IbottaProductClassification

May 21, 2023

This project deals with classifying products in a dataset from Ibotta. Products are classified as one of 7 different categories using the name of the product and the product brand. This is done using a tokenizer to build a dictionary of vectors associated with words commonly found in the product names and brands.

I tried several different types of models to classify the Ibotta dataset. First I explored using CNNs without recursion, varying the pooling structure, the number of convolutional layers, and the kernel sizes of these layers. Then I tried using RNNs without convolution. I tried using simple RNN layers, LSTM layers, and GRU layers, all while varying the number of layers and the number of parameters in each layer. For both the CNN and RNN models, I also testing various dropout and regularization rates. However, none of these models had as much predictive power as the models that combined recursion and convolution.

My final model is a recurrent convolutional neural network. The data is first run through a time distributed CNN with two convolutional layers and a dense layer. One convolutional layer has a smaller kernal size than the other, but all three layers utilize batch normalization and L2 regularization. There is a dropout layer after the time distributed CNN, followed by a bidirectional GRU layer with recurrent dropout and batch normalization. Then there is another dense layer with dropout, batch normalization, and L2 regularization. Finally, there is a dense layer with a softmax activation to make predictions.

[149]:
```
#load in various libraries and packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from random import seed
from tensorflow.keras.utils import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import Dense, Embedding, GRU,\
    Bidirectional, Conv1D, MaxPooling1D, GlobalMaxPooling1D,\
    Flatten, TimeDistributed, Dropout, BatchNormalization, SimpleRNN
from tensorflow.keras import models
from tensorflow.keras.utils import to_categorical as to_cat
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2
```

[150]:
```
#load in and format data
train = pd.read_csv('ibotta_train.csv')
test = pd.read_csv('ibotta_test.csv')
```

```
train_name = train['Name']
train_brand = train['Brand_name']
test_id = test['Id']
test_name = test['Name']
test_brand = test['Brand_name']
train_labels = train['Cat_code']
#combine data for processing
names = pd.concat([train_name, test_name]).reset_index(drop=True)
brands = pd.concat([train_brand, test_brand]).reset_index(drop=True)
brands = brands.fillna('unknown')
data = names + ' ' + brands
#convert strings to integer sequences
max_words = 5000
max_len = 15
tokenizer = Tokenizer(num_words = max_words)
tokenizer.fit_on_texts(data)
sequences = tokenizer.texts_to_sequences(data)
word_index = tokenizer.word_index
#pad sequences
data = pad_sequences(sequences, maxlen = max_len)
#split data back into training and testing sets
train = data[:8000]
test = data[8000:]
#convert labels to categorical data
train_labels = to_cat(train_labels)
#convert to float
train = np.asarray(train).astype('float32')
test = np.asarray(test).astype('float32')
#reshape data for recurrent convolution
rcnntrain = train
rcnntest = test
rcnntrain.shape = ((len(rcnntrain), max_len, 1))
rcnntest.shape = (len(rcnntest), max_len, 1)
```

[156]:
```
#define early stopping parameters
es = EarlyStopping(monitor='val_loss', mode='min',\
                   patience=5)
```

[160]:
```
seed(578)

#define rcnn model

cnn = models.Sequential()

cnn.add(Embedding(max_words, 256, input_length = max_len))

cnn.add(Conv1D(64, 3, padding = 'same', activation = 'relu',\
```

```
              kernel_regularizer = l2(1e-8)))
cnn.add(BatchNormalization())

cnn.add(Conv1D(64, 5, padding = 'same', activation = 'relu',\
              kernel_regularizer = l2(1e-8)))
cnn.add(BatchNormalization())

cnn.add(GlobalMaxPooling1D())

cnn.add(Dense(64, activation = 'relu',\
              kernel_regularizer = l2(1e-8)))
cnn.add(BatchNormalization())

rcnn = models.Sequential()
rcnn.add(TimeDistributed(cnn))
rcnn.add(Dropout(0.5))
rcnn.add(Bidirectional(GRU(128,
            dropout = 0.5,
            recurrent_dropout = 0.5,
            kernel_regularizer = l2(1e-8))))
rcnn.add(BatchNormalization())
rcnn.add(Dense(128, activation = 'relu',
            kernel_regularizer = l2(1e-8)))
rcnn.add(BatchNormalization())
rcnn.add(Dropout(0.5))

rcnn.add(Dense(7, activation = 'softmax'))

#compile model
rcnn.compile(optimizer = 'rmsprop',
             loss = 'categorical_crossentropy',
             metrics = ['accuracy'])
```

[ ]:
```
#fit model

history_rcnn = rcnn.fit(rcnntrain, train_labels,
                        epochs = 100,
                        batch_size = 64,
                        validation_split = 0.2,
                        callbacks = [es])
```

[ ]:
```
preds = rcnn.predict(test)
preds = np.array([np.argmax(x) for x in preds])
out = pd.DataFrame(np.transpose([test_id, preds]),\
                   columns = ['Id', 'Cat_code'])
out.to_csv('predictions.csv', index = False)
```