Codes: Human-Like AI Research Paper

(Zi Ying) Sheila Teo

```
import pandas as pd
pd.options.display.max_columns=100
import numpy as np

from tqdm.notebook import tqdm  #to print out progress bars
```

```
df = pd.read_csv('dataset.csv')
```

1) Text Preprocessing

```
df['question'] = df['question_title'] + ' ' + df['question_body']
```

```
def clean_words(text):
    #make all words lower case
    text = text.lower()

    #remove stopwords
    stopwords = nltk.corpus.stopwords.words("english")
    text = ' '.join([word for word in str(text).split() if word not in stopwords])

    #remove punctuations
    import string
    for punc_char in string.punctuation:
        text = str(text).replace(punc_char, ' ')

    return text
```

```
df['question'] = df['question'].apply(lambda x: clean_words(x))
```

```
df['answer'] = df['answer'].apply(lambda x: clean_words(x))
```

2) EDA

Distribution of Question Categories

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn')

grouped = df.groupby('category')  #gives a groupby object
grouped_count = grouped.size()  #gives a series
grouped_count.plot(kind='bar')

plt.xlabel('Question Category')
plt.ylabel('Frequency' )
plt.xticks(rotation=0)
```

```
plt.savefig('category_dist.png', dpi=700)
```

Word Cloud: Top 100 Frequently Occurring Words

In the questions and answers of the most frequent category (Technology)

```
df['all_question_and_answer'] = df['question'] + ' ' + df['answer']

text = ''
for i in df[df['category'] == 'TECHNOLOGY']['all_question_and_answer']:
    text += i

from wordcloud import WordCloud
stopwords = ['using', 'need', 'use', 's', 'gt', 'one', 'know', 'two', 'thing', 'lt', 'get', 'would', 'w

wordcloud = WordCloud(stopwords=stopwords, background_color='white', width=3000, height=3000,
max_words=100).generate(text)  #'max_words' sets the max no. of words to be plotted on word cloud

plt.imshow(wordcloud)
plt.axis('off')

plt.savefig('technology_wordcloud.png', dpi=700)
```

Distribution of 30 Targets

```
df.iloc[:, 11:].boxplot()

plt.xticks(rotation=90)
plt.title('Distribution of 30 Targets')

plt.tight_layout()
plt.savefig('target_dist', dpi=1000)
```

3) Split Data into Training and Test Sets

```
X = df['all_question_and_answer']
```

```
y = df.iloc[:, 11:32]  #30 targets
```

```
from sklearn.model_selection import train_test_split
X_train, X_test , y_train, y_test = train_test_split(X, y , test_size = 0.10)
```

4) Transform Text Features

Method 1: GloVe

```
from keras.preprocessing.text import Tokenizer  #removes punctuation marks and special characters and
#converts text to lowercase

vocab_size = 1000000  #define the size of vocabulary desired
oov_token = '<OOV>'  #out-of-verb token: used to represent words in the test set that aren't be found
```

```
#in the vocabulary of the training set

#fit tokenizer on training set
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)
tokenizer.fit_on_texts(X_train)
```

```
word_index = tokenizer.word_index  #depicts how words in training set are mapped to numbers
```

```
X_train_sequences = tokenizer.texts_to_sequences(X_train)  #converts text into a list of lists, where
#each inner list represents 1 sentence in the text
```

```
#since sentences have different lengths, sequences above also have different lengths
#but: sequences need to have an equal length to be passed into model later -> truncating longer
#sequences and padding shorter sequences with zeros

from keras.preprocessing.sequence import pad_sequences

max_length = max([len(x) for x in X_train_sequences])  #max sequence length = max number of words
#in sentence in the training set
padding_type='post'
truncation_type='post'

X_train_sequences_padded = pad_sequences(X_train_sequences, padding=padding_type,
truncating=truncation_type, maxlen=max_length)
```

```
#check that sequences have been padded correctly

X_train_sequences_padded.shape
```

```
#transform X_test in the same way

X_test_sequences = tokenizer.texts_to_sequences(X_test)
X_test_sequences_padded = pad_sequences(X_test_sequences, padding=padding_type,
truncating=truncation_type, maxlen=max_length)
```

```
#download pre-trained GloVe embeddings previously trained on the Wikipedia and GigaWord datasets

!wget --no-check-certificate \
    http://nlp.stanford.edu/data/glove.6B.zip \
    -O /tmp/glove.6B.zip
```

```
#extract downloaded word embeddings to a temporary folder

import zipfile

with zipfile.ZipFile('/tmp/glove.6B.zip', 'r') as zip_ref:
    zip_ref.extractall('/tmp/glove')
```

```
#append word embeddings to a dictionary

embeddings_index = {}
f = open('/tmp/glove/glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
```

```
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

```
#use dictionary above to create an embedding matrix for each word in the training set

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

for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)  #shape: (100,)
    if embedding_vector is not None:  #words not found in embedding index will be all zeros
        embedding_matrix[i, :100] = embedding_vector
```

```
#create embedding layer for deep learning model

from keras.layers import Embedding

embedding_layer = Embedding(len(word_index) + 1,
                            max_length,
                            weights=[embedding_matrix],
                            input_length=max_length,
                            trainable=False)
```

Method 2: BERT Model

```
import torch  #pytorch!
from transformers import DistilBertModel
from transformers import DistilBertTokenizer
```

```
#import pre-trained BERT model: distilbert-base-uncased-distilled-squad

tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased-distilled-squad')
model = DistilBertModel.from_pretrained('distilbert-base-uncased-distilled-squad',
output_hidden_states=True)
```

```
#tokenize text using BERT tokenizer

tokenized = df['question'].apply((lambda x: tokenizer.encode(x, add_special_tokens=True,
truncation=True)))  #need to truncate bc sentences are longer than max length of 512
```

```
#pad sequence due to reasons outlined previously

max_len = 0
for i in tokenized.values:
    if len(i) > max_len:
        max_len = len(i)

padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
```

```
#create attention mask that BERT requires

attention_mask = np.where(padded != 0, 1, 0)
attention_mask = torch.tensor(attention_mask)  #convert into format required: tensor
```

```
#put BERT model into evaluation mode (as opposed to training mode) which turns off dropout
#regularization

model.eval()
```

```
#pass text into BERT to extract text embeddings

all_embeddings = np.zeros((6079, 512, 3072))

for i in tqdm(range(len(padded))):
    input_id = padded[i]
    mask = attention_mask[i]

    input_id_tensor = torch.tensor([input_id])

    with torch.no_grad():
        outputs = model(input_id_tensor, attention_mask=mask)

        #use concatenation of last four layers as the word embeddings we want
        all_hidden_states = outputs[1]
        word_embeddings = torch.cat([all_hidden_states[i] for i in [-1,-2,-3,-4]], dim=-1)

        all_embeddings[i:i+1, :, :] = word_embeddings.numpy()
        #change 'all_embeddings' array in-place: to optimize on memory
```

```
#create features from BERT word embeddings to be used in deep learning model
X_train = all_embeddings
```

5) Build Model: Bi-directional LSTM with 1D Convolutions

```
from keras.layers import Embedding, SpatialDropout1D, BatchNormalization, Bidirectional, LSTM, Conv1D,
GlobalMaxPooling1D, Dropout, Dense
from keras.models import Sequential
from keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val_loss', patience=5)
```

```
def model_1():
    model = Sequential()

    model.add(embedding_layer)

    model.add(SpatialDropout1D(0.2))

    model.add(Conv1D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu'))
    model.add(GlobalMaxPooling1D())

    model.add(Bidirectional(LSTM(128, dropout=0.2)))
    model.add(BatchNormalization())

    model.add(Dense(30))

    model.compile(optimizer='Adam', loss='binary_crossentropy')
```

```
    return model
```

```
model = model_1()
```

```
history = model.fit(X_train, y_train, epochs=200, batch_size=64, validation_data=(X_test, y_test),
                    callbacks=[es], verbose=1)
```

```
#create plot of loss against epochs

loss = history.history['loss']
test_loss = history.history['val_loss']

epochs = range(1, 201)  #use 201 here bc 200 epochs used

plt.plot(epochs, loss, 'b', label='training set')  #'b'=plot blue line
plt.plot(epochs, test_loss, 'r', label='test set')  #'r'=plot red line
plt.xlabel('Epochs')
plt.ylabel('Cross Entropy')
plt.legend()
plt.ylim((0.3, 2))

#annotate minimum test mae on graph
minimum = np.min(test_loss)
epoch_with_min = test_loss.index(np.min(test_loss))
plt.plot(epoch_with_min, minimum, marker='o', ms=8, color='black')  #ms = marker size
plt.text(epoch_with_min-15, minimum+0.1, 'minimum')

plt.savefig('results 1', dpi=700)
```

6) Tune Model

Increase regularization

- Increase traditional dropout from 20% to 40%
- Add recurrent dropout at 40% to the LSTM layer

```
def model_2():
    model = Sequential()

    model.add(embedding_layer)

    model.add(SpatialDropout1D(0.4))

    model.add(Conv1D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu'))
    model.add(GlobalMaxPooling1D())

    model.add(Bidirectional(LSTM(128, dropout=0.4, recurrent_dropout=0.4)))
    model.add(BatchNormalization())

    model.add(Dense(30))

    model.compile(optimizer='Adam', loss='binary_crossentropy')

    return model
```

```
model = model_2()
```

```
history = model.fit(X_train, y_train, epochs=200, batch_size=64, validation_data=(X_test, y_test),
                    callbacks=[es], verbose=1)
```

```
#create plot of loss against epochs

loss = history.history['loss']
test_loss = history.history['val_loss']

epochs = range(1, 201)  #use 201 here bc 200 epochs used

plt.plot(epochs, loss, 'b', label='training set')  #'b'=plot blue line
plt.plot(epochs, test_loss, 'r', label='test set')  #'r'=plot red line
plt.xlabel('Epochs')
plt.ylabel('Cross Entropy')
plt.legend()
plt.ylim((0.5, 2))

#annotate minimum test mae on graph
minimum = np.min(test_loss)
epoch_with_min = test_loss.index(np.min(test_loss))
plt.plot(epoch_with_min, minimum, marker='o', ms=8, color='black')  #ms = marker size
plt.text(epoch_with_min-15, minimum-0.1, 'minimum')

plt.savefig('results 2', dpi=700)
```

Increase model complexity

- Increase number of convolutional layers from 1 to 2
- Increase number of LSTM layers from 1 to 3
- Tune the learning rate

```
from keras.layers import MaxPooling1D
from keras.optimizers import Adam

def model_3():
    model = Sequential()

    model.add(embedding_layer)

    model.add(SpatialDropout1D(0.4))

    model.add(Conv1D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu'))
    model.add(MaxPooling1D())

    model.add(Conv1D(filters=64, kernel_size=3, strides=1, padding='same', activation='relu'))
    model.add(GlobalMaxPooling1D())

    model.add(Bidirectional(LSTM(128, dropout=0.4, recurrent_dropout=0.4, return_sequences=True)))
    model.add(BatchNormalization())

    model.add(Bidirectional(LSTM(128, dropout=0.4, recurrent_dropout=0.4, return_sequences=True)))
    model.add(BatchNormalization())
```

```
    model.add(Bidirectional(LSTM(128, dropout=0.4, recurrent_dropout=0.4)))
    model.add(BatchNormalization())

    model.add(Dense(30))

    model.compile(optimizer=Adam(learning_rate=0.04), loss='binary_crossentropy')

    return model
```

```
model = model_3()
```

```
history = model.fit(X_train, y_train, epochs=200, batch_size=64, validation_data=(X_test, y_test),
                    callbacks=[es], verbose=1)
```

```
#create plot of loss against epochs

loss = history.history['loss']
test_loss = history.history['val_loss']

epochs = range(1, 201)  #use 201 here bc 200 epochs used

plt.plot(epochs, loss, 'b', label='training set')  #'b'=plot blue line
plt.plot(epochs, test_loss, 'r', label='test set')  #'r'=plot red line
plt.xlabel('Epochs')
plt.ylabel('Cross Entropy')
plt.legend()

#annotate minimum test mae on graph
minimum = np.min(test_loss)
epoch_with_min = test_loss.index(np.min(test_loss))
plt.plot(epoch_with_min, minimum, marker='o', ms=8, color='black')  #ms = marker size
plt.text(epoch_with_min-5, minimum+0.05, 'minimum')

plt.savefig('results 3', dpi=700)
```