Student: Bahram Khodabandehlouee

Cs 614 - 900

Assignment: 3

Contents

Pitch

Reading customer reviews to understand their experiences is vital for businesses to identify strengths and weaknesses. Using CNN and RNN models, I categorized magazine subscription reviews into Positive, Negative, and Neutral categories. The objective was to predict sentiment, enabling businesses to make informed decisions based on customer feedback.

Data Source

To build this NLP system, I trained my model on the reviews data in the Magazine Subscriptions dataset. The Magazine Subscriptions dataset contains over 80 thousands of reviews by users, as well as additional information such as magazine title and user id. To get the dataset we have two options:

- from amazon website.

  Link: https://nijianmo.github.io/amazon/index.html

- from my google drive: https://drive.google.com/drive/folders/1m81xY8EdlE_G7mlmU5waa_u3HmBdRWCt?usp=sharing

Model and data justification

In my CNN model, I have an Input, which is a sequence vector of integers that are pre-padded. T is the feature number of my train-data. I choose my embedding dimension to be V+1, D. v is the number of observation and D is the dimensionality that I choose for embedding. I added 1 to v as indices start from 1 in TensorFlow. Lastly, I used 5 as num-class in softmax.

```
i = Input(shape=(T,))
x = Embedding(v+1, D)(i)
# we double the number of feature map in each conv layer,
x = Conv1D(32, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)
x = Conv1D(64, 3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(5, activation='softmax')(x)

model = Model(i,x)
```

At the time to build my model and compiling it I used

```
loss='sparse_categorical_crossentropy'
```

As I didn't use One-Hot encoder and this loss function will take care of it and internally implement One-Hot encoder on my classes.

Compile and Fit (CNN with 2 conv layer)

```
[ ]  y_train_modified = y_train - 1
     y_test_modified = y_test - 1

     model.compile(
         optimizer='adam',
         loss='sparse_categorical_crossentropy',
         metrics=['accuracy']
         )
     # By using 'sparse_categorical_crossentropy' as the loss function
     # and passing integer labels as the target, the model will internally
     # handle the conversion to one-hot encoded vectors.
     print('Training Model...')
     r = model.fit(
         data_train,
         y_train_modified,
         epochs = 15,
         validation_data = (data_test,y_test_modified)
     )
```

```
Training Model...
Epoch 1/15
```

Commented examples

I built two CNN model, One with2 conv and another one with 3 conv layer. I got 86% accuracy in CNN with 3 conv layer and 92% accuracy through 15 epochs, in CNN with 2 conv layer. In my explores, I tried CNN with 4 conv layer and the accuracy was less that cnn with 3 conv layer.

Creating CNN model (with 2 conv layer)

```
# we choose embedding dimensionality
D = 20

# We want the size of our embedding to be (v+1)x D,
# As the first index starts from 1
# thus if the final index of embedding matrix is v,
# Then it must have sise of v+1.

i = Input(shape=(T,))
x = Embedding(v+1, D)(i)
# we double the number of feature map in each conv layer, 23-->64-->128
x = Conv1D(32, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)
x = Conv1D(64, 3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(5, activation='softmax')(x)

model = Model(i,x)
```

Compile and Fit (CNN with 2 conv layer)

```
y_train_modified = y_train - 1
y_test_modified = y_test - 1

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
    )
# By using 'sparse_categorical_crossentropy' as the loss function
# and passing integer labels as the target, the model will internally
# handle the conversion to one-hot encoded vectors.
print('Training Model...')
r = model.fit(
    data_train,
    y_train_modified,
    epochs = 15,
    validation_data = (data_test,y_test_modified)
)
```

```
Training Model...
Epoch 1/15
1962/1962 [==============================] - 156s 75ms/step - loss: 0.9622 -
Epoch 2/15
1962/1962 [==============================] - 41s 21ms/step - loss: 0.7898 - a
Epoch 3/15
1962/1962 [==============================] - 36s 19ms/step - loss: 0.7100 - a
```

```
1962/1962 [==============================] - 26s 13ms/step - loss: 0.2338 - accuracy: 0.9226 - val_loss: 2.
Epoch 15/15
1962/1962 [==============================] - 25s 13ms/step - loss: 0.2222 - accuracy: 0.9275 - val_loss: 2.
```

I ran this model at first with 10 epochs and I got 86% accuracies. As it did not converge, I reran the model with 15 epochs and got 92% and almost converged. I didn't get a high validation accuracy. In my future work, I will work on fixing possible overfitting to fix this issue of having high accuracy in training but low in test set.

Then I made a RNN model and used LSTM.

create RNN model

```
# choosing Embedding dimensionality
D = 20

# Hidden state dimensionality
M =15

# we want the size of the embedding to be (v+1)x D,
# because in tensorflow, the first index starts from 1 ai
# thus it must have size of V+1 (because the final index

i = Input(shape=(T,))
x = Embedding(v+1, D)(i)
x = LSTM(M, return_sequences=True)(x)
x = GlobalMaxPooling1D()(x)
x = Dense(5, activation='softmax')(x)

model = Model(i,x)
```

Compile and fit RNN model:

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

print('Training Model...')
r = model.fit(
    data_train,
    y_train_modified,
    epochs = 10,
    validation_data = (data_test,y_test_modified)
)
```

I got accuracies around 80% in RNN model

Testing

As an example of confusion matrix, here we see the confusion matrix for RNN with LSTM model.

```
# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_labels)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)

report = classification_report(y_test, y_pred_labels)
# Print classification report
print("Classification Report:")
print(report)
```

I gave a random positive text to see how my CNN model predicts its rating.

Testing performance

```
random_text = "This magazine subscription is great!"
def sample_review(random_text):
  preprocessed_text = preprocess_text(random_text)
  tokenized_text = tokenizer.texts_to_sequences([preprocessed_text])
  padded_text = pad_sequences(tokenized_text, maxlen=T)
  return padded_text
predictions = model.predict(sample_review(random_text))
# predictions = model.predict(padded_text)

# Interpret the predictions.
predicted_class = np.argmax(predictions[0])

# Print the predicted class:
print("Predicted class:", predicted_class+1)
if predicted_class>=4:
  print('Positive Review')
if predicted_class <= 2:
  print('Negative Review')
if predicted_class ==3 :
  print('Neutral Review')
```

```
1/1 [==============================] - 0s 199ms/step
Predicted class: 5
Positive Review
```

checking out some reviews that got predicted wrong:

```
# Assuming you have already computed the predicted labels, y_pred_labels, a

# Create a list to store the indices of misclassified samples
misclassified_indices = []

# Find the indices of misclassified samples
for i in range(len(y_test)):
    if y_pred_labels[i] != y_test[i]:
        misclassified_indices.append(i)

# data_test,y_test_modified

# Print the misclassified samples
print("Misclassified Samples:")
for i in range(5):
# for index in misclassified_indices:
    print("Review:", data_test[misclassified_indices[i]])  # Assuming x_tes
    print("True Label:", y_test_modified[misclassified_indices[i]])
    print("Predicted Label:", y_pred_labels[misclassified_indices[i]])
    print("-------------------------------------")
```

```
Misclassified Samples:
Review: [  0    0    0 ... 102   88 933]
True Label: 4
Predicted Label: 3
-------------------------------------
Review: [  0    0    0 ...   4 393   95]
True Label: 3
Predicted Label: 4
-------------------------------------
```

```
841/841 [==============================] - 41s 49ms/step

Confusion Matrix:
[[     0      0      0      0      0      0]
 [  2271    356    220    109    399      0]
 [   571    322    330    122    246      0]
 [   378    223    588    345    527      0]
 [   146     86    290    972   2239      0]
 [   367     92    224   1187  14287      0]]
Classification Report:
              precision    recall  f1-score   support
```

0	0.00	0.00	0.00	0
1	0.33	0.11	0.16	3355
2	0.20	0.21	0.20	1591
3	0.13	0.17	0.14	2061
4	0.13	0.60	0.21	3733
5	0.00	0.00	0.00	16157
accuracy			0.12	26897
macro avg	0.13	0.18	0.12	26897
weighted avg	0.08	0.12	0.07	26897

As you see the rating of 2 and 3 are the hardest review star to predict as they could have both some parts of positive and negative sentiment in them. The are in green color. On the other hand, rating of 1 with all negative points and 4 and 5 star with almost all positive points are the easiest to predict in this model. They are in blue color in the matrix.

Code and instructions to run it

You can have access to the code in the link bellow. If you run this code in your local computer, please get the dataset from my google drive from link: https://drive.google.com/drive/folders/1m81xY8EdlE_G7mlmU5waa_u3HmBdRWCt?usp=sharing , unzip it and then load the json file in your local computer by

- df = pd.read_json('directory to the dataset/Magazine_Subscriptions.json', lines=True)

Then just skip the section for loading the dataset in my code as it is set to connect to my gdrive and will ask for permission. So, load the dataset and continue the code after the part of loading dataset

I agree to share my code with other students