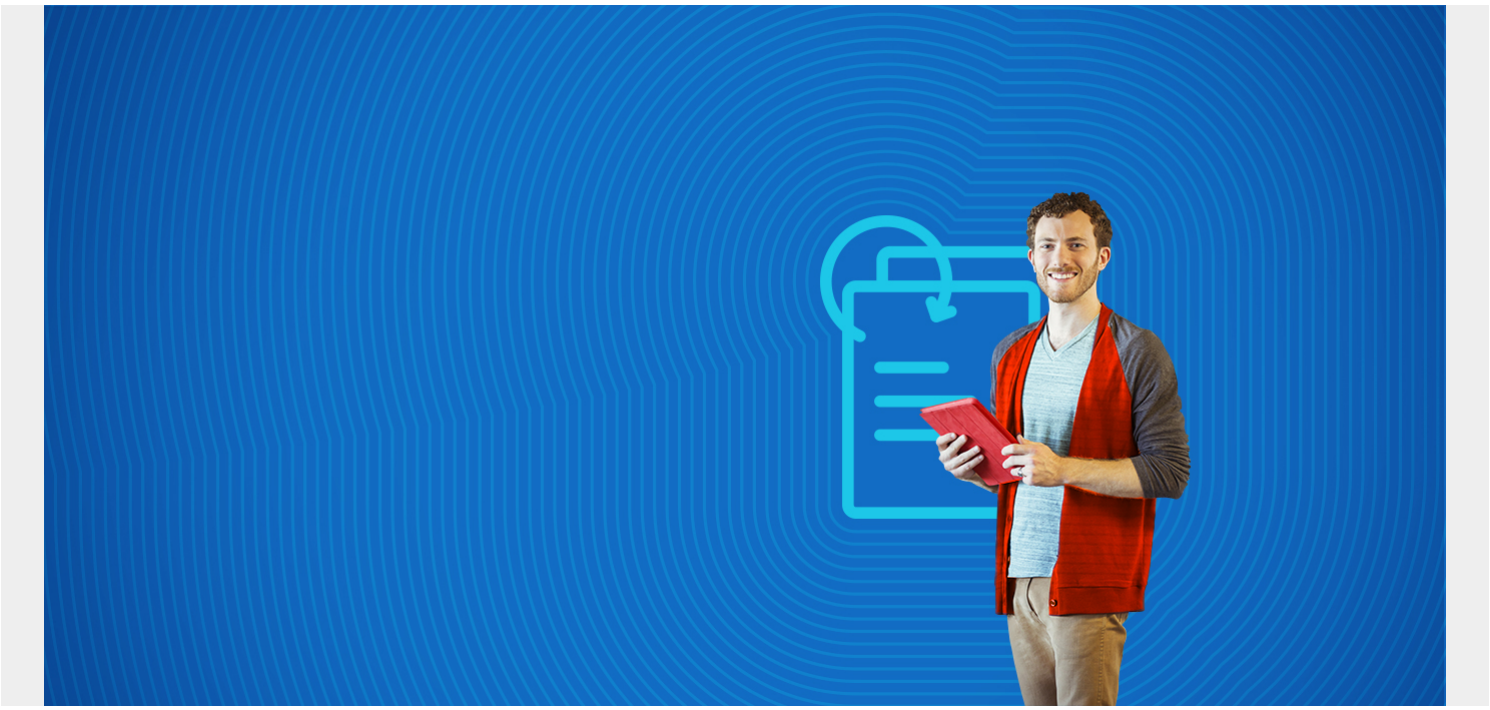
USING TENSORFLOW NEURAL NETWORK FOR MACHINE LEARNING PREDICTIONS WITH TRIPADVISOR DATA



Here is the last part of our analysis of the Tripadvisor data. Part one is here. In order to understand this, you will need to know Python and Numpy Arrays and the basics behind tensorflow and neural networks. If you do not, you can read an introduction to tensorflow here.

The code from this example is here and input data here. We create a neural network using the Tensorflow tf.estimator.DNNClassifier. (DNN means deep neural network, i.e., one with hidden layers between the input and output layers.)

Below we discuss each section of the code.

(This tutorial is part of our Guide to Machine Learning with TensorFlow & Keras. Use the right-hand menu to navigate.)

parse_line

feature_names is the name we have assigned to the feature columns.

FIELD_DEFAULTS is an array of 20 integers. This tells tensorflow that our inputs are integers and that there are 20 features. If we had used 1.0 it would declare those as floats.

```
import tensorflow as tf
import numpy as np
feature_names =
FIELD_DEFAULTS = , , , , ,
, , , , ,
```

```
, , , , ,
, , , , ]
```

parse_line

DNNClassifier.train requires an **input_fn** that returns features and labels. It is not supposed to be called with arguments, so we use **lambda** below to iteratively call it and to pass it a parameter, which is the name of the text file to read..

We cannot simply use one of the examples provided by TensorFlow, such as the helloword-type one that reads Iris flower data, to read the data. We made our own data and put it into a .csv file. So we need our own parser. So, in this case, we use the tf.data.TextLineDataset method to read from the csv text file and feed it into this parser. That will read those lines and return the features and labels as a dictionary and tensor pair.

In **del parsed_line** we deleted the 5th tensor from the input, which is the Tripadvisor score. Because that is an label (i.e., output) and not a feature (input).

tf.decode_csv(line, FIELD_DEFAULTS) creates tensors for each items read from the .csv file.

You cannot see tensors using they have value. And they do not have value until you run a tensor session. But you can inspect these values using **tp.Print().** Note also that for debug purposes you could do this to test the parse functions:

```
import pandas as pd
df = pd.read_csv("/home/walker/TripAdvisor.csv")
ds = df.map(parse_line)
```

Continuing with our explanation, **dict(zip(feature_names, features))** create a dictionary from the features tensors and features name. For the label we just assign that **label = parsed_line** from the 5th item in **parsed_line.**

```
def parse_line(line):
parsed_line = tf.decode_csv(line, FIELD_DEFAULTS)
tf.Print(input_=parsed_line , data=, message="parsed_line ")
tf.Print(input_=parsed_line, data=], message="score")
label = parsed_line
del parsed_line
features = parsed_line
d = dict(zip(feature_names, features))
return d, label
```

csv_input

A dataset is a Tensorflow dataset and not a simpler Python object. We call **parse_line** with the **dataset.map()** method after having created the dataset from the .csv text file with **tf.data.TextLineDataset(csv_path).**

```
def csv_input_fn(csv_path, batch_size):
dataset = tf.data.TextLineDataset(csv_path)
dataset = dataset.map(parse_line)
dataset = dataset.shuffle(1000).repeat().batch(batch_size)
return dataset
```

Create Tensors

Here we create the tensors as continuous numbers as opposed to categorical. This is correct but could be improved. See the note below.

Note: User country, is a set of discrete values. So we could have used, for example, **Usercountry = tf.feature_column.indicator_column(tf.feature_column.categorical_column_with_identity("Usercountry",47))** since there are 47 countries in our dataset. You can experiment with that and see if you can make that change. I got errors trying to get that to work since tf.decode_csv() appeared to be reading the wrong column in certain cases this given values that were, for example, not one of the 47 countries. So there must be a few rows in the input data that has a different number of commas than the others. You can experiment with that.

Finally **feature_columns** is an array of the tensors we have created.

```
Usercountry = tf.feature_column.numeric_column("Usercountry")
Nrreviews = tf.feature_column.numeric_column("Nrreviews")
Nrhotelreviews = tf.feature_column.numeric_column("Nrhotelreviews")
Helpfulvotes = tf.feature_column.numeric_column("Helpfulvotes")
Periodofstay = tf.feature_column.numeric_column("Periodofstay")
Travelertype = tf.feature_column.numeric_column("Travelertype")
Pool = tf.feature_column.numeric_column("Pool")
Gym = tf.feature_column.numeric_column("Gym")
Tenniscourt = tf.feature_column.numeric_column("Tenniscourt")
Spa = tf.feature_column.numeric_column("Spa")
Casino = tf.feature_column.numeric_column("Casino")
Freeinternet = tf.feature_column.numeric_column("Freeinternet")
Hotelname = tf.feature_column.numeric_column("Hotelname")
Hotelstars = tf.feature_column.numeric_column("Hotelstars")
Nrrooms = tf.feature_column.numeric_column("Nrrooms")
Usercontinent = tf.feature_column.numeric_column("Usercontinent")
Memberyears = tf.feature_column.numeric_column("Memberyears")
Reviewmonth = tf.feature_column.numeric_column("Reviewmonth")
Reviewweekday = tf.feature_column.numeric_column("Reviewweekday")
feature_columns =
```

Create Classifier

Now we train the model. The **hidden_units** means the first hidden layer of the deep neural network has 10 nodes and the second has 10. The **model_dir** is the temporary folder where to store the trained model. The hotel scores range from 1 to 5 so **n_classes** is 6 since it must be greater than that number of buckets.

```
classifier=tf.estimator.DNNClassifier(
feature_columns=feature_columns,
hidden_units=,
```

```
n_classes=6,
model_dir="/tmp")
batch_size = 100
```

Train the model

Now we train the model. We use lambda because the documentation says "Estimators expect an input_fn to take no arguments. To work around this restriction, we use lambda to capture the arguments and provide the expected interface."

```
classifier.train(
steps=100,
input_fn=lambda : csv_input_fn("/home/walker/tripAdvisorFL.csv", batch_size))
```

Make a Prediction

Now we make a prediction on the trained model. In practice you should also run an evaluation step. You will see in the code on github that I wrote that, but it never exited the evaluation step. So that remains an open issue to sort out here.
We need some data to test with. To we have the first line from the training set input and key it in here. That reviewer gave the hotel a score of 5. So our expected result is 5. The neural network will give the probability that the expected result is 5. The **classifier.predict()** method runs the input function we tell it to run, in this case. **predict_input_fn().** It that returns the features as a dictionary. If we had been using running the evaluation we would need both the features and the label.

```
features = {'Usercountry': np.array(), 'Nrreviews':
np.array(),'Nrhotelreviews': np.array(),'Helpfulvotes':
np.array(),'Periodofstay': np.array(),'Travelertype': np.array(),'Pool' :
np.array(),'Gym' : np.array(),'Tenniscourt' : np.array(),'Spa' :
np.array(),'Casino' : np.array(),'Freeinternet' : np.array(),'Hotelname' :
np.array(),'Hotelstars' : np.array(),'Nrrooms' : np.array(),'Usercontinent' :
np.array(),'Memberyears' : np.array(),'Reviewmonth' :
np.array(),'Reviewweekday' : np.array()}
def predict_input_fn():
return features
expected =
prediction = classifier.predict(input_fn=predict_input_fn)
for pred_dict, expec in zip(prediction, expected):
class_id = pred_dict
probability = pred_dict
print ('class_ids=', class_id, ' probabilities=',  probability)
```

We then print the results. The probability of a 5 is in this example is 38%. We would hope to get something close to, say, 90%. This could be an outlier value. We do not know since he **have** yet to evaluation the model.

Obviously we need to go back and evaluation the model and try again with additional data. One would think that hotel scores are indeed correlated with the Tripadvisor data that we have given it. But the focus here is just to get the model to work. Now we need to fine tune in and see if another

ML model might be more appropriate.

```
class_ids= 5  probabilities= 0.38341486
```

Addendum

You can try these to make the discrete value columns as mentioned above:

```
Usercountry = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Usercountry",47))
Nrreviews = tf.feature_column.numeric_column("Nrreviews")
Nrhotelreviews = tf.feature_column.numeric_column("Nrhotelreviews")
Helpfulvotes = tf.feature_column.numeric_column("Helpfulvotes")
Periodofstay = tf.feature_column.numeric_column("Periodofstay")
Travelertype = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Travelertype",5))
Pool = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Pool",2))
Gym = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Gym",2))
Tenniscourt = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Tenniscourt",2))
Spa = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Spa",2))
Casino = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Casino",2))
Freeinternet = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Freeinternet",2))
Hotelname = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Hotelname",22))
Hotelstars = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Hotelstars",5))
Nrrooms = tf.feature_column.numeric_column("Nrrooms")
Usercontinent = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Usercontinent",6))
Memberyears = tf.feature_column.numeric_column("Memberyears")
Reviewmonth = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Reviewmonth",12))
Reviewweekday = tf.feature_column.indicator_column(tf.feature_column.
categorical_column_with_identity("Reviewweekday",7))
```