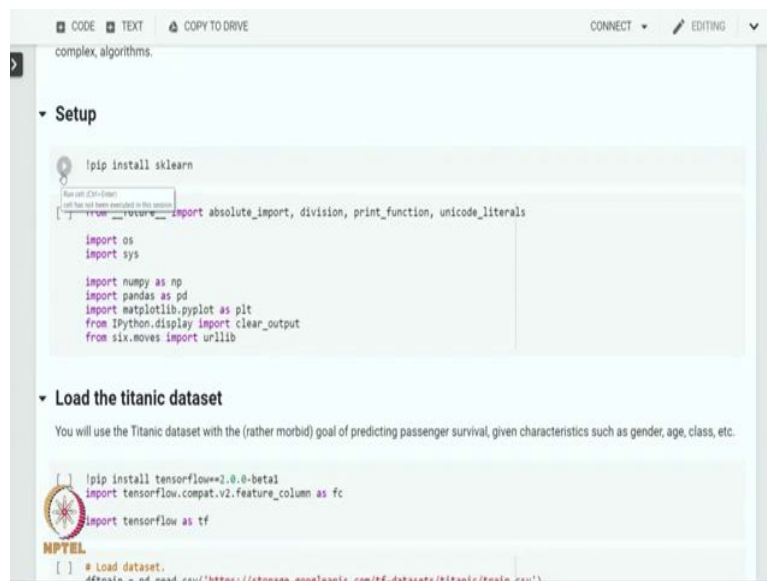


Practical Machine Learning
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 28
Logistic Regression

In this session, we will build a Logistic Regression model with tf.estimator API. Logistic regression is often used as a baseline for classification tasks.

(Refer Slide Time: 00:32)



```
CODE TEXT COPY TO DRIVE CONNECT EDITING
complex algorithms.

Setup

!pip install sklearn
[Run cell (Ctrl-Enter)]
[Cell has not been executed in this session]
import absolute_import, division, print_function, unicode_literals

import os
import sys

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import clear_output
from six.moves import urllib

Load the titanic dataset

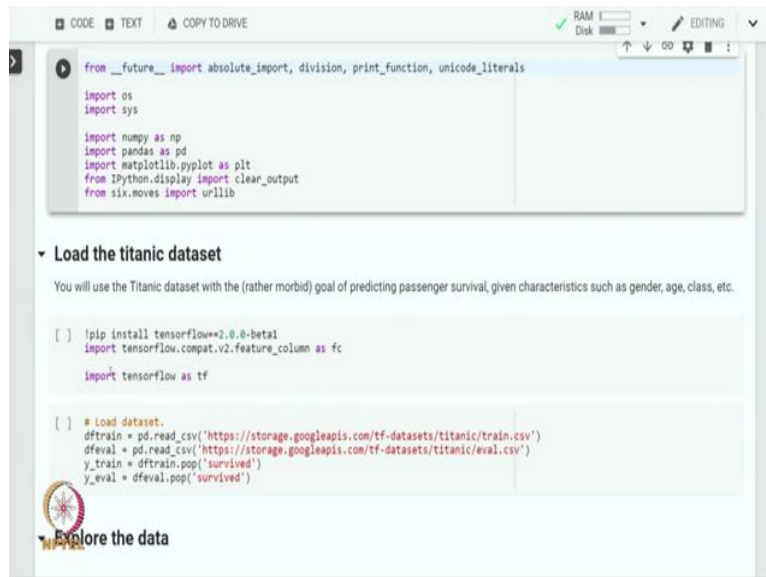
You will use the Titanic dataset with the (rather morbid) goal of predicting passenger survival, given characteristics such as gender, age, class, etc.

!pip install tensorflow==2.0.0-beta1
import tensorflow.compat.v2.feature_column as fc
import tensorflow as tf

MPTEL
[ ] # Load dataset.
dftrain = pd.read_csv("https://storage.googleapis.com/tf-datasets/titanic/train.csv")
```

In this exercise, we will use a titanic dataset with a goal of predicting passenger survival given characteristics such as gender, age and class. Let us first install the required libraries like sklearn and tensorflow. We use matplotlib for plotting and numpy and pandas for data manipulation.

(Refer Slide Time: 01:03)



```
from __future__ import absolute_import, division, print_function, unicode_literals

import os
import sys

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import clear_output
from six.moves import urllib

# Load the titanic dataset

You will use the Titanic dataset with the (rather morbid) goal of predicting passenger survival, given characteristics such as gender, age, class, etc.

[ ] !pip install tensorflow==2.0.0-beta1
import tensorflow.compat.v2.feature_column as fc

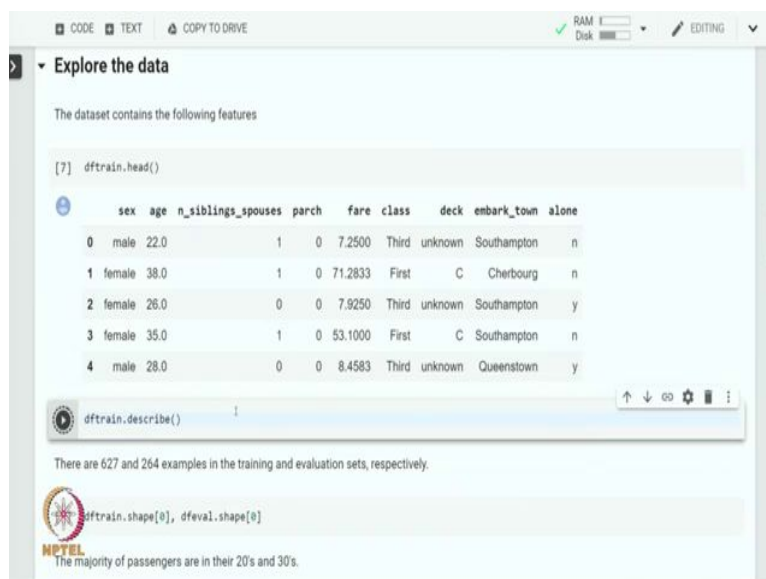
import tensorflow as tf

[ ] # Load dataset.
dfttrain = pd.read_csv('https://storage.googleapis.com/tf-datasets/titanic/train.csv')
dfeval = pd.read_csv('https://storage.googleapis.com/tf-datasets/titanic/eval.csv')
y_train = dfttrain.pop('survived')
y_eval = dfeval.pop('survived')
```

Explore the data

Let us load the titanic dataset, we will install tensorflow 2.0.

(Refer Slide Time: 01:10)



```
Explore the data

The dataset contains the following features

[7] dfttrain.head()

   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

dfttrain.describe()

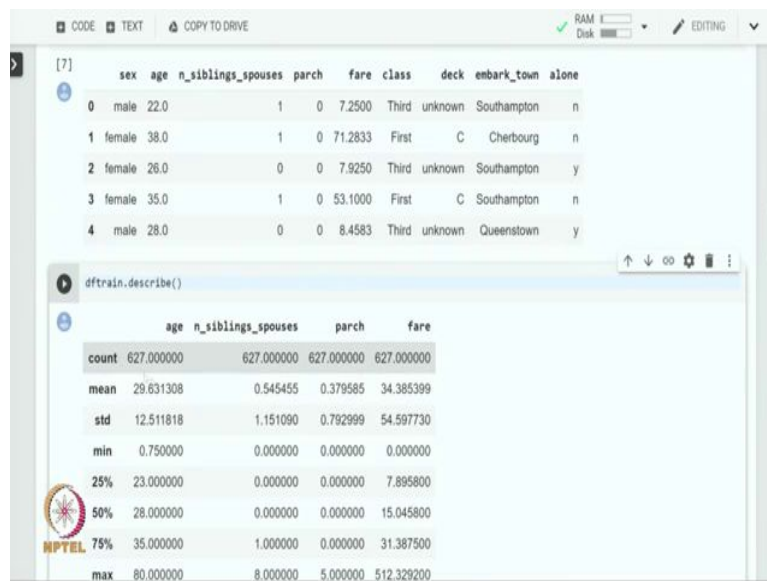
There are 627 and 264 examples in the training and evaluation sets, respectively.

dfttrain.shape[0], dfeval.shape[0]

The majority of passengers are in their 20's and 30's.
```

Let us load the dataset. So, we have training data in dfttrain dataframe, the evaluation data in dfeval, and the respective labels are in y_train and y_eval. Let us first explore the data, we examine a few rows in the dataset. So, you can see that there are features like the sex of the passenger, the age, the number of siblings and spouses, the fare, the class etcetera.

(Refer Slide Time: 01:59)

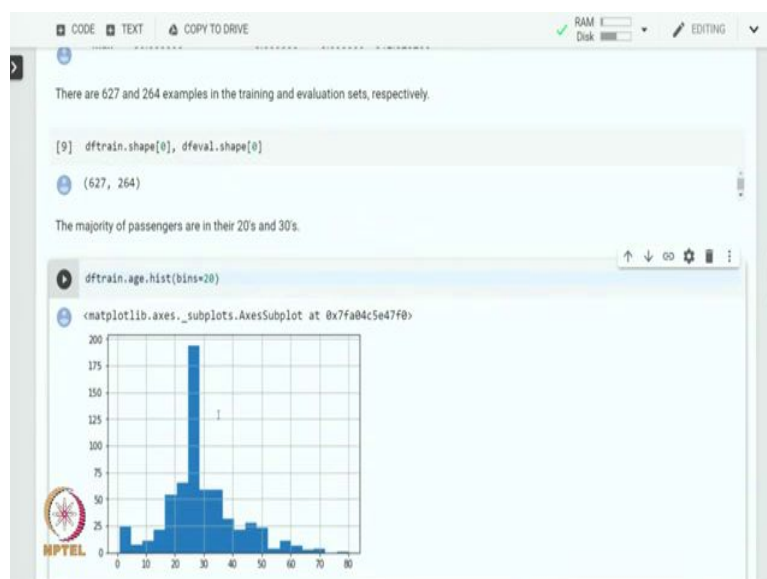


The screenshot shows a Jupyter Notebook interface. The top part displays a data frame with columns: sex, age, n_siblings_spouses, parch, fare, class, deck, embark_town, and alone. The bottom part shows the output of the `dftrain.describe()` command, which provides summary statistics for the numeric columns.

	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

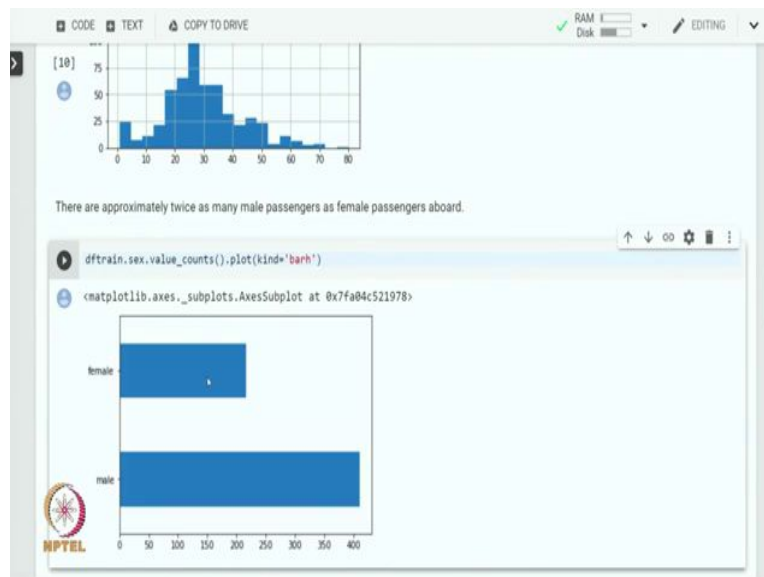
Let us also use a `describe` command on the data frame to get statistics about each of the numeric columns. So, we will get statistics like count, mean, standard deviation and various quartiles. You can see that there are 627 examples in training and 264 examples in evaluation set.

(Refer Slide Time: 02:22)



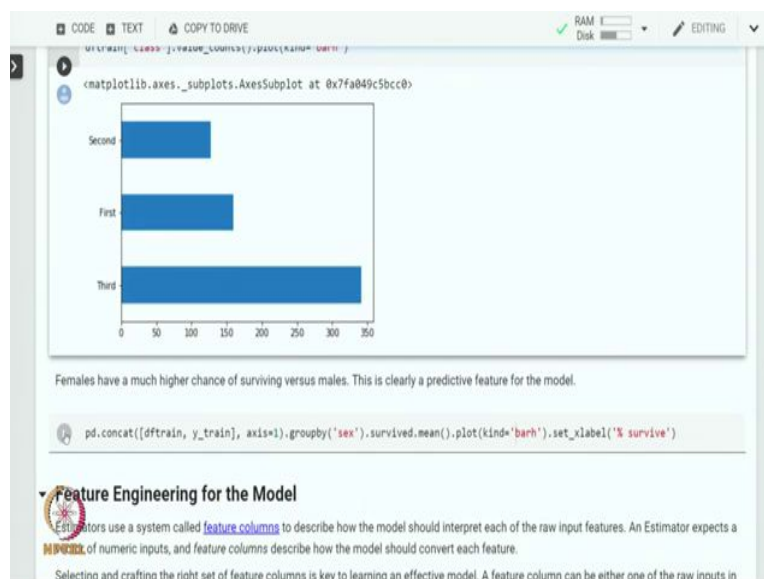
Let us plot histogram of the age and we can see that the majority of the passengers are in their 20s and 30s.

(Refer Slide Time: 02:35)



There are approximately twice as many male passengers as female passengers on board.

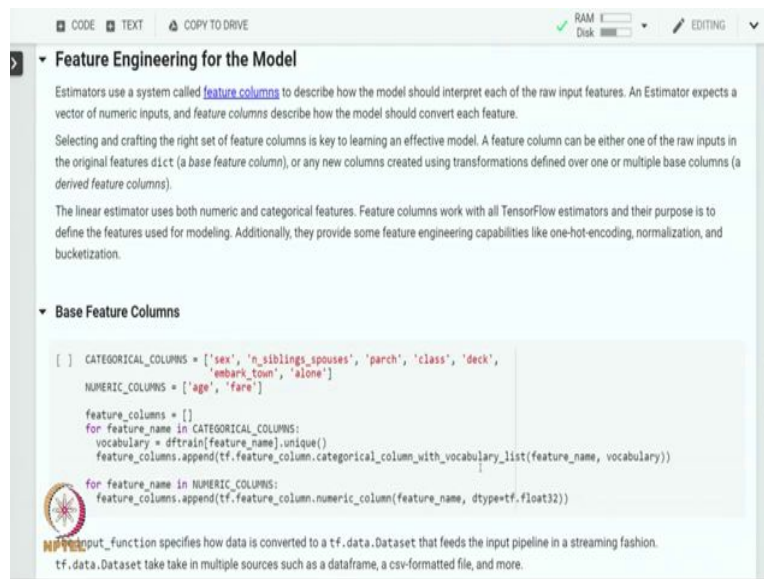
(Refer Slide Time: 02:41)



You can also see that a majority of the passengers were in the third class. Let us look at the survival by sex of the passenger. And you can see that females have higher chances of

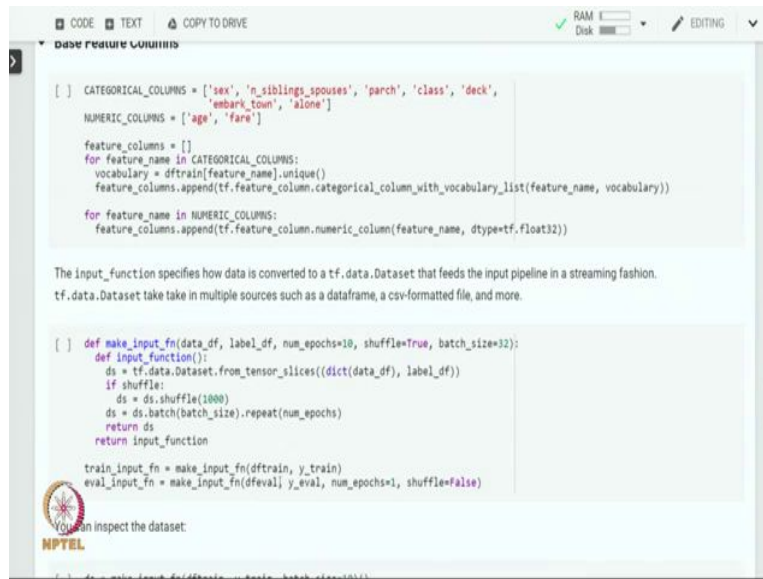
survival than their male counterparts. So, you can see that sex can be a very good predictive feature for the model. After exploring data a bit, let us get into engineering features for the model. Feature engineering is extremely important to get good results.

(Refer Slide Time: 03:22)



So, we will be using feature columns for converting the raw features into a form that can be consumed by estimator API as well as for constructing cross features from the original features. So, we have several categorical features and a few numeric features. We use numeric feature columns for numeric features. For categorical features, we first find out unique values from each of the categorical features and use categorical column with vocabulary list for converting the categorical features into one hot encoding.

(Refer Slide Time: 04:03)



```
[ ] CATEGORICAL_COLUMNS = ['sex', 'n_siblings_spouses', 'parch', 'class', 'deck',
                             'embark_town', 'alone']
    NUMERIC_COLUMNS = ['age', 'fare']

    feature_columns = []
    for feature_name in CATEGORICAL_COLUMNS:
        vocabulary = dftrain[feature_name].unique()
        feature_columns.append(tf.feature_column.categorical_column_with_vocabulary_list(feature_name, vocabulary))

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

    The input_function specifies how data is converted to a tf.data.Dataset that feeds the input pipeline in a streaming fashion.
    tf.data.Dataset take take in multiple sources such as a dataframe, a csv-formatted file, and more.

    [ ] def make_input_fn(data_df, label_df, num_epochs=10, shuffle=True, batch_size=32):
        def input_function():
            ds = tf.data.Dataset.from_tensor_slices((dict(data_df), label_df))
            if shuffle:
                ds = ds.shuffle(1000)
            ds = ds.batch(batch_size).repeat(num_epochs)
            return ds
        return input_function

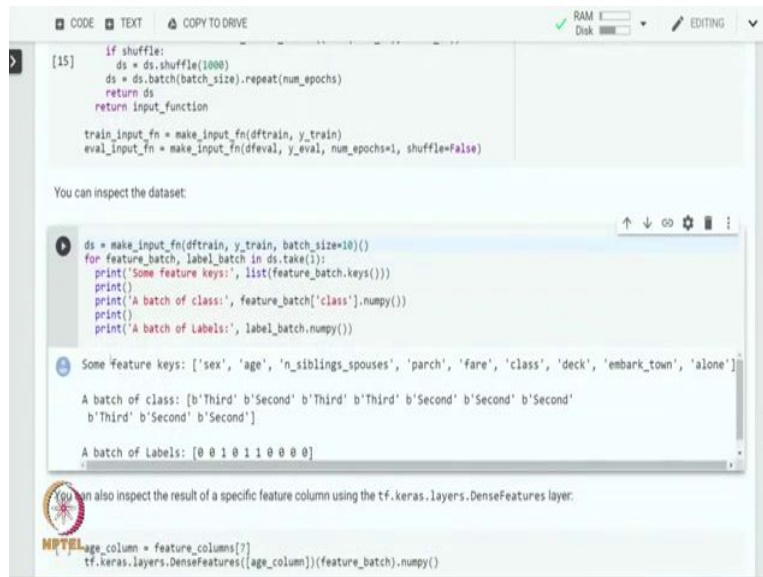
    train_input_fn = make_input_fn(dftrain, y_train)
    eval_input_fn = make_input_fn(dfeval, y_eval, num_epochs=1, shuffle=False)

    You can inspect the dataset:
```

Then we define input function to specify how data is converted into `tf.data.Dataset` format that feeds the input pipeline in a streaming fashion. The dataset takes multiple sources such as dataframe, csv, formatted files and many more. Let us define a dataset from tensor slices and in if required shuffle the dataset and return the dataset in the batches.

We repeat the batching operation for the number of specified epochs. This is how we define our input function and, we make the input function for training as well as for evaluation. In training we set shuffle to be true, and in evaluation we set shuffle to be false and we specified number of epochs to be one in case of evaluation. Let us inspect the dataset.

(Refer Slide Time: 05:09)



```
if shuffle:
    ds = ds.shuffle(1000)
ds = ds.batch(batch_size).repeat(num_epochs)
return ds

train_input_fn = make_input_fn(dfttrain, y_train)
eval_input_fn = make_input_fn(dfeval, y_eval, num_epochs=1, shuffle=False)
```

You can inspect the dataset:

```
ds = make_input_fn(dfttrain, y_train, batch_size=10)()
for feature_batch, label_batch in ds.take(1):
    print('Some feature keys:', list(feature_batch.keys()))
    print()
    print('A batch of class:', feature_batch['class'].numpy())
    print()
    print('A batch of Labels:', label_batch.numpy())
```

Some feature keys: ['sex', 'age', 'n_siblings_spouses', 'parch', 'fare', 'class', 'deck', 'embark_town', 'alone']

A batch of class: [b'Third' b'Second' b'Third' b'Third' b'Second' b'Second' b'Second' b'Third' b'Second' b'Second']

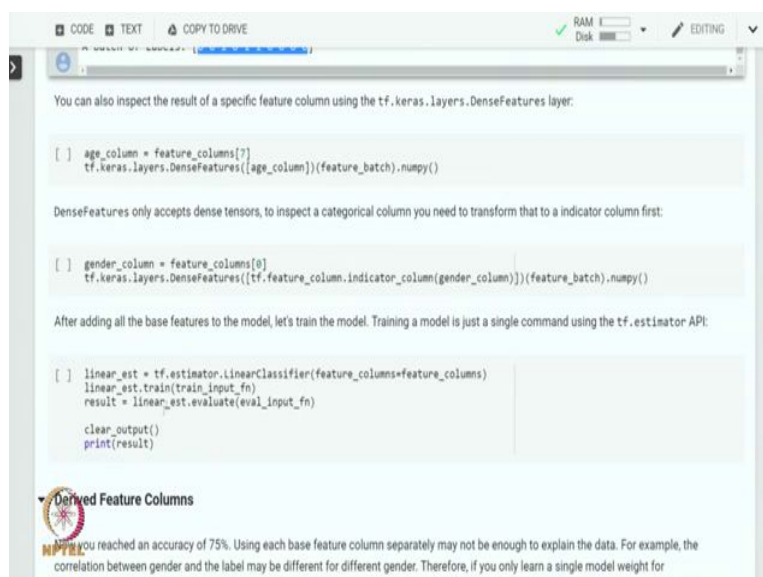
A batch of Labels: [0 0 1 0 1 1 0 0 0 0]

You can also inspect the result of a specific feature column using the `tf.keras.layers.DenseFeatures` layer:

```
age_column = feature_columns[7]
tf.keras.layers.DenseFeatures([age_column])(feature_batch).numpy()
```

You can see that the dataset has feature keys which are the same as the original dataset that we explored and we can also see the first batch for the attribute class and you can see values like third, second, third and so on. There are 10 values in a batch because we specified a batch size of 10. And in the same manner, we can see that for a label batch also there are 10 values that are present.

(Refer Slide Time: 05:50)



```
age_column = feature_columns[7]
tf.keras.layers.DenseFeatures([age_column])(feature_batch).numpy()
```

DenseFeatures only accepts dense tensors, to inspect a categorical column you need to transform that to an indicator column first:

```
gender_column = feature_columns[0]
tf.keras.layers.DenseFeatures([tf.feature_column.indicator_column(gender_column)])(feature_batch).numpy()
```

After adding all the base features to the model, let's train the model. Training a model is just a single command using the `tf.estimator` API:

```
linear_est = tf.estimator.LinearClassifier(feature_columns=feature_columns)
linear_est.train(train_input_fn)
result = linear_est.evaluate(eval_input_fn)

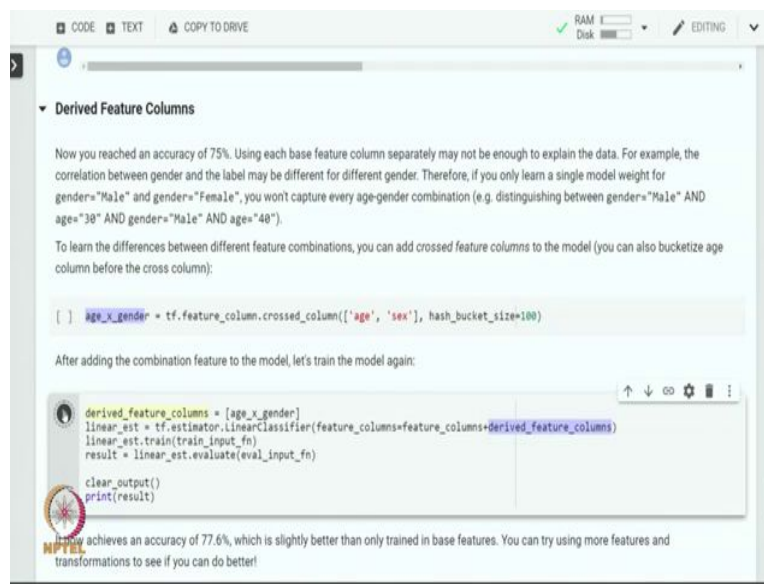
clear_output()
print(result)
```

Derived Feature Columns

When you reached an accuracy of 75%. Using each base feature column separately may not be enough to explain the data. For example, the correlation between gender and the label may be different for different gender. Therefore, if you only learn a single model weight for

Let us define our logistic regression classifier using `tf.estimator.LinearClassifier` command. It takes feature columns as its argument, then we train the model by giving `train_input_fn` as an argument and we evaluate the model using `eval_input_function` as an argument. We get we store the result in the result variable and we will print it at the end of the execution.

(Refer Slide Time: 06:28)



```
CODE TEXT COPY TO DRIVE RAM Disk EDITING
```

▼ Derived Feature Columns

Now you reached an accuracy of 75%. Using each base feature column separately may not be enough to explain the data. For example, the correlation between gender and the label may be different for different gender. Therefore, if you only learn a single model weight for gender="Male" and gender="Female", you won't capture every age-gender combination (e.g. distinguishing between gender="Male" AND age="38" AND gender="Male" AND age="48").

To learn the differences between different feature combinations, you can add crossed feature columns to the model (you can also bucketize age column before the cross column):

```
[ ] age_x_gender = tf.feature_column.crossed_column(['age', 'sex'], hash_bucket_size=100)
```

After adding the combination feature to the model, let's train the model again:

```
derived_feature_columns = [age_x_gender]
linear_est = tf.estimator.LinearClassifier(feature_columns=feature_columns+derived_feature_columns)
linear_est.train(train_input_fn)
result = linear_est.evaluate(eval_input_fn)

clear_output()
print(result)
```

achieves an accuracy of 77.6%, which is slightly better than only trained in base features. You can try using more features and transformations to see if you can do better!

You can see that we got accuracy of 76 percent on the evaluation data. The baseline accuracy was 62 percent and there are a bunch of other metrics that are also printed. Now, we have reached accuracy of 75 percent. Now, what we will do is we will try to include even more features and see whether we can get a better accuracy. So, what we will do is we will combine different features.

One of the feature combination that we will try is between age and gender. So, we construct a crossed column between age and gender and we set the hash bucket size to be 100. I would like to remind you about crossed column. So, whenever we cross whenever we construct a crossed column, it uses hashing for storing the values in order to avoid the problem of sparsity. Here we specify the hash bucket size to be 100.

Let us add the combined feature in the model along with the earlier feature columns. We can construct a derived feature column with all the crossed features, but for now, there is only a single crossed feature. And, we specify feature columns to be original feature columns and

derived feature columns and that is how we instantiate a linear classifier and then we train it and evaluate it as earlier.

(Refer Slide Time: 08:24)



```
[21] age_x_gender = tf.feature_column.crossed_column(['age', 'sex'], hash_bucket_size=100)

After adding the combination feature to the model, let's train the model again:

derived_feature_columns = [age_x_gender]
linear_est = tf.estimator.LinearClassifier(feature_columns=feature_columns+derived_feature_columns)
linear_est.train(train_input_fn)
result = linear_est.evaluate(eval_input_fn)

clear_output()
print(result)

{'accuracy': 0.7613636, 'accuracy_baseline': 0.625, 'auc': 0.84631157, 'auc_precision_recall': 0.7991083, 'average...
```

It now achieves an accuracy of 77.6%, which is slightly better than only trained in base features. You can try using more features and transformations to see if you can do better!

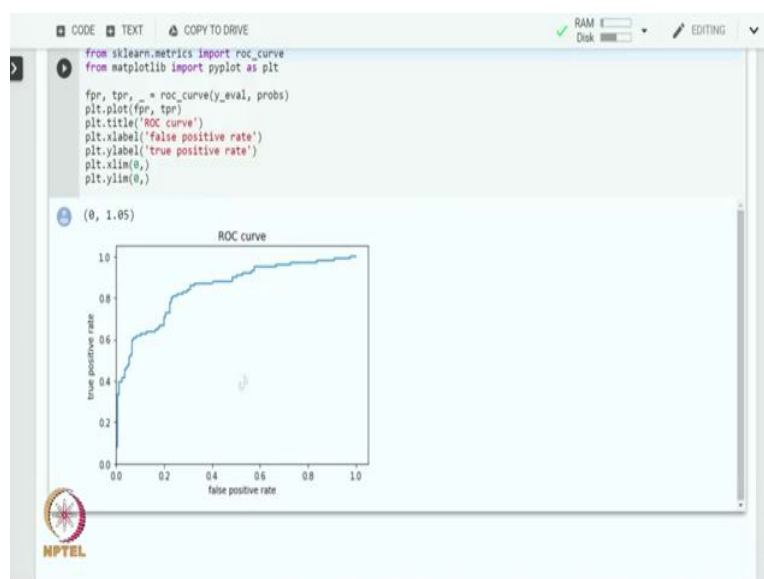
Now you can use the train model to make predictions on a passenger from the evaluation set. TensorFlow models are optimized to make predictions on a batch, or collection, of examples at once. Earlier, the eval_input_fn was defined using the entire evaluation set.

```
[ ] pred_dicts = list(linear_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')
```

NOTE: look at the receiver operating characteristic (ROC) of the results, which will give us a better idea of the tradeoff between the true positive rate and false positive rate.

We have achieved an accuracy of 76 percent which was which is 1 percentage point higher than the earlier model.

(Refer Slide Time: 08:38)



Finally let us look at the ROC curve to get an idea about the tradeoff between true positive rate and false positive rate. So, this is the ROC curve that we got. It is a reasonable ROC curve for a classification task. So, in this session we studied how to build logistic regression classifier with `tf.estimator` API. We use titanic dataset as an example dataset to predict the probability of survival of a passenger using a logistic regression model.