Using TensorFlow with Categorical Features to Predict Income Bracket

May 2, 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**

[1]:
```python
import tensorflow as tf
from keras.models import Sequential
import pandas as pd
from keras.layers import Dense, Input
from keras.optimizers import SGD
import pandas as pd
import numpy as np
```

[2]:
```python
df = pd.read_csv('/Users/sandinatatu/Desktop/Tensorflow-Bootcamp-master/
    ↪02-TensorFlow-Basics/census_data.csv')
```

[3]:
```python
df.head()
```

[3]:
```
   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

[4]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from keras.layers import Input, Dense, Concatenate, StringLookup
from keras.models import Model
```

[5]:
```python
df = pd.read_csv('/Users/sandinatatu/Desktop/Tensorflow-Bootcamp-master/
    ↪02-TensorFlow-Basics/census_data.csv')
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

[8]:
```
import tensorflow as tf
from keras.layers import Input, StringLookup, Embedding, Concatenate, Dense
from keras.models import Model

input_layers = []
input_features = []

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

from keras.layers import Flatten

for feature in categorical_features:
    input_layer = Input(shape=(1,), dtype='string', name=f'{feature}_input')
    lookup = 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 = Embedding(input_dim=num_tokens, output_dim=embedding_dim,␣
↪name=f'{feature}_embedding')(lookup(input_layer))

    flatten = 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.

[9]:
```
concatenated_features = Concatenate()(input_features)

output = Dense(1, activation='sigmoid')(concatenated_features)

model = 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               1s 552us/step -
accuracy: 0.9202 - loss: 0.2086 - val_accuracy: 1.0000 - val_loss: 0.0026
Epoch 2/10
814/814               0s 398us/step -
accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 1.0000 - val_loss: 6.4740e-04
Epoch 3/10
814/814               0s 396us/step -
accuracy: 1.0000 - loss: 5.4252e-04 - val_accuracy: 1.0000 - val_loss:
2.6562e-04
Epoch 4/10
814/814               0s 398us/step -
accuracy: 1.0000 - loss: 2.3224e-04 - val_accuracy: 1.0000 - val_loss:
1.3161e-04
Epoch 5/10
814/814               0s 397us/step -
accuracy: 1.0000 - loss: 1.1776e-04 - val_accuracy: 1.0000 - val_loss:
7.2023e-05
Epoch 6/10
814/814               0s 397us/step -
```

```
accuracy: 1.0000 - loss: 6.6719e-05 - val_accuracy: 1.0000 - val_loss:
4.1636e-05
Epoch 7/10
814/814          0s 398us/step -
accuracy: 1.0000 - loss: 3.6895e-05 - val_accuracy: 1.0000 - val_loss:
2.4844e-05
Epoch 8/10
814/814          0s 397us/step -
accuracy: 1.0000 - loss: 2.3331e-05 - val_accuracy: 1.0000 - val_loss:
1.5171e-05
Epoch 9/10
814/814          0s 395us/step -
accuracy: 1.0000 - loss: 1.4148e-05 - val_accuracy: 1.0000 - val_loss:
9.4080e-06
Epoch 10/10
814/814          0s 398us/step -
accuracy: 1.0000 - loss: 8.6715e-06 - val_accuracy: 1.0000 - val_loss:
5.8934e-06
```

1.6 Compute model accuracy on test set.

[10]:
```
performance = model.evaluate(test_inputs, y_test)
print('Test Loss and Accuracy:', performance)
```

```
204/204          0s 252us/step -
accuracy: 1.0000 - loss: 5.9458e-06
Test Loss and Accuracy: [5.893413799640257e-06, 1.0]
```