Home work 7

Kevin Ritter 4/7/19 Training Boosted Trees models in TensorFlow

In [1]:
```
# Data preparation
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

tf.enable_eager_execution()

tf.logging.set_verbosity(tf.logging.ERROR)

# new random seed
tf.set_random_seed(966)

# load dataset:
# training data frame
dftrain = pd.read_csv('https://storage.googleapis.com/tfbt/titanic_train.csv')
y_train = dftrain.pop('survived')
# testing data frame
dfeval = pd.read_csv('https://storage.googleapis.com/tfbt/titanic_eval.csv')
y_eval = dfeval.pop('survived')
```

1.

Run the four commands below and show the output for each:

In [2]:
```
dftrain.head()
```

Out[2]:

	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n
1	female	38.0	1	0	71.2833	First	C	Cherbourg	n
2	female	26.0	0	0	7.9250	Third	unknown	Southampton	y
3	female	35.0	1	0	53.1000	First	C	Southampton	n
4	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y

In [3]:
```
dftrain.describe()
```

Out[3]:

	age	n_siblings_spouses	parch	fare
count	627.000000	627.000000	627.000000	627.000000
mean	29.631308	0.545455	0.379585	34.385399
std	12.511818	1.151090	0.792999	54.597730
min	0.750000	0.000000	0.000000	0.000000
25%	23.000000	0.000000	0.000000	7.895800
50%	28.000000	0.000000	0.000000	15.045800
75%	35.000000	1.000000	0.000000	31.387500
max	80.000000	8.000000	5.000000	512.329200

In [4]:
```
dfeval.head()
```

Out[4]:

	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	male	35.0	0	0	8.0500	Third	unknown	Southampton	y
1	male	54.0	0	0	51.8625	First	E	Southampton	y
2	female	58.0	0	0	26.5500	First	C	Southampton	y
3	female	55.0	0	0	16.0000	Second	unknown	Southampton	y
4	male	34.0	0	0	13.0000	Second	D	Southampton	y

In [5]:
```
dfeval.describe()
```

Out[5]:

	age	n_siblings_spouses	parch	fare
count	264.000000	264.000000	264.000000	264.000000
mean	28.720985	0.469697	0.386364	27.023880
std	14.157538	0.978393	0.837775	34.973108
min	0.420000	0.000000	0.000000	0.000000
25%	21.000000	0.000000	0.000000	7.925000
50%	28.000000	0.000000	0.000000	13.250000
75%	35.250000	1.000000	0.000000	27.900000
max	74.000000	8.000000	6.000000	263.000000

2.

Show the output for shape. What do other values mean?

In [6]:
```
dftrain.shape[0], dfeval.shape[0]
```

Out[6]:
```
(627, 264)
```

In [7]:
```
dftrain.shape[1], dfeval.shape[1]
```

Out[7]:
```
(9, 9)
```

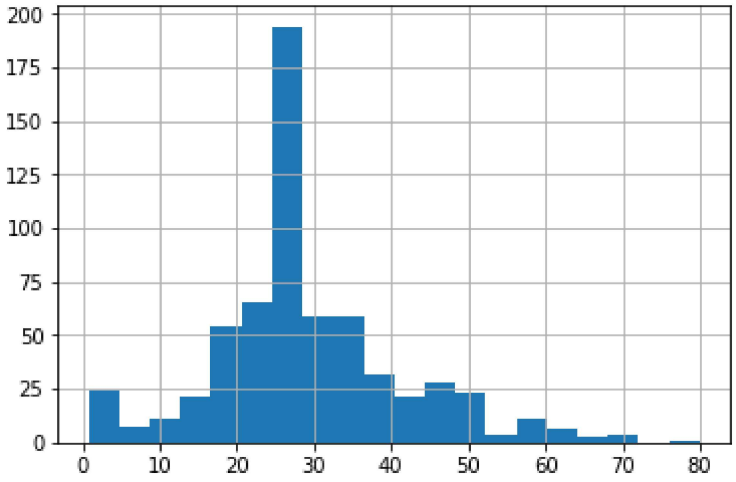
The index numbers in '.shape[x]' indicate the observed axis for a data frame. The training and test data have 627 and 264 rows respectively. The training and test data have 9 columns each. Changing to axis 2 produces an error by searching for a 3rd dimention that does not exists.

3.

What information can you ascertain about age and gender about the passengers by plotting their histograms?

In [8]:
```
dftrain.age.hist(bins=20)
```

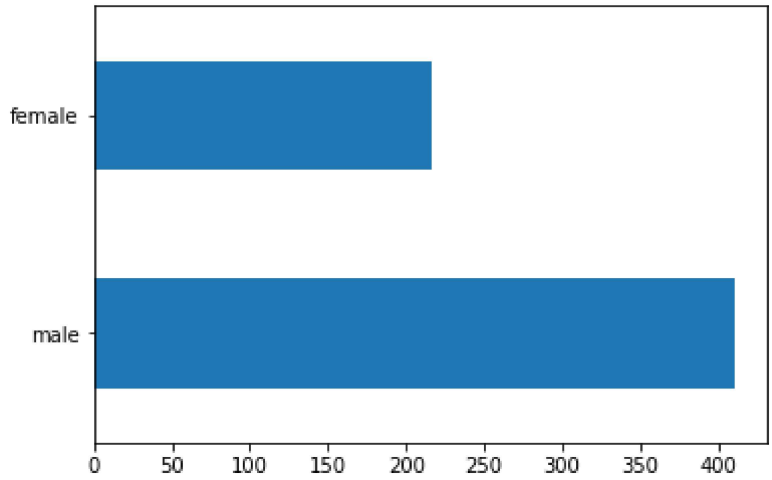
Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2986a57c4a8>
```



The age plot shows a majority of the passengers were between 15 and 50 years old. The group around 25-26 was far and away the most populous.

In [9]:
```
dftrain.sex.value_counts().plot(kind='barh')
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2986c668cf8>
```



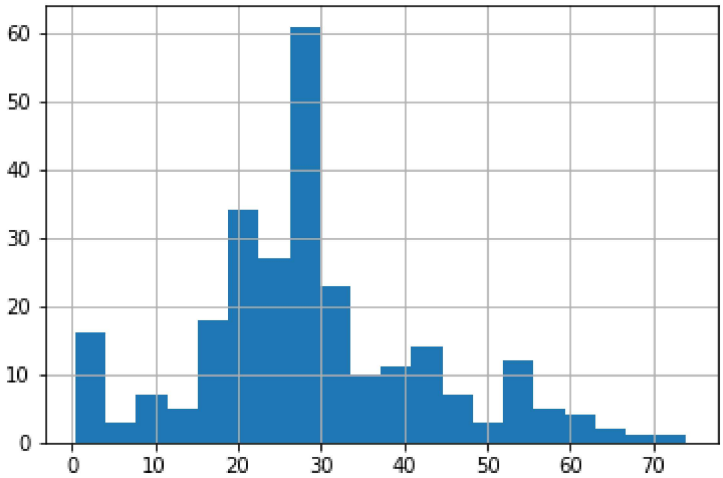
The gender plot shows nearly two thirds of the passengers were male.

4.

Run the plots again for the test data. What information can be gleaned about the population?

In [10]:
```
dfeval.age.hist(bins=20)
```

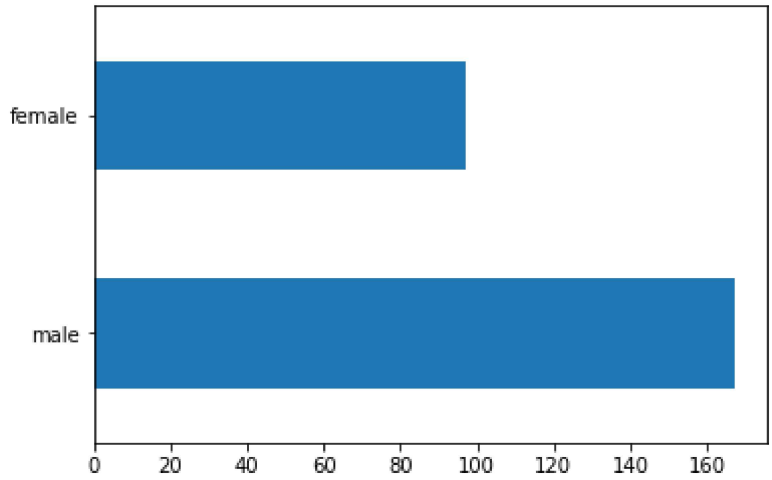
Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2986c6e7eb8>
```



This group is a little more scattered. The largest single subset is now 28-30.

In [11]:
```
dfeval.sex.value_counts().plot(kind='barh')
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2986c7726d8>
```



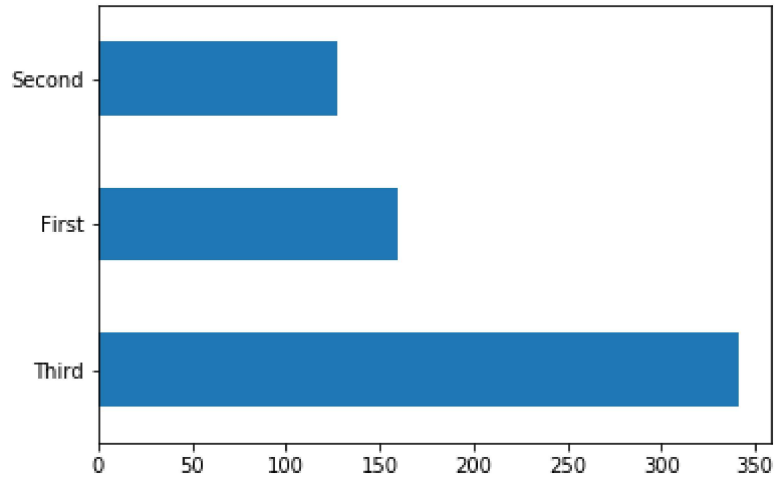
The disparity between male and female again is close to 2/3 to 1/3.

5.

Run the given code and show the outputs. These two plots divide the passengers by comfort and level of service (fist, second, and third 'class') as well as boarding location aka 'embark_town.'

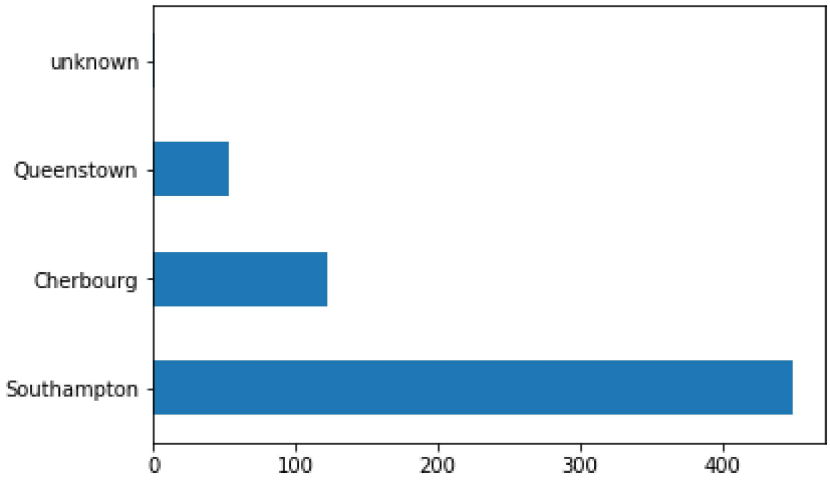
In [12]:
```
(dftrain['class'].value_counts().plot(kind='barh'))
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2986c7e6b70>
```



In [13]:
```
(dftrain['embark_town'].value_counts().plot(kind='barh'))
```

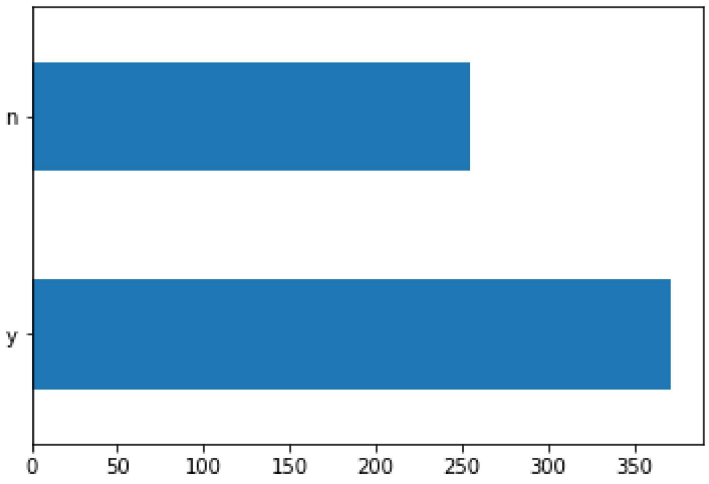
Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2986c8419e8>
```



Running the code a third time I changed the column choice to whether passengers traveled 'alone' or as a family. It turns out many more people traveled alone that I expected.

In [14]:
```
(dftrain['alone'].value_counts().plot(kind='barh'))
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2986c89fd68>
```

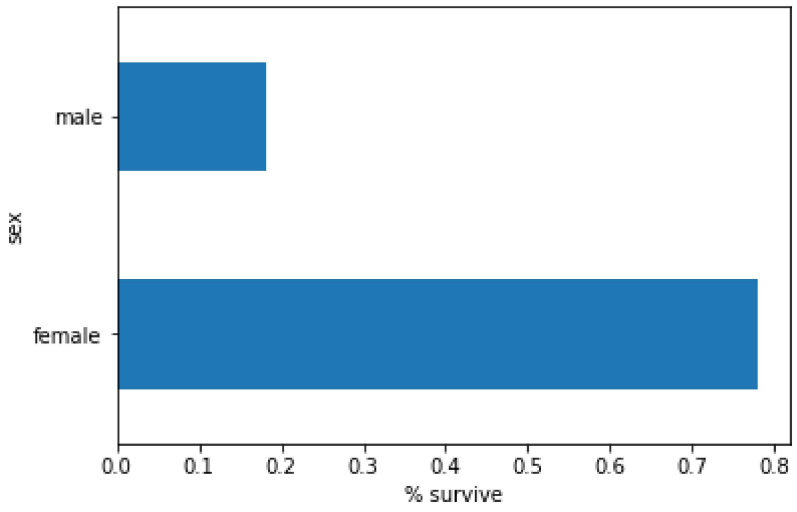


6.

Run the following and show your output:

In [15]:
```
ax = (pd.concat([dftrain, y_train], axis=1)\
      .groupby('sex')
      .survived.mean()
      .plot(kind='barh'))
ax.set_xlabel('% survive')
```

Out[15]: Text(0.5, 0, '% survive')

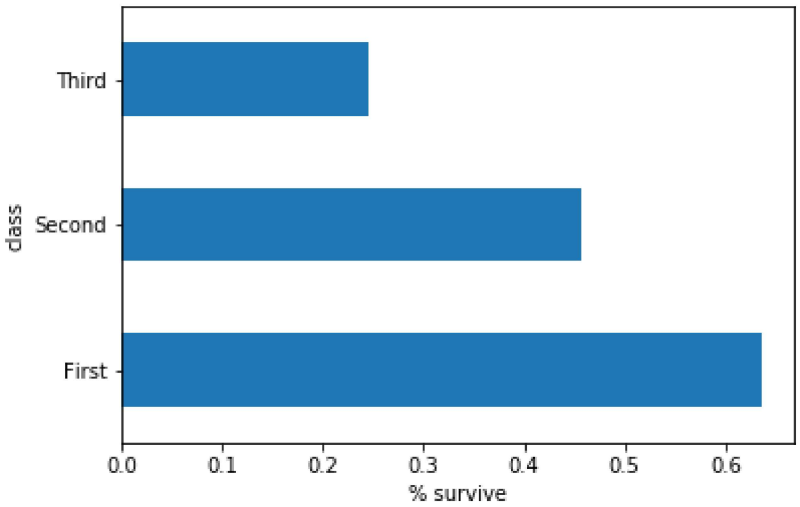


7.

Change the above code to see survival by 'class' and show the output.

In [16]:
```
ax = (pd.concat([dftrain, y_train], axis=1)\
      .groupby('class')
      .survived.mean()
      .plot(kind='barh'))
ax.set_xlabel('% survive')
```

Out[16]: Text(0.5, 0, '% survive')



Under the category of 'no suprises there' we have: first class had the best chance of survival followed by second class and then third.

In [17]:
```
fc = tf.feature_column
CATEGORICAL_COLUMNS = ['sex', 'n_siblings_spouses', 'parch', 'class', 'deck',
                       'embark_town', 'alone']
NUMERIC_COLUMNS = ['age', 'fare']

def one_hot_cat_column(feature_name, vocab):
  return fc.indicator_column(
      fc.categorical_column_with_vocabulary_list(feature_name,
                                                 vocab))
feature_columns = []
for feature_name in CATEGORICAL_COLUMNS:
  # Need to one-hot encode categorical features.
  vocabulary = dftrain[feature_name].unique()
  feature_columns.append(one_hot_cat_column(feature_name, vocabulary))

for feature_name in NUMERIC_COLUMNS:
  feature_columns.append(fc.numeric_column(feature_name,
                                           dtype=tf.float32))
```

8.

Run the given code and show the output. Why do we need to one-hot encode the class?

In [18]:
```
example = dftrain.head(1)
class_fc = one_hot_cat_column('class', ('First', 'Second', 'Third'))
print('Feature value: "{}"'.format(example['class'].iloc[0]))
print('One-hot encoded: ', fc.input_layer(dict(example), [class_fc]).numpy())
```

```
Feature value: "Third"
One-hot encoded:  [[0. 0. 1.]]
```

We one hot encode categorical variables when their values are mutually exclusive.

In [19]:
```
fc.input_layer(dict(example), feature_columns).numpy()
```

Out[19]:
```
array([[22.  ,  1.  ,  0.  ,  1.  ,  0.  ,  0.  ,  1.  ,  0.  ,  0.  ,
         0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  1.  ,  0.  ,  0.  ,  0.  ,
         7.25,  1.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  1.  ,
         0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  1.  ,  0.  ]], dtype=float32)
```

9.

Run the above and explain what information it is providing us. This is showing the output of a single row after it has had categorical variables one hot encoded. We now have 34 columns as opposed to the original 9.

In [20]:
```
# Use entire batch since this is such a small dataset.
NUM_EXAMPLES = len(y_train)

def make_input_fn(x, y, n_epochs=None, shuffle=True):
  def input_fn():
    dataset = tf.data.Dataset.from_tensor_slices((dict(x), y))
    if shuffle:
      dataset = dataset.shuffle(NUM_EXAMPLES)
    # For training, cycle thru dataset as many times as need (n_epochs=None).
    dataset = dataset.repeat(n_epochs)
    # In memory training doesn't use batching.
    dataset = dataset.batch(NUM_EXAMPLES)
    return dataset
  return input_fn

# Training and evaluation input functions.
train_input_fn = make_input_fn(dftrain, y_train)
eval_input_fn = make_input_fn(dfeval, y_eval, shuffle=False, n_epochs=1)
```

10.

Run the given code for logistic regression model to establish a benchmark. Describe what the classifier is doing.

In [21]:
```
linear_est = tf.estimator.LinearClassifier(feature_columns)

# Train model.
linear_est.train(train_input_fn, max_steps=100)

# Evaluation.
results = linear_est.evaluate(eval_input_fn)
print('Accuracy : ', results['accuracy'])
print('Dummy model: ', results['accuracy_baseline'])
```

```
Accuracy :  0.78409094
Dummy model:  0.625
```

This classifier is using a single epoch, regression model, with no shuffeling to predict survival of the passengers.

In [22]:
```
# Since data fits into memory, use entire dataset per layer. It will be faste
r.
# Above one batch is defined as the entire dataset.
n_batches = 1
est = tf.estimator.BoostedTreesClassifier(feature_columns,
                                          n_batches_per_layer=n_batches)

# The model will stop training once the specified number of trees is built, no
t
# based on the number of steps.
est.train(train_input_fn, max_steps=100)

# Eval.
results = est.evaluate(eval_input_fn)
print('Accuracy : ', results['accuracy'])
print('Dummy model: ', results['accuracy_baseline'])
```

```
Accuracy :  0.82954544
Dummy model:  0.625
```

11.

Describe what the above code is doing in terms of boosted tree classifiers.

Boosted tree classifiers are used to make non-linear connections between features and the target. This is useful for our model because we only have two features that might influence the prediction linearly: age and money spent. The model makes a sequence of base models, or trees to quickly model decisions then evaluates them by gradient boosting (descent).

12.

What happens if we increase the batch size beyond 1?

In [23]:
```
# Since data fits into memory, use entire dataset per layer. It will be faste
r.
# Above one batch is defined as the entire dataset.
n_batches = 5
est = tf.estimator.BoostedTreesClassifier(feature_columns,
                                          n_batches_per_layer=n_batches)

# The model will stop training once the specified number of trees is built, no
t
# based on the number of steps.
est.train(train_input_fn, max_steps=100)

# Eval.
results = est.evaluate(eval_input_fn)
print('Accuracy : ', results['accuracy'])
print('Dummy model: ', results['accuracy_baseline'])
```

```
Accuracy :  0.8030303
Dummy model:  0.625
```

In [24]:
```
# Since data fits into memory, use entire dataset per layer. It will be faste
r.
# Above one batch is defined as the entire dataset.
n_batches = 2
est = tf.estimator.BoostedTreesClassifier(feature_columns,
                                          n_batches_per_layer=n_batches)

# The model will stop training once the specified number of trees is built, no
t
# based on the number of steps.
est.train(train_input_fn, max_steps=100)

# Eval.
results = est.evaluate(eval_input_fn)
print('Accuracy : ', results['accuracy'])
print('Dummy model: ', results['accuracy_baseline'])
```

```
Accuracy :  0.8068182
Dummy model:  0.625
```

I tried a couple different batch sizes, both had negative effects on model accuracy.

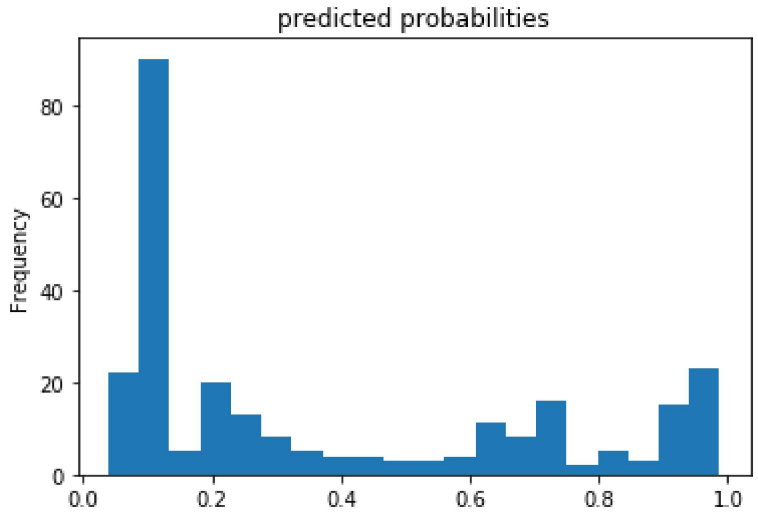
13.

Explain how the boosted tree classifier differs from the linear classifier.

The boosted tree classifier

In [25]:
```
pred_dicts = list(est.predict(eval_input_fn))
probs = pd.Series([pred['probabilities'][1] for pred in pred_dicts])

probs.plot(kind='hist', bins=20, title='predicted probabilities');
```



15.

Provide an explanation of the output.

This histogram is showing the number of people in the testing data whose predicted probability of survival falls in each decimal equivalent of percent. This is multi modal, with the greatest mode near 0.1 or 10% chance to survive.

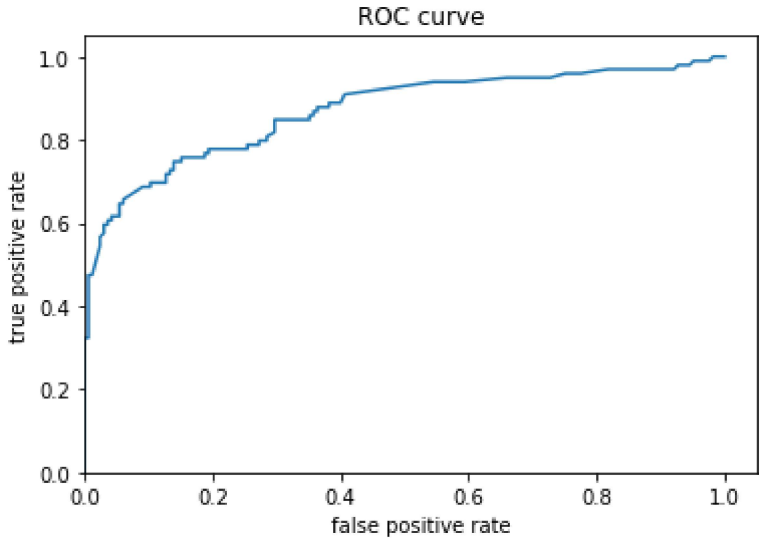
16.

Run the following code and show the output.

In [26]:
```
from sklearn.metrics import roc_curve
from matplotlib import pyplot as plt

fpr, tpr, _ = roc_curve(y_eval, probs)
plt.plot(fpr, tpr)
plt.title('ROC curve')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.xlim(0,)
plt.ylim(0,)
```

Out[26]: (0, 1.05)



17.

What general information does the ROC curve provide about the survival rate of passengers on the Titanic?

When reading an ROC curve we want to maximize the area under the curve. Given the shape, this means moving the curve up and to the left that is higher y-axis values for lower x-axis values. This particular curve tells us that our model is fairly accurate in it's predictions for survivors on the titanic, however there are some false positives skewing the graph to the right.