dataset on a plane with coordinates corresponding to two independent features, displaying points of different classes in different colors. Label the axes and figure, and create a legend for the dataset classes.

[6]:
```
# Port of Embarkation is unknown
df = df[df['embarked'] != 3]
```

[7]:
```
# age is -1
df['age'].value_counts()
```

[7]:
```
age
-1.0000    263
 24.0000     47
 22.0000     43
 21.0000     41
 30.0000     40
           …
 24.5000      1
 80.0000      1
 0.3333       1
 23.5000      1
 55.5000      1
Name: count, Length: 99, dtype: int64
```

[8]:
```
#263 age entries are missing we need to impute correctly
#Replace -1 with NaN
df['age'].replace(-1, np.nan, inplace=True)

# Fill missing age values with the mean age grouped by 'pclass' and 'sex'
df['age'].fillna(df.groupby(['pclass', 'sex'])['age'].transform('mean'),␣
 ↪inplace=True)
```

```
C:\Users\Mo\AppData\Local\Temp\ipykernel_7268\746953612.py:3: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['age'].replace(-1, np.nan, inplace=True)
C:\Users\Mo\AppData\Local\Temp\ipykernel_7268\746953612.py:6: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
```

```
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['age'].fillna(df.groupby(['pclass', 'sex'])['age'].transform('mean'),
inplace=True)
```

[9]:

```
# age is -1
df['age'].value_counts()
```

[9]:

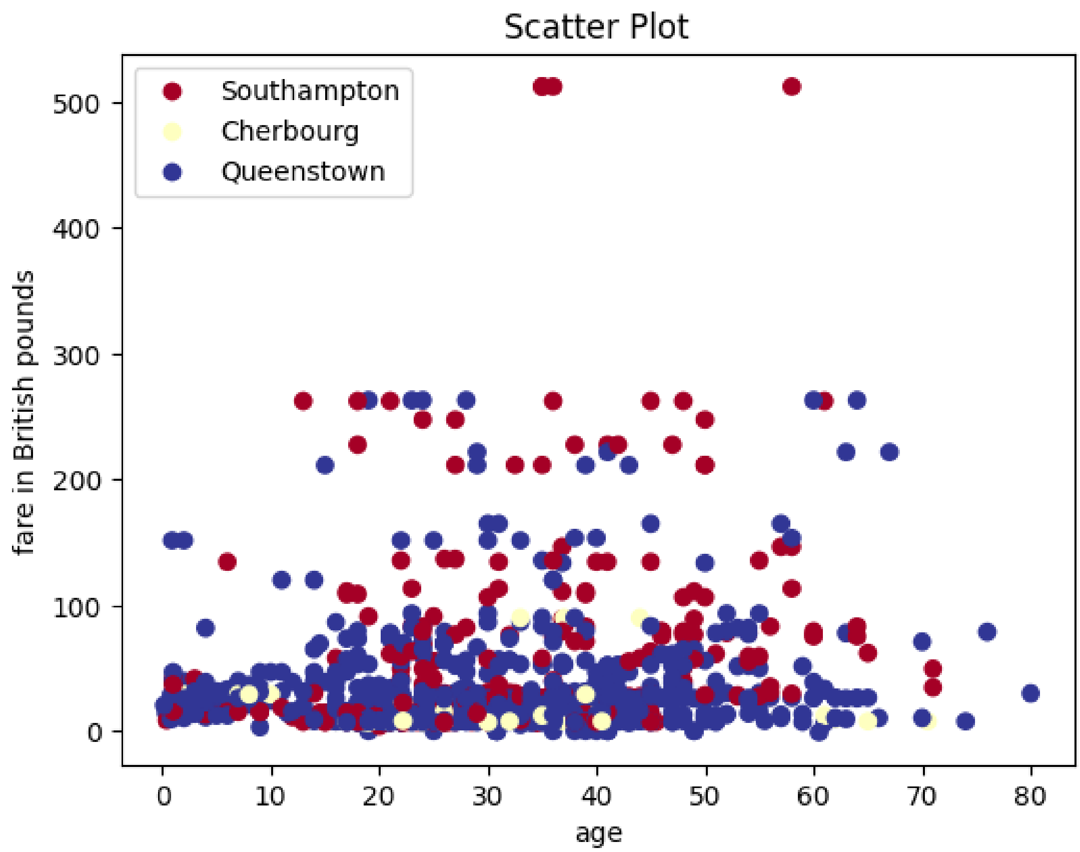
```
age
25.962273    144
22.185308     64
24.000000     47
22.000000     43
21.000000     41
             ...
80.000000      1
0.666700       1
74.000000      1
26.500000      1
55.500000      1
Name: count, Length: 104, dtype: int64
```

[10]:

```
df.info()
```

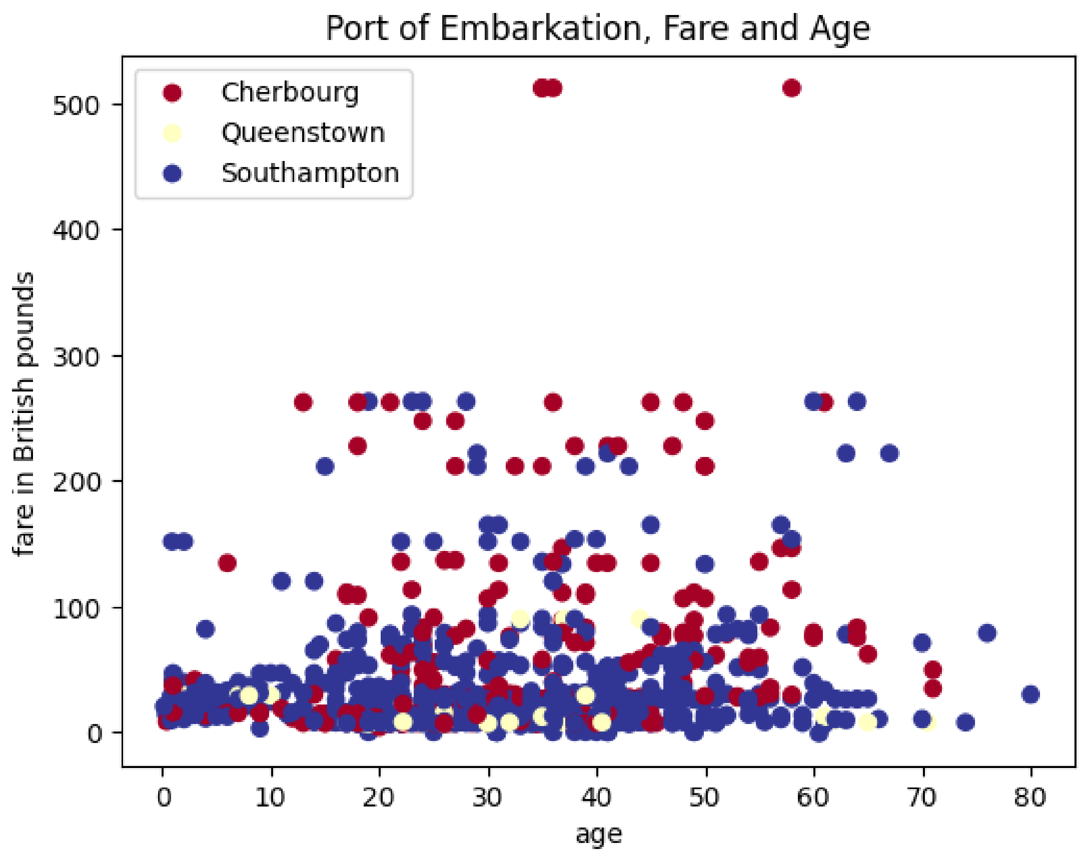
```
<class 'pandas.core.frame.DataFrame'>
Index: 1307 entries, 0 to 1308
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1307 non-null   float32
 1   body      1307 non-null   int32
 2   embarked  1307 non-null   int64
 3   fare      1307 non-null   float32
 4   parch     1307 non-null   int32
 5   pclass    1307 non-null   int64
 6   sex       1307 non-null   int64
 7   sibsp     1307 non-null   int32
 8   survived  1307 non-null   int64
dtypes: float32(2), int32(3), int64(4)
memory usage: 76.6 KB
```

[11]:
```
embarked_labels = {
    0: 'Southampton',
    1: 'Cherbourg',
    2: 'Queenstown',
    3: 'Unknown'
}
plt.xlabel('age')  #
plt.ylabel('fare in British pounds')  #
plt.title('Scatter Plot')
scatter = plt.scatter(df['age'], df['fare'], c = df['embarked'], cmap = plt.cm.
 ↪RdYlBu, label=[embarked_labels[i] for i in df['embarked']])
plt.legend(handles=scatter.legend_elements()[0], labels=[embarked_labels[i] for␣
 ↪i in range(4)], loc='upper left')
plt.show()
```



[12]:
```
embarked_labels = {
    0: 'Cherbourg',
    1: 'Queenstown',
    2: 'Southampton',
```

```
    3: 'Unknown'
}
plt.xlabel('age')  #
plt.ylabel('fare in British pounds')  #
plt.title('Port of Embarkation, Fare and Age')
scatter = plt.scatter(df['age'], df['fare'], c = df['embarked'], cmap = plt.cm.
 ↪RdYlBu, label=[embarked_labels[i] for i in df['embarked']])
plt.legend(handles=scatter.legend_elements()[0], labels=[embarked_labels[i] for␣
 ↪i in range(4)], loc='upper left')
plt.show()
```



0.3 3. If a feature with class labels contains more than two classes, then combine some of the classes to obtain a set for binary classification. Combine classes so that the positive and negative classes are comparable in number of points.¶

[13]:
```
df['embarked'].value_counts()
```

[13]:
```
embarked
2    914
0    270
1    123
Name: count, dtype: int64
```

[14]:
```
# Combine classes 0 (Cherbourg) and 1 (Queenstown) into one class
df.loc[df['embarked'].isin([0, 1]), 'embarked'] = 0
```

[16]:
```
# Let's rename class 2 into class 1
# Replace class 2 with 1
df['embarked'] = df['embarked'].replace(2, 1)
```

[17]:
```
df['embarked'].value_counts()
```

[17]:
```
embarked
1    914
0    393
Name: count, dtype: int64
```

0.4 4. Split the data set of two features and binary class labels into training and test sets. Build neural networks with a normalizing layer and the parameters specified in the individual task for binary classification and train them on the training set, controlling the learning process of the neural networks. Determine the neural network with higher binary classification quality based on the binary classification score specified in the individual assignment.

0.5 5. Visualize the decision boundaries of the constructed neural networks in separate figures on the entire data set of two features and binary class labels.

[18]:
```
from sklearn.model_selection import train_test_split
```

[19]:
```
X_train, X_test, y_train, y_test = train_test_split(df[['age', 'fare']],␣
↪df['embarked'], test_size=0.3, random_state=42)
```

[20]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[20]:
```
((914, 2), (393, 2), (914,), (393,))
```

Deep neural network for binary classification problem Since the features of the set have different ranges of change, we use a normalization layer adapted to all independent features:

[30]:
```
# Normalizlation layer:
feature_normalizer = tf.keras.layers.Normalization(axis=None,␣
↪input_shape=(X_train.shape[1],))
feature_normalizer.adapt(X_train.to_numpy())
```

Let's create a neural network with three hidden dense layers with 128 neurons with elu, sigmoid, tanh activation functions, and an output layer of one neuron with a sigmoid activation function:

[31]:
```
model = tf.keras.Sequential([
    feature_normalizer,
    tf.keras.layers.Dense(128, activation='elu', input_shape=()),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

```
C:\Users\Mo\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kf
ra8p0\LocalCache\local-packages\Python311\site-
packages\keras\src\layers\core\dense.py:86: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
normalization_5 (Normalization)	(None, 2)	3
dense_4 (Dense)	(None, 128)	384
dense_5 (Dense)	(None, 128)	16,512
dense_6 (Dense)	(None, 128)	16,512
dense_7 (Dense)	(None, 1)	129

Total params: 33,540 (131.02 KB)

Trainable params: 33,537 (131.00 KB)

Non-trainable params: 3 (16.00 B)

binary_crossentropy.
(accuracy):

[32]:
```
model.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.AdamW(learning_rate=0.01),
    metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')]
)
```

[33]:
```
history = model.fit(X_train, y_train, epochs=3)
```

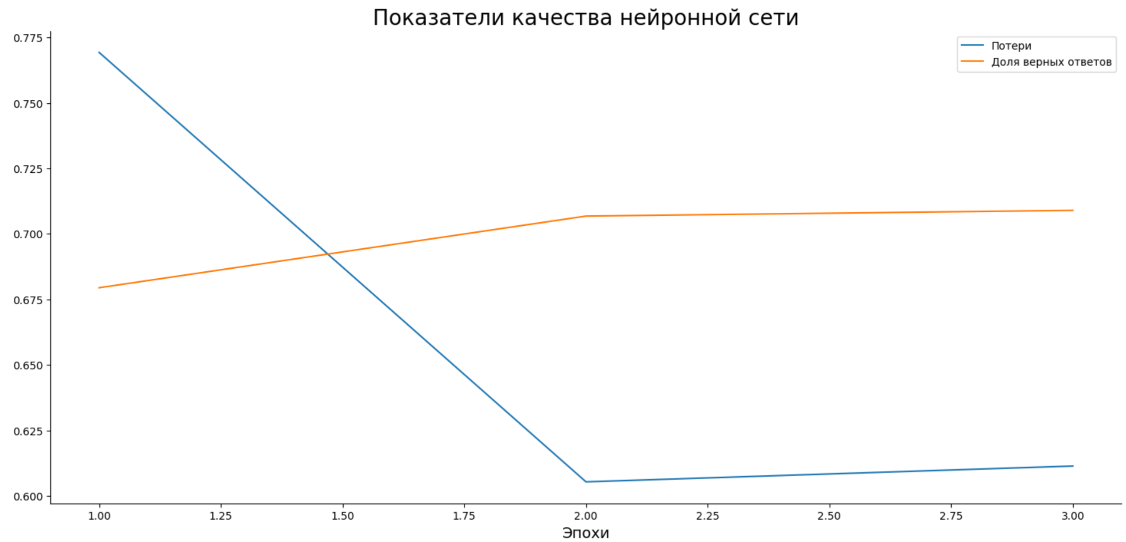
```
Epoch 1/3
29/29              1s 750us/step -
accuracy: 0.6738 - loss: 0.9487
Epoch 2/3
29/29              0s 607us/step -
accuracy: 0.7080 - loss: 0.6134
Epoch 3/3
29/29              0s 590us/step -
accuracy: 0.7214 - loss: 0.5968
```

0.5.1 Visualization of model training

[35]:
```
from matplotlib import rcParams

rcParams['figure.figsize'] = (18, 8)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
```

[36]:
```
plt.plot(np.arange(1, 4), history.history['loss'], label='    ')
plt.plot(np.arange(1, 4), history.history['accuracy'], label='           ')
plt.title('                   ', size=20)
plt.xlabel('   ', size=14)
plt.legend();
```



0.5.2 Model prediction

Using a trained neural network, we obtain output values that can be interpreted as probabilities:

[37]:
```
prediction = model.predict(X_test)
prediction
```

```
13/13              0s 3ms/step
```

[37]:
```
array([[0.56305224],
       [0.62253666],
       [0.31531483],
       [0.6231352 ],
       [0.6185719 ],
       [0.61565423],
       [0.6223922 ],
       [0.6231463 ],
       [0.6190608 ],
       [0.61494374],
       [0.6224157 ],
       [0.6182091 ],
       [0.6092111 ],
       [0.42504388],
       [0.6222489 ],
       [0.6234504 ],
       [0.6194412 ],
       [0.6155466 ],
       [0.5371984 ],
       [0.41345227],
       [0.5884235 ],
```

```
[0.62365246],
[0.62188095],
[0.6152756 ],
[0.62324595],
[0.6215776 ],
[0.30854163],
[0.6233369 ],
[0.5505328 ],
[0.62242556],
[0.48188126],
[0.6235304 ],
[0.6169645 ],
[0.41855913],
[0.5371984 ],
[0.62348515],
[0.6230541 ],
[0.62152326],
[0.31203115],
[0.6237026 ],
[0.33484817],
[0.31279063],
[0.6186439 ],
[0.6235425 ],
[0.30754274],
[0.62236   ],
[0.62384295],
[0.62129235],
[0.623364  ],
[0.6230787 ],
[0.61998713],
[0.6235782 ],
[0.4329073 ],
[0.47607824],
[0.40437704],
[0.31055233],
[0.62354606],
[0.30836937],
[0.5605323 ],
[0.3167587 ],
[0.4916468 ],
[0.6231204 ],
[0.623464  ],
[0.6226061 ],
[0.6239263 ],
[0.61366737],
[0.3506364 ],
[0.62083626],
```

```
[0.6241932 ],
[0.623788  ],
[0.6226061 ],
[0.62340623],
[0.6234504 ],
[0.6232311 ],
[0.6224934 ],
[0.6236886 ],
[0.62305135],
[0.62055385],
[0.6217409 ],
[0.30393457],
[0.62397766],
[0.60542345],
[0.622893  ],
[0.6206666 ],
[0.62327886],
[0.56016684],
[0.5654798 ],
[0.6170194 ],
[0.6234249 ],
[0.59283036],
[0.6215855 ],
[0.5473324 ],
[0.6232203 ],
[0.62430406],
[0.62411034],
[0.62265444],
[0.62322164],
[0.5322448 ],
[0.57894915],
[0.6236086 ],
[0.6230608 ],
[0.62376714],
[0.6239624 ],
[0.6130197 ],
[0.6200319 ],
[0.5955006 ],
[0.62344134],
[0.6230496 ],
[0.62386656],
[0.3539107 ],
[0.58545303],
[0.6059326 ],
[0.6183897 ],
[0.4380337 ],
[0.5841717 ],
```

```
[0.59922427],
[0.38137716],
[0.62349045],
[0.6237403 ],
[0.62171495],
[0.6222403 ],
[0.62304664],
[0.62235785],
[0.62225544],
[0.6237757 ],
[0.4111939 ],
[0.61649   ],
[0.62361944],
[0.5519779 ],
[0.314443  ],
[0.6189085 ],
[0.61508954],
[0.5948527 ],
[0.6200771 ],
[0.52495015],
[0.6230541 ],
[0.6234664 ],
[0.6233754 ],
[0.62434924],
[0.48716575],
[0.6236154 ],
[0.6234504 ],
[0.6234277 ],
[0.6230381 ],
[0.6148167 ],
[0.62306887],
[0.6234504 ],
[0.62040794],
[0.61745024],
[0.6237713 ],
[0.6120924 ],
[0.62284327],
[0.6237514 ],
[0.6234424 ],
[0.6230541 ],
[0.62357295],
[0.60070163],
[0.6234484 ],
[0.6184776 ],
[0.6221728 ],
[0.42215514],
[0.3123741 ],
```

```
[0.6234293 ],
[0.62217855],
[0.5095346 ],
[0.46205226],
[0.6243398 ],
[0.37740296],
[0.62040794],
[0.6205773 ],
[0.44454554],
[0.62319165],
[0.41791856],
[0.5707644 ],
[0.62317264],
[0.62344134],
[0.6239414 ],
[0.6158035 ],
[0.3806427 ],
[0.6210679 ],
[0.61998713],
[0.62386656],
[0.6237834 ],
[0.6217524 ],
[0.31316596],
[0.5013102 ],
[0.3114098 ],
[0.61992145],
[0.6185188 ],
[0.6126151 ],
[0.61771244],
[0.6123527 ],
[0.6233518 ],
[0.6180385 ],
[0.6205338 ],
[0.58901703],
[0.6114726 ],
[0.6231121 ],
[0.62156403],
[0.6040692 ],
[0.6192478 ],
[0.31523016],
[0.4868322 ],
[0.61623794],
[0.6226061 ],
[0.6231204 ],
[0.6190373 ],
[0.62348545],
[0.6230381 ],
```

```
[0.6218926 ],
[0.62335074],
[0.6230381 ],
[0.42458707],
[0.61549723],
[0.6214963 ],
[0.6222167 ],
[0.5803593 ],
[0.4220822 ],
[0.31640872],
[0.56886494],
[0.62283665],
[0.6237607 ],
[0.6198125 ],
[0.6222688 ],
[0.4493296 ],
[0.62305576],
[0.62153137],
[0.6164312 ],
[0.62306887],
[0.62321216],
[0.48509312],
[0.44826117],
[0.59802043],
[0.62185407],
[0.6232555 ],
[0.62293124],
[0.623049  ],
[0.38415268],
[0.6166916 ],
[0.62331235],
[0.62312025],
[0.621667  ],
[0.5409555 ],
[0.6198337 ],
[0.6234564 ],
[0.62306195],
[0.6243755 ],
[0.62287194],
[0.6234504 ],
[0.58959645],
[0.554698  ],
[0.60295445],
[0.62345153],
[0.5717027 ],
[0.5961038 ],
[0.6230541 ],
```

```
[0.62349916],
[0.5371984 ],
[0.6164312 ],
[0.3677414 ],
[0.6234504 ],
[0.62193567],
[0.62312   ],
[0.56066406],
[0.6233921 ],
[0.62366116],
[0.62353814],
[0.6238154 ],
[0.6229099 ],
[0.6224462 ],
[0.62306887],
[0.6230541 ],
[0.50513494],
[0.6237366 ],
[0.62243867],
[0.6198261 ],
[0.62312   ],
[0.6223795 ],
[0.6217215 ],
[0.6232722 ],
[0.62255836],
[0.6230381 ],
[0.620783  ],
[0.62269384],
[0.6238215 ],
[0.6238332 ],
[0.33004537],
[0.6135328 ],
[0.6190243 ],
[0.62344486],
[0.6237597 ],
[0.62122095],
[0.6242861 ],
[0.6231606 ],
[0.61893576],
[0.61712027],
[0.62349916],
[0.6185794 ],
[0.57853246],
[0.6101181 ],
[0.6230889 ],
[0.40019605],
[0.30927575],
```

```
[0.6207469 ],
[0.33229455],
[0.6226061 ],
[0.62400556],
[0.62353754],
[0.6182803 ],
[0.62361556],
[0.62164617],
[0.61625135],
[0.6231204 ],
[0.5371984 ],
[0.62208897],
[0.52688336],
[0.6091795 ],
[0.3141163 ],
[0.55330694],
[0.6230889 ],
[0.6229998 ],
[0.6033912 ],
[0.31470603],
[0.30374488],
[0.62040794],
[0.6031983 ],
[0.62368524],
[0.6164312 ],
[0.6226713 ],
[0.550768  ],
[0.61689353],
[0.622646  ],
[0.6237819 ],
[0.62307096],
[0.6230381 ],
[0.4526002 ],
[0.6232809 ],
[0.60205674],
[0.62311804],
[0.6070237 ],
[0.4237171 ],
[0.6219753 ],
[0.6222711 ],
[0.6235595 ],
[0.6237597 ],
[0.47023177],
[0.62338424],
[0.62279963],
[0.62152326],
[0.6231204 ],
```

```
[0.33383062],
[0.6230539 ],
[0.6243658 ],
[0.6223552 ],
[0.48792887],
[0.57108235],
[0.6160902 ],
[0.6232632 ],
[0.6197219 ],
[0.6231924 ],
[0.58818954],
[0.49523336],
[0.6198174 ],
[0.5410435 ],
[0.3443243 ],
[0.623137  ],
[0.61874396],
[0.6114783 ],
[0.5989466 ],
[0.6221531 ],
[0.61023927],
[0.6231352 ],
[0.6223674 ],
[0.6235473 ],
[0.62153137],
[0.3884406 ],
[0.6213188 ],
[0.622005  ],
[0.3567975 ],
[0.6175716 ],
[0.62361944],
[0.62284327],
[0.62186754],
[0.6197219 ],
[0.34012643],
[0.623979  ],
[0.6199609 ],
[0.6210896 ],
[0.6234424 ],
[0.5962812 ],
[0.6047051 ],
[0.62045586],
[0.62421393]], dtype=float32)
```

These probabilities can be converted into predicted classes as follows (a threshold of 0.5 was used):

[38]:
```python
y_pred = np.array([1 if prob > 0.5 else 0 for prob in np.ravel(prediction)])
print(y_pred)
```

```
[1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1
 1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1
 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 0
 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1
 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1
 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1]
```

The model evaluation on the test sample looks like this:

[39]:
```python
loss, accuracy = model.evaluate(X_test, y_test)
loss, accuracy
```

```
13/13             0s 542us/step -
accuracy: 0.7079 - loss: 0.6270
```

[39]: (0.626963198184967, 0.7124682068824768)

0.5.3 Visualization of the decision boundary

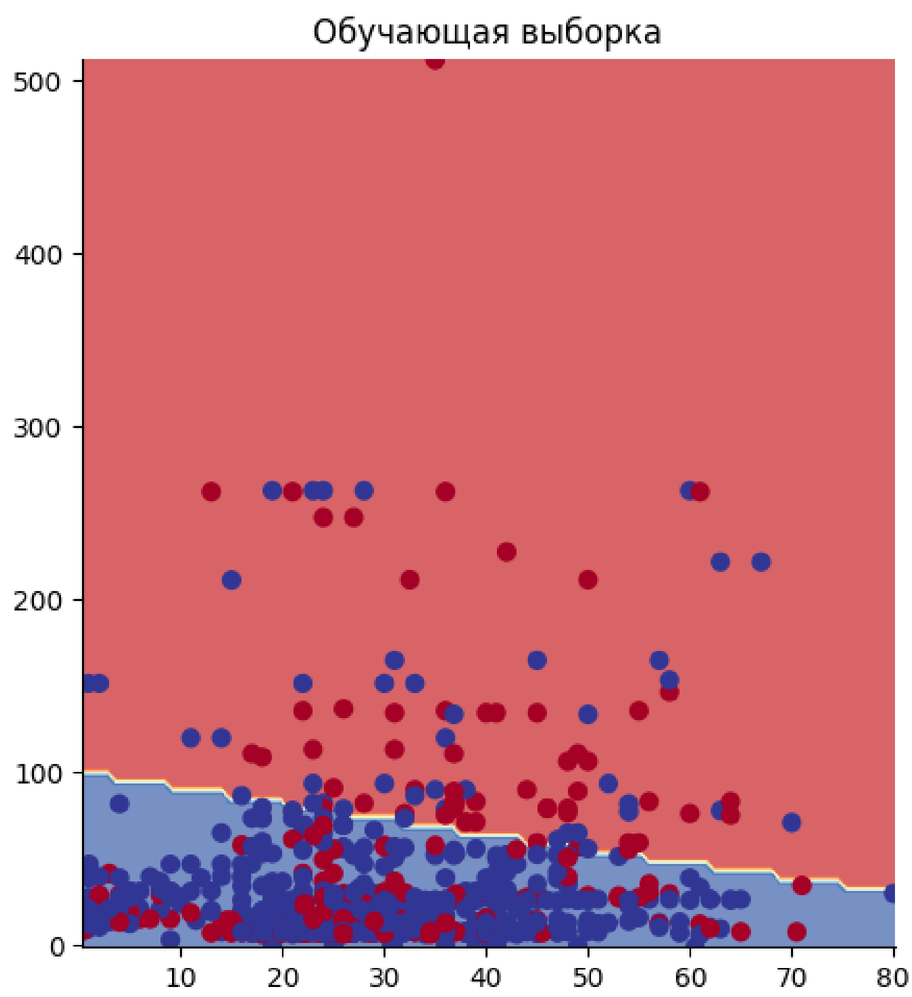
The decision boundary for the constructed classifier:

[46]:
```python
def plot_decision_boundary(model, X, y):
    #
    x_min, x_max = X.iloc[:, 0].min() - 0.1, X.iloc[:, 0].max() + 0.1
    y_min, y_max = X.iloc[:, 1].min() - 0.1, X.iloc[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100))
    #
    X_in = np.c_[xx.ravel(), yy.ravel()]
    #
    y_pred = model.predict(X_in)
    #
    if len(y_pred.shape) > 1 and y_pred.shape[1] > 1:
        #
        #
        y_pred = np.argmax(y_pred, axis=1).reshape(xx.shape)
    else:
        #
        y_pred = np.round(y_pred).reshape(xx.shape)
    #
    plt.contourf(xx, yy, y_pred, cmap=plt.cm.RdYlBu, alpha=0.7)
```

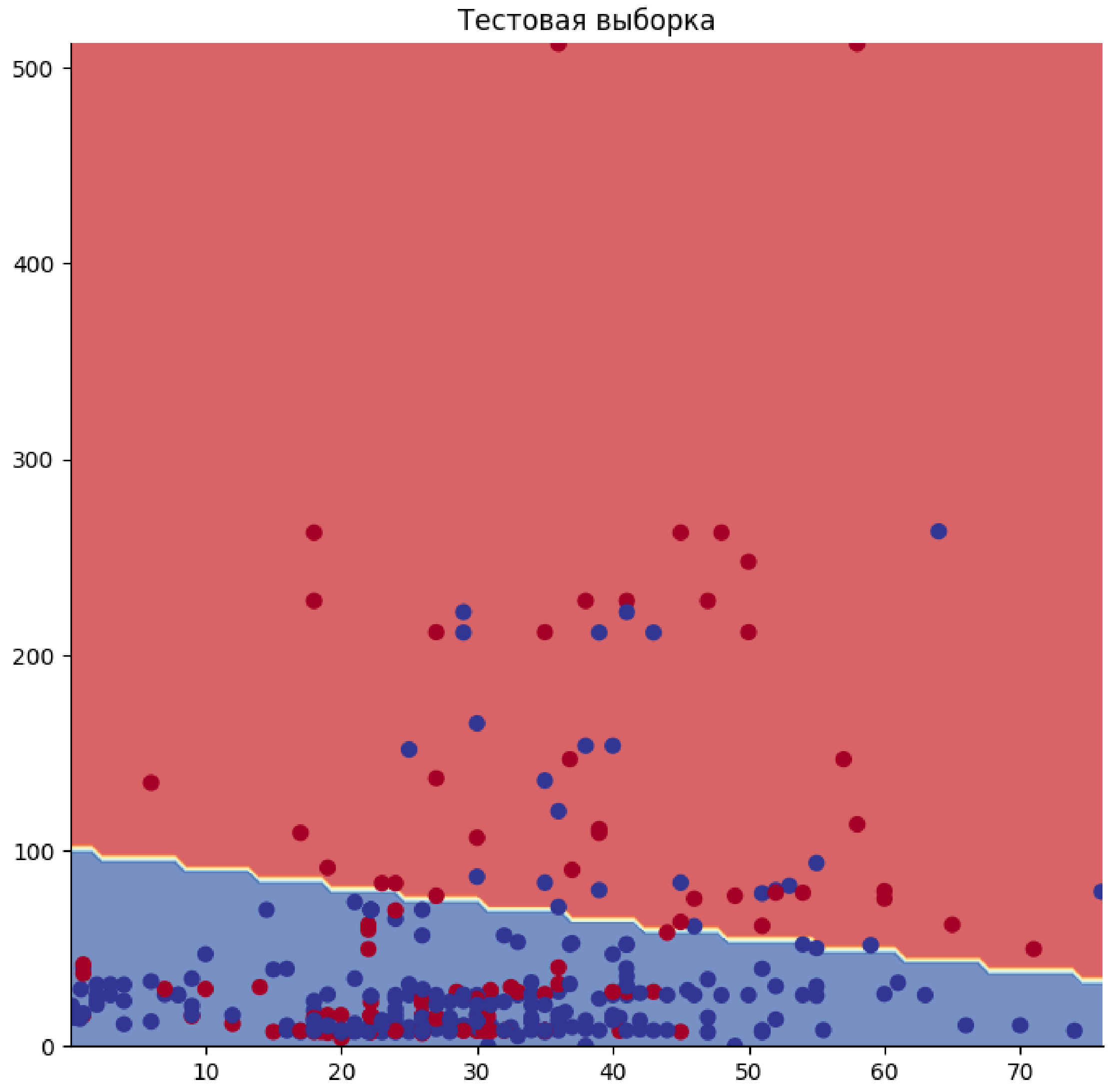
```
    plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, s=40, cmap=plt.cm.RdYlBu)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.show()
```

[47]:
```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("          ")
plot_decision_boundary(model, X_train, y_train)
plt.subplot(1, 2, 2)
plt.title("         ")
plot_decision_boundary(model, X_test, y_test)
```

```
313/313                   0s 398us/step
```




```
313/313                   0s 390us/step
```



0.5.4 Quality indicators for binary classification

When there are only two classes, we will call class c_1 the positive class and class c_2 the negative class. Then the confusion matrix takes the form:

$$\begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix},$$

Where * TP – (True Positives) – number of correctly predicted points in class c_1 * FN – (False Negatives) – number of points in class c_1, incorrectly predicted into class c_2 * FP – (False Positives) – number of points in class c_2, incorrectly predicted to class c_1 * TN – (True Negatives) – number of correctly predicted points in class c_2

[48]:
```
def TN(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 0) & (y_predict == 0))
```

[49]:
```
def FP(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 0) & (y_predict == 1))
```

[50]:
```
def FN(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 1) & (y_predict == 0))
```

[51]:
```
def TP(y_true, y_predict):
    assert len(y_true) == len(y_predict)
    return np.sum((y_true == 1) & (y_predict == 1))
```

The confusion matrix for binary classification is defined as follows:

[52]:
```
def confusion_matrix(y_true, y_predict):
    return np.array([
        [TP(y_true, y_predict), FN(y_true, y_predict)],
        [FP(y_true, y_predict), TN(y_true, y_predict)]
    ])
```

[53]:
```
confusion_matrix(y_test, y_pred)
```

[53]:
```
array([[243,  24],
       [ 89,  37]], dtype=int64)
```

[54]:
```
def tpr_score(y_true, y_predict):
    tp = TP(y_true, y_predict)
    fn = FN(y_true, y_predict)
    try:
        return tp / (tp + fn)
    except:
        return 0.0

def fpr_score(y_true, y_predict):
    fp = FP(y_true, y_predict)
    tn = TN(y_true, y_predict)
    try:
        return fp / (fp + tn)
    except:
        return 0.0
```

[55]:
```
tpr_score(y_test, y_pred), fpr_score(y_test, y_pred)
```

[55]:
```
(0.9101123595505618, 0.7063492063492064)
```

(False Negatives) – the number of points in the positive class that were incorrectly predicted into the negative class

[57]:
```
FN(y_test, y_pred)
```

[57]: 24

0.6 6. Visualize ROC curves for the constructed classifiers based on neural networks in one figure, calculate the areas under the ROC curves using the trapezoidal method or another method, and create a legend indicating the areas of the curves.

0.6.1 ROC analysis

ROC (Receiver Operating Characteristic) analysis is a popular strategy for evaluating the performance of binary classifiers. For ROC analysis, you need not only a prediction of class labels, but also the values of the so-called. scoring function for each point in the test set. As the values of the scoring function, we can take the probabilities returned by the neural network:

[61]:
```
df.head()
```

[61]:

	age	body	embarked	fare	parch	pclass	sex	sibsp	survived
0	30.000000	-1	1	13.0000	0	1	0	0	0
1	37.000000	98	1	7.9250	0	2	0	2	0
2	28.000000	-1	1	13.0000	0	1	1	0	1
3	18.000000	-1	1	73.5000	0	1	0	0	0
4	25.962273	-1	0	7.8958	0	2	0	0	0

[62]:
```
X = np.array(df.drop('embarked', axis=1))
y = np.array(df['embarked'])
```

[58]:
```
def true_false_positive(threshold_vector, y_test):
    true_positive = np.equal(threshold_vector, 1) & np.equal(y_test, 1)
    true_negative = np.equal(threshold_vector, 0) & np.equal(y_test, 0)
    false_positive = np.equal(threshold_vector, 1) & np.equal(y_test, 0)
    false_negative = np.equal(threshold_vector, 0) & np.equal(y_test, 1)

    tpr = true_positive.sum() / (true_positive.sum() + false_negative.sum())
    fpr = false_positive.sum() / (false_positive.sum() + true_negative.sum())

    return tpr, fpr
```

[59]:
```
def roc_from_scratch(probabilities, y_test, partitions=100):
    roc = np.array([])
    for i in range(partitions + 1):

        threshold_vector = np.greater_equal(probabilities, i / partitions).
↪astype(int)
        tpr, fpr = true_false_positive(threshold_vector, y_test)
```

```
        roc = np.append(roc, [fpr, tpr])

    return roc.reshape(-1, 2)
```

[70]:

```
prediction = model.predict(X_test)
prediction.shape
```

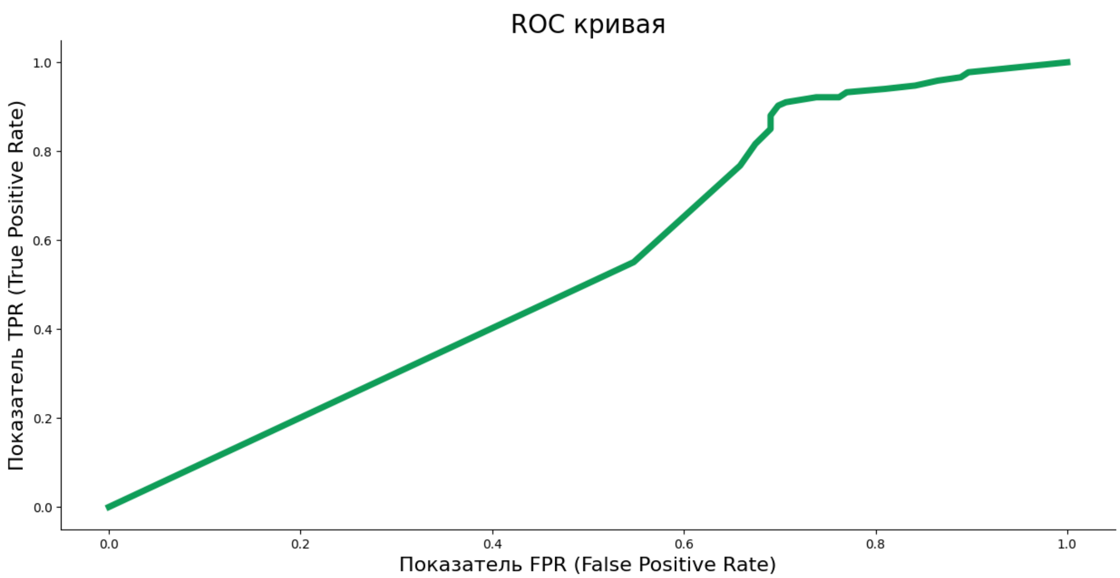
```
13/13                 0s 544us/step
```

[70]: (393, 1)

[72]:

```
plt.figure(figsize=(15,7))

ROC = roc_from_scratch(prediction.reshape(-1),y_test,partitions=50)
#plt.scatter(ROC[:,0],ROC[:,1],color='#0F9D58',s=100)
plt.plot(ROC[:,0],ROC[:,1],color='#0F9D58',lw=5)
plt.title('ROC    ',fontsize=20)
plt.xlabel('      FPR (False Positive Rate)',fontsize=16)
plt.ylabel('      TPR (True Positive Rate)',fontsize=16);
```



[73]:

```
def auc_trapezoidal(roc_curve):
    # Sort ROC curve by increasing FPR
    sorted_roc_curve = roc_curve[np.argsort(roc_curve[:, 0])]

    # Initialize area under the curve
    auc = 0.0

    # Iterate through sorted ROC curve points
```

```
    for i in range(1, len(sorted_roc_curve)):
        # Calculate trapezoidal area between consecutive points
        prev_fpr, prev_tpr = sorted_roc_curve[i - 1]
        curr_fpr, curr_tpr = sorted_roc_curve[i]
        auc += (curr_fpr - prev_fpr) * (curr_tpr + prev_tpr) / 2.0

    return auc
```

[76]:
```
# Calculate AUC and it is very low
auc = auc_trapezoidal(ROC)
auc
```

[76]: 0.5448843707270673

0.7 7. Define an additional feature in the original data set, different from the two independent features specified in the task, taking continuous values and having maximum variance.

[80]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1307 entries, 0 to 1308
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1307 non-null   float32
 1   body      1307 non-null   int32
 2   embarked  1307 non-null   int64
 3   fare      1307 non-null   float32
 4   parch     1307 non-null   int32
 5   pclass    1307 non-null   int64
 6   sex       1307 non-null   int64
 7   sibsp     1307 non-null   int32
 8   survived  1307 non-null   int64
dtypes: float32(2), int32(3), int64(4)
memory usage: 76.6 KB
```

[83]:
```
# Drop independant features
features = df.drop(["age", "fare"], axis=1)

# calcualte var
variances = features.var()

# find max var
max_variance_feature = variances.idxmax()

print("                    (       'age'   'fare'): ", max_variance_feature)
```

('age' 'fare'): body

0.8 8. Visualize data points in 3D space with coordinates corresponding to three independent features, displaying points of different classes in different colors. Label the axes and figure, and create a legend for the dataset classes.

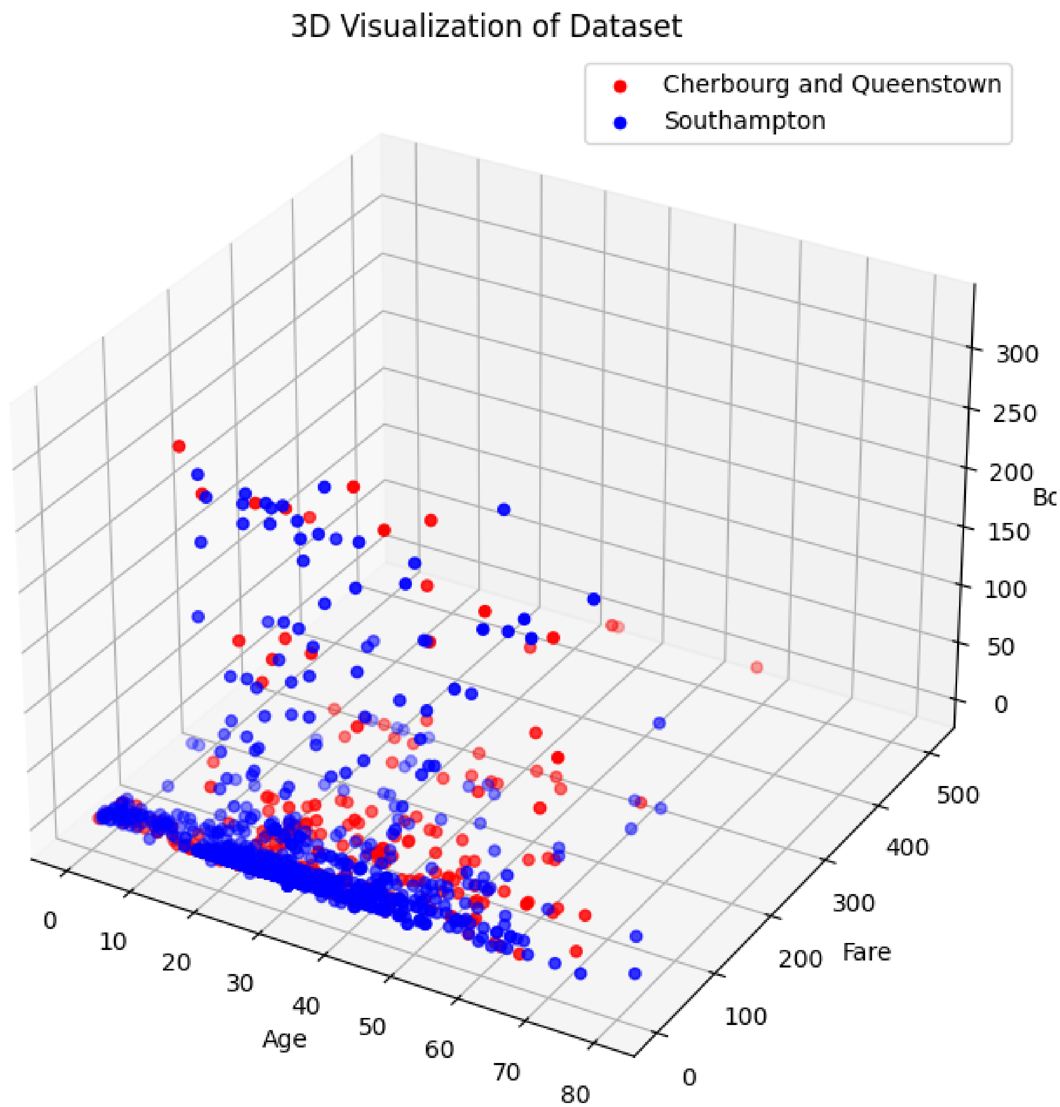
[86]:
```
from mpl_toolkits.mplot3d import Axes3D

# Create 3d graph
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(111, projection='3d')

# Divide into two classes
class_0 = df[df['embarked'] == 0] ## Cherbourg and Queenstown
class_1 = df[df['embarked'] == 1] ## Southampton

# Points of each class
ax.scatter(class_0['age'], class_0['fare'], class_0['body'], c='r',␣
 ↪label='Cherbourg and Queenstown')
ax.scatter(class_1['age'], class_1['fare'], class_1['body'], c='b',␣
 ↪label='Southampton')

# naming axis
ax.set_xlabel('Age')
ax.set_ylabel('Fare')
ax.set_zlabel('Body')
plt.title('3D Visualization of Dataset')
plt.legend()
plt.show()
```



0.9 9. Split the original data set into training and test sets. Build a neural network for multi-class classification with a normalizing layer and parameters corresponding to the best neural network for binary classification from step 4, and train it on the training set, controlling the process of its training.

[87]:
```
X = np.array(df.drop('embarked', axis=1))
y = np.array(df['embarked'])
```

[88]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[90]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[90]: ((914, 8), (393, 8), (914,), (393,))

[91]:
```
feature_normalizer = tf.keras.layers.Normalization(axis=None,input_shape=(X.
↪shape[1],))
feature_normalizer.adapt(X)
```

```
C:\Users\Mo\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kf
ra8p0\LocalCache\local-packages\Python311\site-
packages\keras\src\layers\preprocessing\normalization.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(**kwargs)
```

[93]:
```
model = tf.keras.Sequential([
    feature_normalizer,
    tf.keras.layers.Dense(128, activation='elu'),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='tanh'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.summary()
```

Model: "sequential_2"

```
 Layer (type)                         Output Shape                         ␣
↪Param #

 normalization_6 (Normalization)      (None, 8)                                ␣
↪   3

 dense_8 (Dense)                      (None, 128)                            ␣
↪1,152

 dense_9 (Dense)                      (None, 128)                           ␣
↪16,512

 dense_10 (Dense)                     (None, 128)                           ␣
↪16,512

 dense_11 (Dense)                     (None, 1)                                ␣
↪129
```

Total params: 34,308 (134.02 KB)

Trainable params: 34,305 (134.00 KB)

Non-trainable params: 3 (16.00 B)

[94]:
```
model.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.AdamW(learning_rate=0.01),
    metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')]
)
```

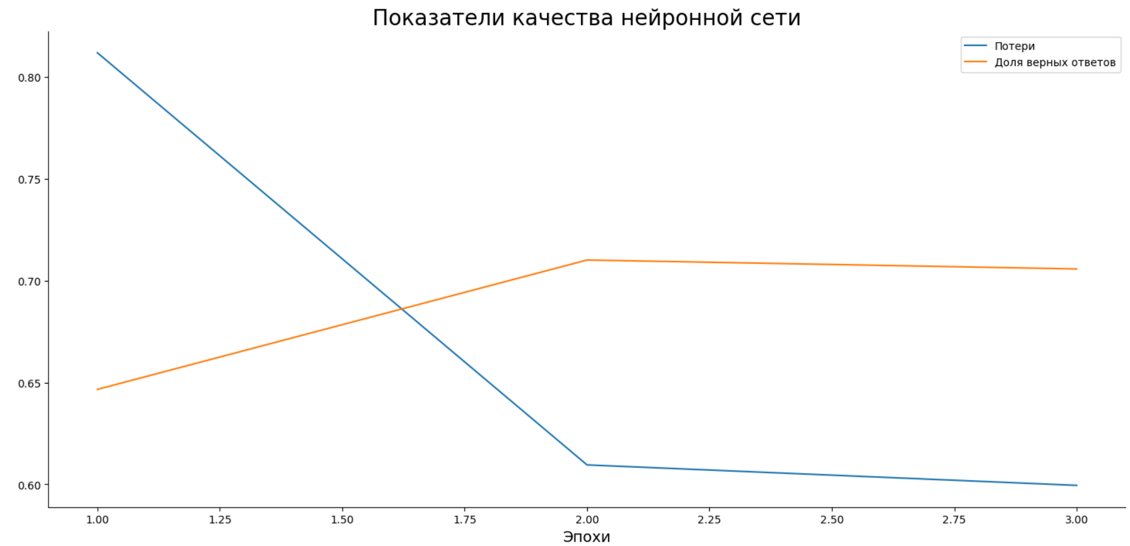
[95]:
```
history = model.fit(X_train, y_train, epochs=3)
```

```
Epoch 1/3
29/29             1s 666us/step -
accuracy: 0.5733 - loss: 1.0464
Epoch 2/3
29/29             0s 639us/step -
accuracy: 0.7073 - loss: 0.6147
Epoch 3/3
29/29             0s 574us/step -
accuracy: 0.7051 - loss: 0.5986
```

[96]:
```
from matplotlib import rcParams

rcParams['figure.figsize'] = (18, 8)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
```

[97]:
```
plt.plot(np.arange(1, 4), history.history['loss'], label='    ')
plt.plot(np.arange(1, 4), history.history['accuracy'], label='           ')
plt.title('                    ', size=20)
plt.xlabel('   ', size=14)
plt.legend();
```



0.9.1

, :

[99]:
```
prediction = model.predict(X_test)
prediction
```

```
13/13              0s 3ms/step
```

[99]:
```
array([[0.7057259 ],
       [0.8097175 ],
       [0.51567745],
       [0.8073348 ],
       [0.8122416 ],
       [0.80478305],
       [0.82047975],
       [0.80475044],
       [0.8190121 ],
       [0.8088746 ],
       [0.81101394],
       [0.81370676],
       [0.80197924],
       [0.599875  ],
       [0.813054  ],
       [0.80083585],
       [0.81420815],
       [0.8086596 ],
       [0.68783474],
       [0.59504104],
```

```
[0.75973654],
[0.7975664 ],
[0.8138774 ],
[0.7973752 ],
[0.8080126 ],
[0.8128584 ],
[0.5107986 ],
[0.8025023 ],
[0.709175  ],
[0.8122195 ],
[0.62666965],
[0.8002905 ],
[0.80622035],
[0.5955385 ],
[0.68783474],
[0.7995473 ],
[0.80614674],
[0.8032245 ],
[0.5135273 ],
[0.8023027 ],
[0.5566317 ],
[0.5141018 ],
[0.8172711 ],
[0.799966  ],
[0.5104643 ],
[0.8103242 ],
[0.7932349 ],
[0.8154781 ],
[0.8021564 ],
[0.80737734],
[0.81327385],
[0.7989514 ],
[0.6053078 ],
[0.64609843],
[0.58754176],
[0.5118887 ],
[0.79944015],
[0.5107236 ],
[0.7353146 ],
[0.51645887],
[0.65563715],
[0.80516356],
[0.8005075 ],
[0.810379  ],
[0.8026767 ],
[0.80793464],
[0.5388659 ],
```

```
[0.8157111 ],
[0.79610205],
[0.80464107],
[0.8107102 ],
[0.8021046 ],
[0.80041236],
[0.8006033 ],
[0.8092524 ],
[0.797369  ],
[0.8061661 ],
[0.8168439 ],
[0.8036368 ],
[0.50610286],
[0.7924508 ],
[0.7919253 ],
[0.80910444],
[0.8161069 ],
[0.8036406 ],
[0.7150682 ],
[0.7623734 ],
[0.817799  ],
[0.80142504],
[0.7840259 ],
[0.81517065],
[0.6929847 ],
[0.80637735],
[0.78950197],
[0.798535  ],
[0.8094209 ],
[0.8040494 ],
[0.6716118 ],
[0.7599703 ],
[0.7989751 ],
[0.8060508 ],
[0.79881984],
[0.80052274],
[0.8034088 ],
[0.82247376],
[0.8075533 ],
[0.8006245 ],
[0.80618995],
[0.7934865 ],
[0.5774561 ],
[0.7784169 ],
[0.7976686 ],
[0.8130052 ],
[0.7826887 ],
```

```
[0.7775237 ],
[0.79145765],
[0.57317984],
[0.80079746],
[0.7953639 ],
[0.8140029 ],
[0.81191933],
[0.8053935 ],
[0.81122035],
[0.8109701 ],
[0.8011112 ],
[0.58983564],
[0.81635445],
[0.79864377],
[0.70595384],
[0.51485217],
[0.81455165],
[0.8042652 ],
[0.7565213 ],
[0.8149642 ],
[0.6785965 ],
[0.80614674],
[0.80017143],
[0.8030819 ],
[0.7827038 ],
[0.6433574 ],
[0.79828346],
[0.80083585],
[0.8012331 ],
[0.8063677 ],
[0.8197042 ],
[0.8055508 ],
[0.80041236],
[0.817127  ],
[0.8162273 ],
[0.7943109 ],
[0.8029992 ],
[0.8103699 ],
[0.8128778 ],
[0.80516857],
[0.80614674],
[0.7999986 ],
[0.7902756 ],
[0.8004604 ],
[0.81668913],
[0.8136674 ],
[0.5806081 ],
```

```
[0.51365376],
[0.80090344],
[0.8121607 ],
[0.6717783 ],
[0.6482744 ],
[0.7847876 ],
[0.5583601 ],
[0.8173468 ],
[0.8174294 ],
[0.79319936],
[0.8073924 ],
[0.5955501 ],
[0.716071  ],
[0.8044831 ],
[0.8006245 ],
[0.78895974],
[0.8090538 ],
[0.5599878 ],
[0.81371605],
[0.8132252 ],
[0.79302585],
[0.7918327 ],
[0.8149867 ],
[0.5147421 ],
[0.77912164],
[0.51327163],
[0.8094855 ],
[0.8190641 ],
[0.815151  ],
[0.8193976 ],
[0.8044236 ],
[0.80421925],
[0.81826085],
[0.81718475],
[0.7670418 ],
[0.8033734 ],
[0.8052938 ],
[0.814296  ],
[0.7888855 ],
[0.8185193 ],
[0.5151159 ],
[0.650571  ],
[0.800704  ],
[0.810379  ],
[0.80516356],
[0.8138512 ],
[0.7995403 ],
```

```
[0.8063677 ],
[0.8079773 ],
[0.8071994 ],
[0.8063677 ],
[0.5915969 ],
[0.8201722 ],
[0.814892  ],
[0.8121464 ],
[0.7607995 ],
[0.59795153],
[0.51620984],
[0.72008055],
[0.8079859 ],
[0.79574543],
[0.8145661 ],
[0.8113483 ],
[0.61308   ],
[0.81187683],
[0.81476265],
[0.8102315 ],
[0.8059361 ],
[0.8080227 ],
[0.62946475],
[0.6086857 ],
[0.7827612 ],
[0.812246  ],
[0.8040629 ],
[0.80858153],
[0.80695045],
[0.7832099 ],
[0.81388354],
[0.80577934],
[0.80619407],
[0.8136809 ],
[0.70084876],
[0.8140754 ],
[0.8013289 ],
[0.8060158 ],
[0.78035116],
[0.80752814],
[0.80083585],
[0.8015517 ],
[0.7318312 ],
[0.78754216],
[0.8003849 ],
[0.7585192 ],
[0.8062244 ],
```

```
[0.80614674],
[0.8002686 ],
[0.68783474],
[0.8102315 ],
[0.5511611 ],
[0.80041236],
[0.812936  ],
[0.8051698 ],
[0.7223875 ],
[0.8011361 ],
[0.7972635 ],
[0.8000835 ],
[0.7946142 ],
[0.8083825 ],
[0.8114393 ],
[0.8059361 ],
[0.80614674],
[0.6785941 ],
[0.8037367 ],
[0.8202473 ],
[0.81825435],
[0.8051698 ],
[0.8100971 ],
[0.81152403],
[0.8037628 ],
[0.80873054],
[0.8063677 ],
[0.8129623 ],
[0.8119131 ],
[0.7937898 ],
[0.8037597 ],
[0.52745914],
[0.7925076 ],
[0.8124865 ],
[0.80054253],
[0.79480743],
[0.80985695],
[0.7796078 ],
[0.8050697 ],
[0.8179883 ],
[0.81238604],
[0.79933786],
[0.81931484],
[0.7643934 ],
[0.79742324],
[0.80564433],
[0.5706054 ],
```

```
[0.51158917],
[0.81666374],
[0.52657425],
[0.810379  ],
[0.7942246 ],
[0.8022599 ],
[0.81457764],
[0.79832864],
[0.814174  ],
[0.80831426],
[0.80516356],
[0.6910186 ],
[0.8135005 ],
[0.6643596 ],
[0.8154745 ],
[0.5151571 ],
[0.7112627 ],
[0.80564433],
[0.8069362 ],
[0.79278237],
[0.5154029 ],
[0.5060167 ],
[0.8173468 ],
[0.7948538 ],
[0.79662776],
[0.81032634],
[0.8089067 ],
[0.7401285 ],
[0.8050842 ],
[0.8096203 ],
[0.7938343 ],
[0.8059059 ],
[0.8063677 ],
[0.60820925],
[0.8031266 ],
[0.7948142 ],
[0.8052008 ],
[0.80427086],
[0.595155  ],
[0.8128208 ],
[0.81268525],
[0.80116355],
[0.79578936],
[0.6147822 ],
[0.80542517],
[0.80834717],
[0.81501365],
```

```
[0.80516356],
[0.52841175],
[0.8056287 ],
[0.7801646 ],
[0.8112396 ],
[0.66635275],
[0.76408327],
[0.81310487],
[0.8059957 ],
[0.81745726],
[0.80396575],
[0.8043804 ],
[0.6399717 ],
[0.8189032 ],
[0.7016444 ],
[0.53412914],
[0.804508  ],
[0.81510043],
[0.8121656 ],
[0.7844235 ],
[0.8123174 ],
[0.8057501 ],
[0.80685663],
[0.8100353 ],
[0.7998341 ],
[0.81476265],
[0.5573513 ],
[0.80657005],
[0.8133599 ],
[0.5416999 ],
[0.81140906],
[0.79813194],
[0.8103699 ],
[0.80374914],
[0.81745726],
[0.5323472 ],
[0.7954198 ],
[0.81516755],
[0.8115436 ],
[0.80268633],
[0.78749245],
[0.7942627 ],
[0.81732875],
[0.785655  ]], dtype=float32)
```

(0.5):

```
[100]: y_pred = np.array([1 if prob > 0.5 else 0 for prob in np.ravel(prediction)])
       print(y_pred)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

:

```
[101]: loss, accuracy = model.evaluate(X_test, y_test)
       loss, accuracy
```

```
13/13              0s 603us/step -
accuracy: 0.6596 - loss: 0.6349
```

```
[101]: (0.6223652362823486, 0.6793892979621887)
```

```
[112]: confusion_matrix(y_test, y_pred)
```

```
[112]: array([[267,   0],
              [126,   0]])
```

```
[113]: tpr_score(y_test, y_pred), fpr_score(y_test, y_pred)
```

```
[113]: (1.0, 1.0)
```

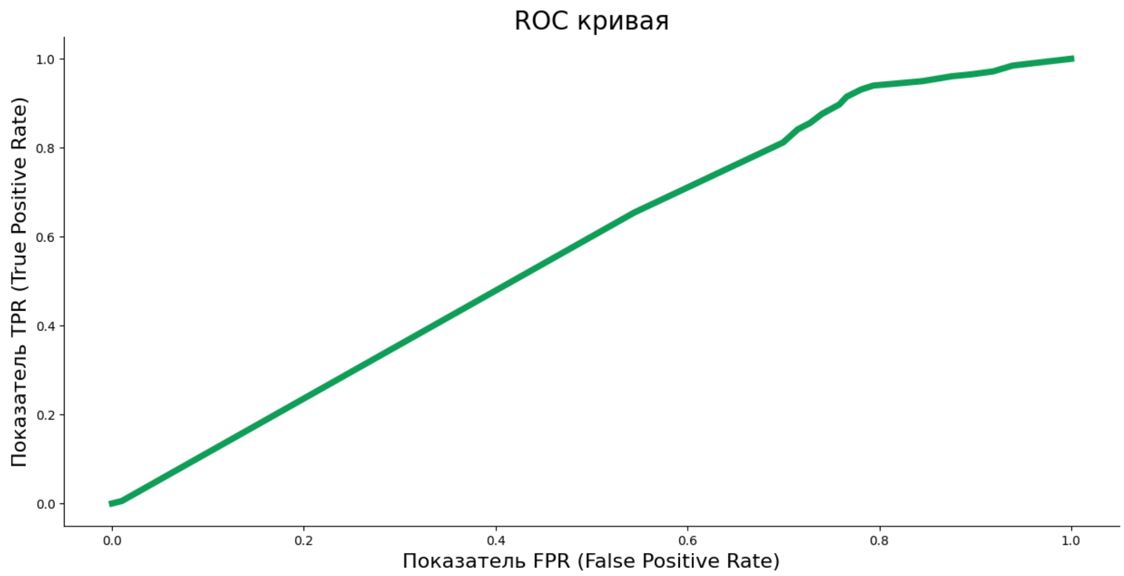
```
[114]: prediction = model.predict(X)
       prediction.shape
```

```
41/41              0s 513us/step
```

```
[114]: (1307, 1)
```

```
[115]: plt.figure(figsize=(15,7))

       ROC = roc_from_scratch(prediction.reshape(-1),y,partitions=50)
       #plt.scatter(ROC[:,0],ROC[:,1],color='#0F9D58',s=100)
       plt.plot(ROC[:,0],ROC[:,1],color='#0F9D58',lw=5)
       plt.title('ROC     ',fontsize=20)
       plt.xlabel('      FPR (False Positive Rate)',fontsize=16)
       plt.ylabel('      TPR (True Positive Rate)',fontsize=16);
```



[116]:
```
# Calculate AUC and it is very low
auc = auc_trapezoidal(ROC)
auc
```

[116]: 0.5727334480320265

[117]:
```
y_train.shape, y_test.shape
```

[117]: ((914,), (393,))

[118]:
```
def to_one_hot(labels, dimension=3):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results
```

[119]:
```
list(enumerate(['a','b','c']))
```

[119]: [(0, 'a'), (1, 'b'), (2, 'c')]

[120]:
```
y_train = to_one_hot(y_train)
y_test = to_one_hot(y_test)
y_train.shape, y_test.shape
```

[120]: ((914, 3), (393, 3))

[121]:
```
y_train
```

[121]:
```
array([[0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       …,
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.]])
```

[122]:
```
feature_normalizer = tf.keras.layers.Normalization(axis=None,input_shape=(X.
 ↪shape[1],))
feature_normalizer.adapt(X_train)
```

[123]:
```
model = tf.keras.Sequential([
    feature_normalizer,
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.Dense(3, activation="softmax")
])
```

[124]:
```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

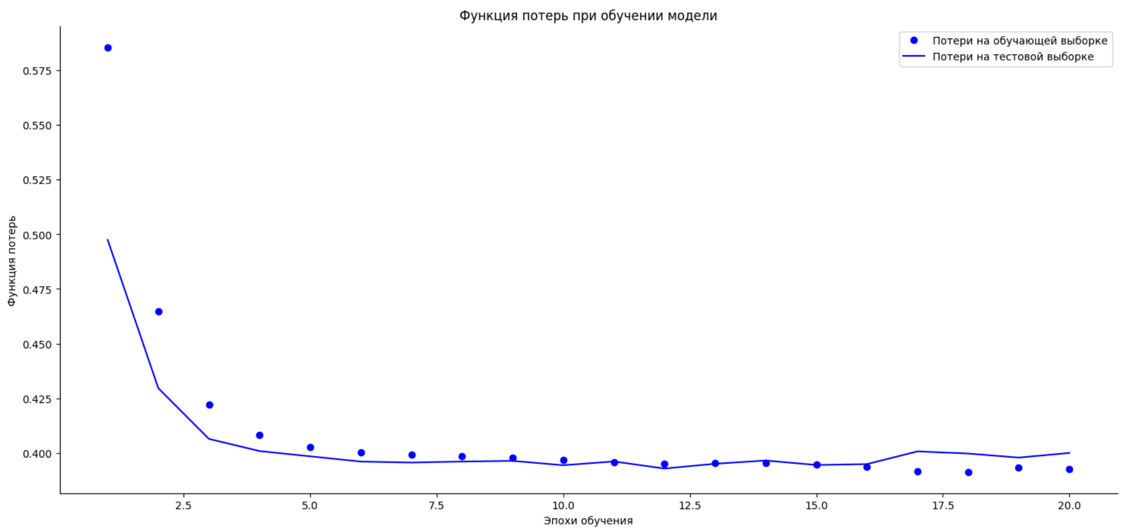
[125]:
```
history = model.fit(X_train,
                    y_train,
                    epochs=20,
                    #
                    verbose=1,
                    #       (     )    20%
                    validation_split = 0.2)
```

```
Epoch 1/20
23/23                 1s 5ms/step -
accuracy: 0.6067 - loss: 0.6341 - val_accuracy: 0.7158 - val_loss: 0.4974
Epoch 2/20
23/23                 0s 1ms/step -
accuracy: 0.7241 - loss: 0.4752 - val_accuracy: 0.7158 - val_loss: 0.4296
Epoch 3/20
23/23                 0s 1ms/step -
accuracy: 0.6951 - loss: 0.4323 - val_accuracy: 0.7213 - val_loss: 0.4065
Epoch 4/20
23/23                 0s 1ms/step -
accuracy: 0.7129 - loss: 0.4029 - val_accuracy: 0.7049 - val_loss: 0.4009
Epoch 5/20
23/23                 0s 1ms/step -
accuracy: 0.6911 - loss: 0.4085 - val_accuracy: 0.7104 - val_loss: 0.3985
Epoch 6/20
23/23                 0s 1ms/step -
```

```
accuracy: 0.7199 - loss: 0.3947 - val_accuracy: 0.7158 - val_loss: 0.3962
Epoch 7/20
23/23            0s 1ms/step -
accuracy: 0.6982 - loss: 0.3985 - val_accuracy: 0.7049 - val_loss: 0.3957
Epoch 8/20
23/23            0s 1ms/step -
accuracy: 0.6936 - loss: 0.4057 - val_accuracy: 0.6995 - val_loss: 0.3962
Epoch 9/20
23/23            0s 1ms/step -
accuracy: 0.7258 - loss: 0.3891 - val_accuracy: 0.7104 - val_loss: 0.3965
Epoch 10/20
23/23            0s 1ms/step -
accuracy: 0.7094 - loss: 0.3966 - val_accuracy: 0.7049 - val_loss: 0.3945
Epoch 11/20
23/23            0s 1ms/step -
accuracy: 0.7298 - loss: 0.3834 - val_accuracy: 0.7049 - val_loss: 0.3962
Epoch 12/20
23/23            0s 1ms/step -
accuracy: 0.7157 - loss: 0.3886 - val_accuracy: 0.7104 - val_loss: 0.3930
Epoch 13/20
23/23            0s 1ms/step -
accuracy: 0.7102 - loss: 0.3933 - val_accuracy: 0.7049 - val_loss: 0.3951
Epoch 14/20
23/23            0s 1ms/step -
accuracy: 0.7130 - loss: 0.3942 - val_accuracy: 0.7158 - val_loss: 0.3966
Epoch 15/20
23/23            0s 1ms/step -
accuracy: 0.7005 - loss: 0.4018 - val_accuracy: 0.6995 - val_loss: 0.3946
Epoch 16/20
23/23            0s 2ms/step -
accuracy: 0.6990 - loss: 0.4017 - val_accuracy: 0.7104 - val_loss: 0.3950
Epoch 17/20
23/23            0s 1ms/step -
accuracy: 0.7243 - loss: 0.3884 - val_accuracy: 0.7322 - val_loss: 0.4008
Epoch 18/20
23/23            0s 1ms/step -
accuracy: 0.7042 - loss: 0.3941 - val_accuracy: 0.7158 - val_loss: 0.3998
Epoch 19/20
23/23            0s 1ms/step -
accuracy: 0.7080 - loss: 0.3986 - val_accuracy: 0.7158 - val_loss: 0.3980
Epoch 20/20
23/23            0s 1ms/step -
accuracy: 0.6984 - loss: 0.4030 - val_accuracy: 0.7104 - val_loss: 0.4001
```

[126]:
```
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(loss) + 1)
```

```
plt.plot(epochs, loss, "bo", label="                ")
plt.plot(epochs, val_loss, "b", label="               ")
plt.title("                    ")
plt.xlabel("        ")
plt.ylabel("        ")
plt.legend();
```




```
[ ]:
```