Tensorflow Linear Model Tutorial

Task:

Given census data about a person such as age, gender, education and occupation, we will try to predict whether or not the person earns more than 50,000 dollars a year. We will train a **logistic regression** model, and given an individual's information our model will output a number between 0 and 1, which can be interpreted as the probability that the individual has an annual income of over 50,000 dollars.

The Data

The dataset we'll be using is the Census Income Dataset. The dataset comprise of both features like age, gender, workclass, education, occupation, native country and income bracket. Out of these features some are Categorical and some are Continuous.

B	C	D	E	F	G	H	I	J	K	L	M	N	
age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	gender	capital_gain	capital_loss	hours_per_week	native_
39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-
50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-
38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-
53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-
28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba
37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-
49	Private	160187	9th	5	Married-spouse-abse	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica
52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45	United-
31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-family	White	Female	14084	0	50	United-
42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	5178	0	40	United-
37	Private	280464	Some-college	10	Married-civ-spouse	Exec-managerial	Husband	Black	Male	0	0	80	United-
30	State-gov	141297	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Isl	Male	0	0	40	India
23	Private	122272	Bachelors	13	Never-married	Adm-clerical	Own-child	White	Female	0	0	30	United-
32	Private	205019	Assoc-acdm	12	Never-married	Sales	Not-in-family	Black	Male	0	0	50	United-
40	Private	121772	Assoc-voc	11	Married-civ-spouse	Craft-repair	Husband	Asian-Pac-Isl	Male	0	0	40	?
34	Private	245487	7th-8th	4	Married-civ-spouse	Transport-moving	Husband	Amer-Indian-	Male	0	0	45	Mexico
25	Self-emp-not-inc	176756	HS-grad	9	Never-married	Farming-fishing	Own-child	White	Male	0	0	35	United-
32	Private	186824	HS-grad	9	Never-married	Machine-op-inspct	Unmarried	White	Male	0	0	40	United-
38	Private	28887	11th	7	Married-civ-spouse	Sales	Husband	White	Male	0	0	50	United-
43	Self-emp-not-inc	292175	Masters	14	Divorced	Exec-managerial	Unmarried	White	Female	0	0	45	United-
40	Private	193524	Doctorate	16	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	60	United-
54	Private	302146	HS-grad	9	Separated	Other-service	Unmarried	Black	Female	0	0	20	United-
35	Federal-gov	76845	9th	5	Married-civ-spouse	Farming-fishing	Husband	Black	Male	0	0	40	United-
43	Private	117037	11th	7	Married-civ-spouse	Transport-moving	Husband	White	Male	0	2042	40	United-

Steps:

1. Downloading and reading the data

The dataset is downloaded from the given link and is loaded into Pandas data frames for further computations. The training data frame is named as df_train and testing data as df_test. As the given data is not labeled so I assigned labels to the dataset and also introduced a new column called label which can further be used as a target attribute. The value of label attribute will be 0 or 1 as it is a case of binary classification so only two classes are used, label 1 shows an income bracket of more than 50k whereas label 0 shows an income bracket of less than or equals to 50k.

```
# Downloading Data
train_file = tempfile.NamedTemporaryFile(delete=False)
test_file = tempfile.NamedTemporaryFile(delete=False)
urllib.request.urlretrieve("http://mlr.cs.umass.edu/ml/machine-learning-databases/adult/adult.data", train_file.name)
train_file_name = train_file.name
urllib.request.urlretrieve("http://mlr.cs.umass.edu/ml/machine-learning-databases/adult/adult.test", test_file.name)
test_file_name = test_file.name


COLUMNS = ["age", "workclass", "fnlwgt", "education", "education_num",
           "marital_status", "occupation", "relationship", "race", "gender",
           "capital_gain", "capital_loss", "hours_per_week", "native_country",
           "income_bracket"]
df_train = pd.read_csv(train_file, names=COLUMNS, skipinitialspace=True)
df_test = pd.read_csv(test_file, names=COLUMNS, skipinitialspace=True, skiprows=1)

LABEL_COLUMN = "label"
df_train[LABEL_COLUMN] = (df_train["income_bracket"].apply(lambda x: ">50K" in x)).astype(int)
df_test[LABEL_COLUMN] = (df_test["income_bracket"].apply(lambda x: ">50K" in x)).astype(int)
```

2. Converting the data into Tensors

As in Tensorflow the computations are done over tensors so the data is converted into tensors. The values of Continuous columns are converted into constant tensors, meaning that the tensor represents a constant value, which in general is a good format to represent dense data. Whereas the values of Categorical columns are converted into sparse tensors which is a good format to represent sparse data. This is to be performed over both the training and testing data so it is done inside a function i.e. input_fn which returns feature columns and labels for the given data frame.

```
def input_fn(df):
  # Creates a dictionary mapping from each continuous feature column name (k) to
  # the values of that column stored in a constant Tensor.
  continuous_cols = {k: tf.constant(df[k].values)
                     for k in CONTINUOUS_COLUMNS}
  # Creates a dictionary mapping from each categorical feature column name (k)
  # to the values of that column stored in a tf.SparseTensor.
  categorical_cols = {k: tf.SparseTensor(
      indices=[[i, 0] for i in range(df[k].size)],
      values=df[k].values,
      dense_shape=[df[k].size, 1])
                      for k in CATEGORICAL_COLUMNS}
  # Merges the two dictionaries into one.
  feature_cols = dict(continuous_cols)
  feature_cols.update(categorical_cols)
  # Converts the label column into a constant Tensor.
  label = tf.constant(df[LABEL_COLUMN].values)
  # Returns the feature columns and the label.
  return feature_cols, label
```

3. Selecting features for the Model

Now each column is converted into tensors according to their nature i.e. for columns like gender where there are only two possible values male and female we will use **Sparse column with keys** and for the columns like education whose possible values are not known in advance we will use **Sparse column with hash bucket** in which each possible value will be hashed to an integer id. This is done for every Categorical column of the dataset. For Continuous columns we uses **Real valued column.**

```
# Sparse base columns(Categorical Columns).
gender = tf.contrib.layers.sparse_column_with_keys(column_name="gender",keys=["female", "male"])
education = tf.contrib.layers.sparse_column_with_hash_bucket("education", hash_bucket_size=1000)
race = tf.contrib.layers.sparse_column_with_hash_bucket("race", hash_bucket_size=100)
marital_status = tf.contrib.layers.sparse_column_with_hash_bucket("marital_status", hash_bucket_size=100)
relationship = tf.contrib.layers.sparse_column_with_hash_bucket("relationship", hash_bucket_size=100)
workclass = tf.contrib.layers.sparse_column_with_hash_bucket("workclass", hash_bucket_size=100)
occupation = tf.contrib.layers.sparse_column_with_hash_bucket("occupation", hash_bucket_size=1000)
native_country = tf.contrib.layers.sparse_column_with_hash_bucket("native_country", hash_bucket_size=1000)

# Continuous base columns.
age = tf.contrib.layers.real_valued_column("age")
education_num = tf.contrib.layers.real_valued_column("education_num")
capital_gain = tf.contrib.layers.real_valued_column("capital_gain")
capital_loss = tf.contrib.layers.real_valued_column("capital_loss")
hours_per_week = tf.contrib.layers.real_valued_column("hours_per_week")
```

4. Engineering features for the Model

Making Continuous Features Categorical through Bucketization:

Sometimes the relationship between a continuous feature and the label is not linear so to learn the fine-grained correlation between the label and each value group separately, we use bucketization. Bucketization is a process of dividing the entire range of a continuous feature into a set of consecutive bins/buckets, and then converting the original numerical feature into a bucket ID (as a categorical feature) depending on which bucket that value falls into. In this model I have converted the age column into age buckets.

Intersecting Multiple Columns with Crossed Column:

As using base features are not enough to capture every single combination of features so to learn the differences between different feature combinations, we can add crossed feature columns to the model. In this model I have used three crossed columns i.e. education_x_occupation, age_bucket_x_education_x_occupation and native_country_x_occupation.

All of these selected and created features are used to train our model.

```
# Bucketization of age attribute.
age_buckets = tf.contrib.layers.bucketized_column(age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])

# Crossed Feature columns.
education_x_occupation=tf.contrib.layers.crossed_column([education, occupation],hash_bucket_size=int(1e4))
age_bucket_x_education_x_occupation=tf.contrib.layers.crossed_column([age_buckets, education, occupation],
                                                                     hash_bucket_size=int(1e6))
native_country_x_occupation=tf.contrib.layers.crossed_column([native_country, occupation],hash_bucket_size=int(1e4))

# Feature Columns
wide_columns = [gender, native_country, education, occupation, workclass, relationship,
                age_buckets,education_x_occupation,
                age_bucket_x_education_x_occupation,native_country_x_occupation]
```

5. Defining the Logistic Regression Model

The model is trained using the training data and then it is evaluated using the testing data, the model used here is **Linear Classifier** which takes feature columns and model directory as its input parameters. After training phase the model is evaluated against testing data and the results of this evaluations are shown.

```
# Model Defination
model_dir = tempfile.mkdtemp()
m = tf.contrib.learn.LinearClassifier(model_dir=model_dir,feature_columns=wide_columns)
m.fit(input_fn=train_input_fn, steps=200)
results = m.evaluate(input_fn=eval_input_fn, steps=1)

for key in sorted(results):
    print("%s: %s" % (key, results[key]))
```

6. Results

The Accuracy of the trained model is **83.44%** and all other parameters can be seen in the results.

```
accuracy: 0.834408
accuracy/baseline_label_mean: 0.236226
accuracy/threshold_0.500000_mean: 0.834408
auc: 0.877227
auc_precision_recall: 0.682382
global_step: 200
labels/actual_label_mean: 0.236226
labels/prediction_mean: 0.242684
loss: 0.361987
precision/positive_threshold_0.500000_mean: 0.710469
recall/positive_threshold_0.500000_mean: 0.50468
```