Brian Elinsky

Professor Lawrence Fulton, Ph.D.

2020SP_MSDS_422-DL_SEC55

25 May 2020

Assignment 8: Language Modeling With an RNN

PROBLEM DESCRIPTION

The management team would like us to monitor customer reviews and complaint logs. They would like to have a computer program identify the negative reviews so that a human can respond. Presumably, humans are manually reviewing all of the reviews right now. By adding this filtration step to the process, humans will be able to spend less time on the task. If this model gets deployed to production, it will likely be an online model. It will probably make predictions real-time.

DATA PREPARATION

I did more pre-processing for this project than for other ones. I removed URLs, emojis, and punctuation from the tweets. For each tweet, I split it up into a list of words. Then I used GloVe to convert each word into a vector representation.

RESEARCH DESIGN AND MODELING METHODS

For the model, my first layer was an Embedding layer. Se second layer as a SpatialDropout1D layer. The third was a LSTM layer. The last layer was a 1-node dense layer, with a sigmoid activation. I used a binary cross entropy loss function because this is

a binary classification problem. And I used the accuracy metric. Where my two models differed was the optimizer. I fit one model with the Adam optimizer, and a second with the RMSprop optimizer.

RESULTS AND MODEL EVALUATION

Both models performed roughly the same. The choice of optimizer didn't seem to make much of a difference. The Adam optimizer version had a Kaggle score of 0.78527. The RMSprop version had a Kaggle score of 0.78323. (Account User ID 4810027) (https://www.kaggle.com/brianelinsky/).

MANAGEMENT RECOMMENDATIONS

78% accuracy isn't a particularly high score from a business perspective. Instead of optimizing accuracy, I could try optimizing for recall. This would allow management to still filter down the total number of tweets for review, but also ensure that we don't miss any negative reviews. I would recommend that instead of trying to increase the accuracy of the model.

```
>_ kaggle competitions submit -c nlp-getting-started -f submission.csv -m "Message"
```

0 submissions for Brian Elinsky

Sort by Most recent

All Successful Selected

Submission and Description	Public Score
submission-3.csv just now by Brian Elinsky Used the RMSprop optimizer this time	0.78527
submission-2.csv 25 minutes ago by Brian Elinsky Used the Adam Optimizer for this one	0.78323

No more submissions to show

```
import re
import string

import numpy as np
import pandas as pd
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from tensorflow.keras.initializers import Constant
from tensorflow.keras.layers import Embedding, LSTM, Dense, SpatialDropout1D
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tqdm import tqdm

# %%
train_file = "/Users/brianelinsky/Dropbox/ActiveProjects/modeling_projects/disaster_tweets/data/2020-05-25/train.csv"
test_file = "/Users/brianelinsky/Dropbox/ActiveProjects/modeling_projects/disaster_tweets/data/2020-05-25/test.csv"

train = pd.read_csv(train_file)
test = pd.read_csv(test_file)

# Temporarily combine to clean the data
df = pd.concat([train, test])


# Write functions to clean the tweets
def remove_URLs(text):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'', text)


def remove_html(text):
    html = re.compile(r'<.*?>')
    return html.sub(r'', text)


def remove_emoji(text):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)


def remove_punct(text):
    table = str.maketrans('', '', string.punctuation)
    return text.translate(table)


# Clean the test and train data
df['text'] = df['text'].apply(lambda x: remove_URLs(x))
df['text'] = df['text'].apply(lambda x: remove_html(x))
df['text'] = df['text'].apply(lambda x: remove_emoji(x))
df['text'] = df['text'].apply(lambda x: remove_punct(x))


# Takes a sentence, and converts it to a list of words
def create_corpus(df):
    corpus = []
    for tweet in df['text']:
        words = [word.lower() for word in word_tokenize(tweet) if ((word.isalpha() == 1))]
        corpus.append(words)
    return corpus


corpus = create_corpus(df)

# Use glove to convert the words into vectors
embed_dict = {}
with open('/Users/brianelinsky/Downloads/glove/glove.6B.100d.txt', 'r') as file:
    for line in file:
        values = line.split()
```

```
        word = values[0]
        vectors = np.asarray(values[1:], 'float32')
        embed_dict[word] = vectors
file.close()

MAX_LENGTH = 50
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
sequences = tokenizer.texts_to_sequences(corpus)

# Pad the sequences
tweet_padded = pad_sequences(sequences, maxlen=MAX_LENGTH, truncating='post', padding='post')

word_index = tokenizer.word_index
print('Number of unique words:', len(word_index))

num_words = len(word_index) + 1
embedding_matrix = np.zeros((num_words, 100))

for word, i in tqdm(word_index.items()):
    if i > num_words:
        continue

    emb_vec = embed_dict.get(word)
    if emb_vec is not None:
        embedding_matrix[i] = emb_vec

# Create first model.  This model will use the Adam optimizer
model = Sequential()

embedding = Embedding(num_words, 100, embeddings_initializer=Constant(embedding_matrix),
                      input_length=MAX_LENGTH, trainable=False)

model.add(embedding)
model.add(SpatialDropout1D(0.2))
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

optimzer = Adam(learning_rate=1e-5)

model.compile(loss='binary_crossentropy', optimizer=optimzer, metrics=['accuracy'])

model.summary()

# Create second model.  This model will use the RMSprop optimizer instead
model2 = Sequential()

embedding = Embedding(num_words, 100, embeddings_initializer=Constant(embedding_matrix),
                      input_length=MAX_LENGTH, trainable=False)

model2.add(embedding)
model2.add(SpatialDropout1D(0.2))
model2.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model2.add(Dense(1, activation='sigmoid'))

optimzer2 = RMSprop(learning_rate=1e-5)

model2.compile(loss='binary_crossentropy', optimizer=optimzer2, metrics=['accuracy'])

model2.summary()

train = tweet_padded[:train.shape[0]]
test = tweet_padded[train.shape[0]:]

# Split data into test and train sets
X_train, X_test, y_train, y_test = train_test_split(train, train['target'].values, test_size=0.15)
print('Shape of train', X_train.shape)
print("Shape of Validation ", X_test.shape)

# Train both models with 15 epochs
history = model.fit(X_train, y_train, batch_size=4, epochs=15, validation_data=(X_test, y_test), verbose=2)
history2 = model2.fit(X_train, y_train, batch_size=4, epochs=15, validation_data=(X_test, y_test), verbose=2)

sample_sub = pd.read_csv(
    '/Users/brianelinsky/Dropbox/ActiveProjects/modeling_projects/disaster_tweets/data/2020-05-25/sample_submission.csv')

# Generate predictions
y_pred = model.predict(test)
y_pred2 = model2.predict(test)
y_pred = np.round(y_pred).astype(int).reshape(3263)
```

```
y_pred2 = np.round(y_pred2).astype(int).reshape(3263)

# Create submissions
sub = pd.DataFrame({'id': sample_sub['id'].values.tolist(), 'target': y_pred})
sub2 = pd.DataFrame({'id': sample_sub['id'].values.tolist(), 'target': y_pred2})
sub.to_csv('submission.csv', index=False)
sub2.to_csv('submission2.csv', index=False)

sub.head()
sub2.head()
```