Using_TensorFlow_with_Categorical_Features_to_Predict_Income_Bracke

May 13, 2024

1 Classification Exercise

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_bracket

Categorical

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

1.0.1 THE DATA

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

[2]:
```
import pandas as pd
import pandas as pd
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

[3]:
```
df = pd.read_csv('/content/census_data.csv')
```

[4]:
```
df.head()
```

[4]:
```
   age         workclass   education  education_num       marital_status  \
0   39         State-gov   Bachelors             13        Never-married
1   50  Self-emp-not-inc   Bachelors             13   Married-civ-spouse
2   38           Private     HS-grad              9             Divorced
3   53           Private        11th              7   Married-civ-spouse
4   28           Private   Bachelors             13   Married-civ-spouse

           occupation   relationship   race  gender  capital_gain  \
0        Adm-clerical  Not-in-family  White    Male          2174
1     Exec-managerial        Husband  White    Male             0
2   Handlers-cleaners  Not-in-family  White    Male             0
3   Handlers-cleaners        Husband  Black    Male             0
4      Prof-specialty           Wife  Black  Female             0

   capital_loss  hours_per_week native_country income_bracket
0             0              40  United-States          <=50K
1             0              13  United-States          <=50K
2             0              40  United-States          <=50K
3             0              40  United-States          <=50K
4             0              40           Cuba          <=50K
```

1.1 Convert the Lable Column to 0s and 1s

[5]:
```
df['income_bracket'] = (df['income_bracket'] == '>=50k').astype(int)
```

1.2 Perform a Train Test Split on the Data

[6]:
```
df['income_bracket'] = (df['income_bracket']== ">=50K").astype('int')
X = df.drop('income_bracket', axis=1)
y = df['income_bracket']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

1.3 Scale your numeric features

[7]:
```
numeric_features = ['age', 'education_num', 'capital_gain', 'capital_loss',␣
↪'hours_per_week']
categorical_features = ["education", "marital_status", "occupation",␣
↪"relationship", "race", "gender", "native_country"]
scaler = StandardScaler()
X_train_numeric = scaler.fit_transform(X_train[numeric_features])
X_test_numeric = scaler.transform(X_test[numeric_features])
```

1.4 Create an input layer which will be used to train your model

[9]:
```
import tensorflow as tf

input_layers = []
input_features = []

for feature in numeric_features:
    input_layer = tf.keras.Input(shape=(1,), name=f'{feature}_input')
    input_layers.append(input_layer)
    input_features.append(input_layer)

for feature in categorical_features:
    input_layer = tf.keras.Input(shape=(1,), dtype='string',␣
↪name=f'{feature}_input')
    lookup = tf.keras.layers.StringLookup(output_mode='int', num_oov_indices=1)
    lookup.adapt(X_train[feature])

    num_tokens = lookup.vocabulary_size()
    embedding_dim = int(num_tokens**0.5)
    embedding = tf.keras.layers.Embedding(input_dim=num_tokens,␣
↪output_dim=embedding_dim, name=f'{feature}_embedding')(lookup(input_layer))

    flatten = tf.keras.layers.Flatten()(embedding)

    input_layers.append(input_layer)
    input_features.append(flatten)
```

1.5 Build and compute model accuracy on training set. Use 10 epochs, and a batch size of 32.

[10]:
```
concatenated_features = tf.keras.layers.Concatenate()(input_features)

output = tf.keras.layers.Dense(1, activation='sigmoid')(concatenated_features)

model = tf.keras.Model(inputs=input_layers, outputs=output)
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

train_inputs = [X_train_numeric[:, i] for i in range(X_train_numeric.shape[1])]␣
 ↪+ [X_train[feature] for feature in categorical_features]
test_inputs = [X_test_numeric[:, i] for i in range(X_test_numeric.shape[1])] +␣
 ↪[X_test[feature] for feature in categorical_features]

history = model.fit(
    train_inputs,
    y_train,
    validation_data=(test_inputs, y_test),
    epochs=10,
    batch_size=32
)
```

```
Epoch 1/10
814/814 [==============================] - 5s 4ms/step - loss: 0.1488 -
accuracy: 0.9548 - val_loss: 0.0076 - val_accuracy: 1.0000
Epoch 2/10
814/814 [==============================] - 6s 7ms/step - loss: 0.0036 -
accuracy: 1.0000 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 3/10
814/814 [==============================] - 4s 5ms/step - loss: 0.0010 -
accuracy: 1.0000 - val_loss: 6.4399e-04 - val_accuracy: 1.0000
Epoch 4/10
814/814 [==============================] - 3s 4ms/step - loss: 4.5614e-04 -
accuracy: 1.0000 - val_loss: 3.1361e-04 - val_accuracy: 1.0000
Epoch 5/10
814/814 [==============================] - 6s 7ms/step - loss: 2.3373e-04 -
accuracy: 1.0000 - val_loss: 1.6964e-04 - val_accuracy: 1.0000
Epoch 6/10
814/814 [==============================] - 5s 7ms/step - loss: 1.3022e-04 -
accuracy: 1.0000 - val_loss: 9.7527e-05 - val_accuracy: 1.0000
Epoch 7/10
814/814 [==============================] - 3s 4ms/step - loss: 7.6095e-05 -
accuracy: 1.0000 - val_loss: 5.8069e-05 - val_accuracy: 1.0000
Epoch 8/10
814/814 [==============================] - 3s 4ms/step - loss: 4.5812e-05 -
accuracy: 1.0000 - val_loss: 3.5407e-05 - val_accuracy: 1.0000
Epoch 9/10
814/814 [==============================] - 8s 10ms/step - loss: 2.8128e-05 -
accuracy: 1.0000 - val_loss: 2.1934e-05 - val_accuracy: 1.0000
Epoch 10/10
814/814 [==============================] - 6s 7ms/step - loss: 1.7499e-05 -
accuracy: 1.0000 - val_loss: 1.3731e-05 - val_accuracy: 1.0000
```

1.6 Compute model accuracy on test set.

[11]:
```
performance = model.evaluate(test_inputs, y_test)
print('Test Loss and Accuracy:', performance)
```

```
204/204 [==============================] - 0s 2ms/step - loss: 1.3731e-05 -
accuracy: 1.0000
Test Loss and Accuracy: [1.3731121725868434e-05, 1.0]
```