COSC4337_109-Classification-Exercise-Solution

September 17, 2021

1 Classification Exercise - Solutions

We'll be working with some California Census Data, we'll be trying to use various features of an individual to predict what class of income they belogn in (>50k or <=50k).

Here is some information about the data:

Column Name

Type

Description

age

Continuous

The age of the individual

workclass

Categorical

The type of employer the individual has (government, military, private, etc.).

fnlwgt

Continuous

The number of people the census takers believe that observation represents (sample weight). This variable will not be used.

education

Categorical

The highest level of education achieved for that individual.

education_num

Continuous

The highest level of education in numerical form.

marital_status

Categorical

Marital status of the individual.

occupation

Categorical

The occupation of the individual.

relationship

Categorical

Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race

Categorical

White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

gender

Categorical

Female, Male.

capital_gain

Continuous

Capital gains recorded.

capital_loss

Continuous

Capital Losses recorded.

hours_per_week

Continuous

Hours worked per week.

native_country

Categorical

Country of origin of the individual.

income

Categorical

">50K" or "<=50K", meaning whether the person makes more than $50,000 annually.

1.1 Follow the Directions in Bold. If you get stuck, check out the solutions lecture.

1.1.1 THE DATA

** Read in the census_data.csv data with pandas**

[1]:
```
import pandas as pd
```

[2]:
```
census = pd.read_csv("census_data.csv")
```

[3]:
```
census.head()
```

[3]:

	age	workclass	education	education_num	marital_status	occupation	relationship	race	gender	capital_gain	capital_loss	hours_per_week	native_country	income_bracket
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

** TensorFlow won't be able to understand strings as labels, you'll need to use pandas .apply() method to apply a custom function that converts them to 0s and 1s. This might be hard if you aren't very familiar with pandas, so feel free to take a peek at the solutions for this part.**

** Convert the Label column to 0s and 1s instead of strings.**

[4]:
```
census['income_bracket'].unique()
```

[4]:
```
array([' <=50K', ' >50K'], dtype=object)
```

[5]:
```
def label_fix(label):
    if label==' <=50K':
        return 0
    else:
        return 1
```

[6]:
```
census['income_bracket'] = census['income_bracket'].apply(label_fix)
```

[7]:
```
# Cool Alternative
# lambda label:int(label==' <=50k')

# census['income_bracket'].apply(lambda label: int(label==' <=50K'))
```

1.1.2 Perform a Train Test Split on the Data

[8]:
```
from sklearn.model_selection import train_test_split
```

[9]:
```
x_data = census.drop('income_bracket',axis=1)
y_labels = census['income_bracket']
X_train, X_test, y_train, y_test = train_test_split(x_data,y_labels,test_size=0.
 ↪3,random_state=101)
```

1.1.3 Create the Feature Columns for tf.esitmator

** Take note of categorical vs continuous values! **

[10]:
```
census.columns
```

[10]:
```
Index(['age', 'workclass', 'education', 'education_num', 'marital_status',
       'occupation', 'relationship', 'race', 'gender', 'capital_gain',
       'capital_loss', 'hours_per_week', 'native_country', 'income_bracket'],
      dtype='object')
```

** Import Tensorflow **

[11]:
```
import tensorflow as tf
```

** Create the tf.feature_columns for the categorical values. Use vocabulary lists or just use hash buckets. **

[12]:
```
gender = tf.feature_column.categorical_column_with_vocabulary_list("gender",␣
 ↪["Female", "Male"])
occupation = tf.feature_column.
 ↪categorical_column_with_hash_bucket("occupation", hash_bucket_size=1000)
marital_status = tf.feature_column.
 ↪categorical_column_with_hash_bucket("marital_status", hash_bucket_size=1000)
relationship = tf.feature_column.
 ↪categorical_column_with_hash_bucket("relationship", hash_bucket_size=1000)
education = tf.feature_column.categorical_column_with_hash_bucket("education",␣
 ↪hash_bucket_size=1000)
workclass = tf.feature_column.categorical_column_with_hash_bucket("workclass",␣
 ↪hash_bucket_size=1000)
native_country = tf.feature_column.
 ↪categorical_column_with_hash_bucket("native_country", hash_bucket_size=1000)
```

** Create the continuous feature_columns for the continuous values using numeric_column **

[13]:
```
age = tf.feature_column.numeric_column("age")
education_num = tf.feature_column.numeric_column("education_num")
capital_gain = tf.feature_column.numeric_column("capital_gain")
capital_loss = tf.feature_column.numeric_column("capital_loss")
hours_per_week = tf.feature_column.numeric_column("hours_per_week")
```

** Put all these variables into a single list with the variable name feat_cols **

[14]:
```
feat_cols =␣
 ↪[gender,occupation,marital_status,relationship,education,workclass,native_country,
            age,education_num,capital_gain,capital_loss,hours_per_week]
```

1.1.4 Create Input Function

** Batch_size is up to you. But do make sure to shuffle!**

[15]:
```
input_func=tf.estimator.inputs.
 ↪pandas_input_fn(x=X_train,y=y_train,batch_size=100,num_epochs=None,shuffle=True)
```

Create your model with tf.estimator Create a LinearClassifier.(If you want to use a DNNClassifier, keep in mind you'll need to create embedded columns out of the cateogrical feature that use strings, check out the previous lecture on this for more info.)

[16]:
```
model = tf.estimator.LinearClassifier(feature_columns=feat_cols)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory:
C:\Users\RizkN\AppData\Local\Temp\tmpc4n5bi1j
INFO:tensorflow:Using config: {'_model_dir':
'C:\\Users\\RizkN\\AppData\\Local\\Temp\\tmpc4n5bi1j', '_tf_random_seed': None,
'_save_summary_steps': 100, '_save_checkpoints_steps': None,
'_save_checkpoints_secs': 600, '_session_config': allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None,
'_protocol': None, '_eval_distribute': None, '_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x0000015A54FF6848>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '',
'_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

** Train your model on the data, for at least 5000 steps. **

[17]:
```
model.train(input_fn=input_func,steps=5000)
```

```
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\python\training\training_util.py:236:
Variable.initialized_value (from tensorflow.python.ops.variables) is deprecated
```

```
and will be removed in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in
eager and graph (inside tf.defun) contexts.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-packages\tensorf
low_estimator\python\estimator\inputs\queues\feeding_queue_runner.py:62:
QueueRunner.__init__ (from tensorflow.python.training.queue_runner_impl) is
deprecated and will be removed in a future version.
Instructions for updating:
To construct input pipelines, use the `tf.data` module.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-packages\tensorf
low_estimator\python\estimator\inputs\queues\feeding_functions.py:500:
add_queue_runner (from tensorflow.python.training.queue_runner_impl) is
deprecated and will be removed in a future version.
Instructions for updating:
To construct input pipelines, use the `tf.data` module.
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\python\feature_column\feature_column_v2.py:305:
Layer.add_variable (from tensorflow.python.keras.engine.base_layer) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use `layer.add_weight` method instead.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\python\ops\resource_variable_ops.py:1630: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\python\ops\embedding_ops.py:802: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_estimator\python\estimator\canned\linear.py:308: to_float
(from tensorflow.python.ops.math_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Use `tf.cast` instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\python\training\monitored_session.py:882:
```

```
start_queue_runners (from tensorflow.python.training.queue_runner_impl) is
deprecated and will be removed in a future version.
Instructions for updating:
To construct input pipelines, use the `tf.data` module.
INFO:tensorflow:Saving checkpoints for 0 into
C:\Users\RizkN\AppData\Local\Temp\tmpc4n5bi1j\model.ckpt.
INFO:tensorflow:loss = 69.31474, step = 1
INFO:tensorflow:global_step/sec: 198.15
INFO:tensorflow:loss = 2771.567, step = 101 (0.514 sec)
INFO:tensorflow:global_step/sec: 232.797
INFO:tensorflow:loss = 23.679995, step = 201 (0.431 sec)
INFO:tensorflow:global_step/sec: 211.435
INFO:tensorflow:loss = 172.98154, step = 301 (0.469 sec)
INFO:tensorflow:global_step/sec: 211.865
INFO:tensorflow:loss = 754.4713, step = 401 (0.474 sec)
INFO:tensorflow:global_step/sec: 218.318
INFO:tensorflow:loss = 40.32142, step = 501 (0.457 sec)
INFO:tensorflow:global_step/sec: 201.596
INFO:tensorflow:loss = 231.27289, step = 601 (0.498 sec)
INFO:tensorflow:global_step/sec: 179.855
INFO:tensorflow:loss = 262.46594, step = 701 (0.555 sec)
INFO:tensorflow:global_step/sec: 169.935
INFO:tensorflow:loss = 264.06906, step = 801 (0.592 sec)
INFO:tensorflow:global_step/sec: 193.182
INFO:tensorflow:loss = 190.95473, step = 901 (0.514 sec)
INFO:tensorflow:global_step/sec: 221.709
INFO:tensorflow:loss = 40.812557, step = 1001 (0.449 sec)
INFO:tensorflow:global_step/sec: 197.527
INFO:tensorflow:loss = 36.24775, step = 1101 (0.506 sec)
INFO:tensorflow:global_step/sec: 278.916
INFO:tensorflow:loss = 148.35156, step = 1201 (0.361 sec)
INFO:tensorflow:global_step/sec: 259.056
INFO:tensorflow:loss = 311.86063, step = 1301 (0.384 sec)
INFO:tensorflow:global_step/sec: 247.239
INFO:tensorflow:loss = 210.51053, step = 1401 (0.407 sec)
INFO:tensorflow:global_step/sec: 265.578
INFO:tensorflow:loss = 33.848213, step = 1501 (0.376 sec)
INFO:tensorflow:global_step/sec: 257.067
INFO:tensorflow:loss = 38.28836, step = 1601 (0.386 sec)
INFO:tensorflow:global_step/sec: 268.598
INFO:tensorflow:loss = 40.716667, step = 1701 (0.372 sec)
INFO:tensorflow:global_step/sec: 281.689
INFO:tensorflow:loss = 35.435165, step = 1801 (0.357 sec)
INFO:tensorflow:global_step/sec: 257.729
INFO:tensorflow:loss = 67.39629, step = 1901 (0.388 sec)
INFO:tensorflow:global_step/sec: 210.524
INFO:tensorflow:loss = 66.776535, step = 2001 (0.477 sec)
INFO:tensorflow:global_step/sec: 232.043
```

```
INFO:tensorflow:loss = 34.25927, step = 2101 (0.429 sec)
INFO:tensorflow:global_step/sec: 280.299
INFO:tensorflow:loss = 37.18171, step = 2201 (0.358 sec)
INFO:tensorflow:global_step/sec: 268.089
INFO:tensorflow:loss = 58.846634, step = 2301 (0.370 sec)
INFO:tensorflow:global_step/sec: 223.213
INFO:tensorflow:loss = 39.074253, step = 2401 (0.451 sec)
INFO:tensorflow:global_step/sec: 223.689
INFO:tensorflow:loss = 65.20474, step = 2501 (0.447 sec)
INFO:tensorflow:global_step/sec: 205.317
INFO:tensorflow:loss = 403.8542, step = 2601 (0.489 sec)
INFO:tensorflow:global_step/sec: 240.57
INFO:tensorflow:loss = 208.06223, step = 2701 (0.414 sec)
INFO:tensorflow:global_step/sec: 244.498
INFO:tensorflow:loss = 33.415634, step = 2801 (0.408 sec)
INFO:tensorflow:global_step/sec: 271.003
INFO:tensorflow:loss = 30.884544, step = 2901 (0.370 sec)
INFO:tensorflow:global_step/sec: 271.738
INFO:tensorflow:loss = 41.776295, step = 3001 (0.364 sec)
INFO:tensorflow:global_step/sec: 260.984
INFO:tensorflow:loss = 44.12464, step = 3101 (0.384 sec)
INFO:tensorflow:global_step/sec: 218.563
INFO:tensorflow:loss = 37.060394, step = 3201 (0.456 sec)
INFO:tensorflow:global_step/sec: 241.52
INFO:tensorflow:loss = 92.91695, step = 3301 (0.421 sec)
INFO:tensorflow:global_step/sec: 225.288
INFO:tensorflow:loss = 67.31664, step = 3401 (0.437 sec)
INFO:tensorflow:global_step/sec: 222.692
INFO:tensorflow:loss = 104.52739, step = 3501 (0.451 sec)
INFO:tensorflow:global_step/sec: 251.032
INFO:tensorflow:loss = 82.66494, step = 3601 (0.400 sec)
INFO:tensorflow:global_step/sec: 226.245
INFO:tensorflow:loss = 87.70129, step = 3701 (0.443 sec)
INFO:tensorflow:global_step/sec: 222.195
INFO:tensorflow:loss = 185.81793, step = 3801 (0.447 sec)
INFO:tensorflow:global_step/sec: 239.813
INFO:tensorflow:loss = 69.91282, step = 3901 (0.417 sec)
INFO:tensorflow:global_step/sec: 255.104
INFO:tensorflow:loss = 33.424725, step = 4001 (0.391 sec)
INFO:tensorflow:global_step/sec: 207.262
INFO:tensorflow:loss = 250.0559, step = 4101 (0.486 sec)
INFO:tensorflow:global_step/sec: 272.411
INFO:tensorflow:loss = 185.93808, step = 4201 (0.367 sec)
INFO:tensorflow:global_step/sec: 251.254
INFO:tensorflow:loss = 27.12293, step = 4301 (0.397 sec)
INFO:tensorflow:global_step/sec: 268.714
INFO:tensorflow:loss = 90.04298, step = 4401 (0.373 sec)
INFO:tensorflow:global_step/sec: 267.052
```

```
INFO:tensorflow:loss = 59.009064, step = 4501 (0.374 sec)
INFO:tensorflow:global_step/sec: 217.623
INFO:tensorflow:loss = 96.62125, step = 4601 (0.455 sec)
INFO:tensorflow:global_step/sec: 237.141
INFO:tensorflow:loss = 66.35788, step = 4701 (0.430 sec)
INFO:tensorflow:global_step/sec: 204.107
INFO:tensorflow:loss = 69.71101, step = 4801 (0.486 sec)
INFO:tensorflow:global_step/sec: 218.113
INFO:tensorflow:loss = 76.9079, step = 4901 (0.463 sec)
INFO:tensorflow:Saving checkpoints for 5000 into
C:\Users\RizkN\AppData\Local\Temp\tmpc4n5bi1j\model.ckpt.
INFO:tensorflow:Loss for final step: 39.92393.
```

[17]:
```
<tensorflow_estimator.python.estimator.canned.linear.LinearClassifier at
0x15a54ff79c8>
```

1.1.5 Evaluation

** Create a prediction input function. Remember to only supprt X_test data and keep shuffle=False. **

[18]:
```
pred_fn = tf.estimator.inputs.
 ↪pandas_input_fn(x=X_test,batch_size=len(X_test),shuffle=False)
```

** Use model.predict() and pass in your input function. This will produce a generator of predictions, which you can then transform into a list, with list() **

[19]:
```
predictions = list(model.predict(input_fn=pred_fn))
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
C:\Users\RizkN\AppData\Local\Temp\tmpc4n5bi1j\model.ckpt-5000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

** Each item in your list will look like this: **

[20]:
```
predictions[0]
```

[20]:
```
{'logits': array([-0.91882783], dtype=float32),
 'logistic': array([0.2851968], dtype=float32),
 'probabilities': array([0.7148032, 0.2851968], dtype=float32),
 'class_ids': array([0], dtype=int64),
 'classes': array([b'0'], dtype=object),
 'all_class_ids': array([0, 1]),
 'all_classes': array([b'0', b'1'], dtype=object)}
```

** Create a list of only the class_ids key values from the prediction list of dictionaries, these are the predictions you will use to compare against the real y_test values. **

[21]:
```
final_preds = []
for pred in predictions:
    final_preds.append(pred['class_ids'][0])
```

[22]:
```
final_preds[:10]
```

[22]:
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

** Import classification_report from sklearn.metrics and then see if you can figure out how to use it to easily get a full report of your model's performance on the test data. **

[23]:
```
from sklearn.metrics import classification_report
```

[24]:
```
print(classification_report(y_test,final_preds))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      7436
           1       0.71      0.64      0.67      2333

    accuracy                           0.85      9769
   macro avg       0.80      0.78      0.79      9769
weighted avg       0.85      0.85      0.85      9769
```

2 Great Job!

[ ]:
```

```