HU Extension --- Final Project --- S89A DL for NLP

Michael Lee & Micah Nickerson

PART 2A - ANCHOR MODEL GENERATION

This notebook **finds Anchor Words** that predict high scores, to optimize Adversarial Attacks

Project Master Variables

In [0]:
```
#Data Storage Parameters
dataset_dir = "Data Sets/asap-aes"
adversarial_dir = "Data Sets/adversarial_asap"
model_save_dir = "Model Files"
selected_essay_id = 2
training_set_file = dataset_dir+"/training_set_rel3.xls"

###Test sets:
test_set_file = dataset_dir+"/valid_set.xls"
test_set_scores_file = dataset_dir+"/valid_sample_submission_5_column.csv"

#Data Embedding Parameters
# Take First X words from each essay, abandon rest
max_len = 1118 #longest essay

# Word Dimensionality - consider the top 15,000 words in the dataset
max_words = 20000
```

In [0]:
```
def make_prediction(modelname,sampess):
    sample_prediction = modelname.predict(test_set_essays_emb[sampess:sampess+1]
    return sample_prediction

def calculate_score(prediction):
    score = {}
    score[1]=prediction[0,0]
    score[2]=prediction[0,1]
    score[3]=prediction[0,2]
    score[4]=prediction[0,3]
    score[5]=prediction[0,4]
    score[6]=prediction[0,5]
    calculate_score = max(score, key=score.get)
    return(calculate_score)
```

Load Packages and Dependencies

In [0]:
```
#data loading
import os

# python modules
from argparse import Namespace
from collections import Counter
import json
import re
import string
import statistics

####data manipulation####
import numpy as np
from numpy.random import shuffle
import pandas as pd

####word2vec encoding####
import gensim

####data visualization####
%matplotlib notebook
import matplotlib.pyplot as plt
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
%matplotlib inline
plt.style.use('ggplot')

####CNN tools####

#keras
import keras
from keras import layers
from keras import models
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense
from keras.models import load_model
from keras import regularizers
from keras import metrics
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
Using TensorFlow backend.
```

Load and Clean Test Set

In [0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code)

Enter your authorization code:
··········
Mounted at /content/drive
```

- Load, Filter and Clean Data

In [0]:
```
#verify data paths
print(training_set_file)
print(test_set_file)

#load excel into dataframe
raw_training_set = pd.read_excel(training_set_file, sheet_name='training_set')
test_set = pd.read_excel(test_set_file, sheet_name='valid_set')
test_set_scores = pd.read_csv(test_set_scores_file)

print("\nEntire Corpus for ASAP:")
print("Training Set:",raw_training_set.shape)
#print("Validation:",valid_set.shape)
print("Test Set:",test_set.shape,"\n")

#filter data by essay set
essay_fltr =  raw_training_set['essay_set']== selected_essay_id
training_set = raw_training_set[essay_fltr]

essay_fltr =  test_set['essay_set']== selected_essay_id
test_set = test_set[essay_fltr]

essay_fltr =  test_set_scores['essay_set']== selected_essay_id
test_set_scores = test_set_scores[essay_fltr]

#remove empty n/a cells
training_set = training_set.drop(['rater3_domain1','rater1_trait1','rater1_trait
test_set = test_set.drop(['domain2_predictionid'], axis=1)

training_set_top = training_set.head()
#print(training_set_top)
test_set_top = test_set.head()
#print(test_set_top)

#3 sets, training, validation and testing

print("Selected Essay Set #%s Corpus:" % selected_essay_id)
print("Training Set:",training_set.shape)
print("Test Set:",test_set.shape)
print("Total Data Set:", training_set.shape[0]+test_set.shape[0])
```

```
/content/drive/Shared drives/CSCI S-89A - Group Project/Data Sets/asap-aes/trai
ning_set_rel3.xls
/content/drive/Shared drives/CSCI S-89A - Group Project/Data Sets/adversarial_a
sap/valid_set_plus_ADVERSARIAL_ESSAYS-ML.xls

Entire Corpus for ASAP:
Training Set: (12978, 28)
Test Set: (4219, 5)

Selected Essay Set #2 Corpus:
Training Set: (1800, 9)
Test Set: (601, 4)
Total Data Set: 2401
```

- Split data into Essay and Label Sets

In [0]:
```
#extract essays and convert to NumPy for Keras
training_set_essays = training_set['essay']
training_set_essays = training_set_essays.values
test_set_essays = test_set['essay']
test_set_essays = test_set_essays.values

#extract scores and convert to NumPy for Keras
training_set_dom1scores = training_set['domain1_score']
training_set_dom1scores = training_set_dom1scores.values

#extract domain#1 predicted scores
#data cleaning due to strange score input shape
test_set_dom1scores = []
for i in (range(test_set_scores.shape[0])):
    if (i % 2) == 0: #print every other cell, since second cell is domain#2
        asdf = test_set_scores['predicted_score'].values[i]
        i_score_no = float(asdf)
        #print(asdf)
        #test_set_dom1scores = test_set_dom1scores.append({'predicted_score': asd
        test_set_dom1scores.append(i_score_no)
#convert to NumPy Array
test_set_dom1scores = np.asarray(test_set_dom1scores)
```

Encoding Essays

- Tokenization and Word Indexing of Essays

In [0]:
```
# Vectorize the Essays

#TEMPORARILY COMBINE TRAIN AND TEST TO SIMPLIFY EMBEDDING PROCESS
#single embedding process, max token index
lengthmark = len(training_set_essays)
combined_essays = np.append(training_set_essays,test_set_essays)

# Tokenize the data
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(combined_essays)
sequences = tokenizer.texts_to_sequences(combined_essays)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

# Pad sequences that are shorter than others
combined_data_pen = pad_sequences(sequences, maxlen=max_len)

#SPLIT TRAINING AND TEST SETS BACK
train_data_pen = combined_data_pen[:lengthmark]
test_data_pen = combined_data_pen[lengthmark:]

# Load the label
print('Shape of Testing data tensor:', test_data_pen.shape)
```

```
Found 17023 unique tokens.
Shape of Training data tensor: (1800, 1118)
Shape of Testing data tensor: (601, 1118)
```

- One Hot Encoding of Essay Scores 1-6
  - **2** = 010000
  - **6** = 000001

In [0]:
```
train_labels_pen = np.zeros((0, 6))

#Scores to Dummy Variable Conversion
#Training (and Validation) Set
for item in training_set_dom1scores:
      if item==1:
          train_labels_pen = np.append(train_labels_pen, [[1,0,0,0,0,0]],axis = 0
      elif item==2:
          train_labels_pen = np.append(train_labels_pen, [[0,1,0,0,0,0]],axis = 0
      elif item==3:
          train_labels_pen = np.append(train_labels_pen, [[0,0,1,0,0,0]],axis = 0
      elif item==4:
          train_labels_pen = np.append(train_labels_pen, [[0,0,0,1,0,0]],axis = 0
      elif item==5:
          train_labels_pen = np.append(train_labels_pen, [[0,0,0,0,1,0]],axis = 0
      else:
          train_labels_pen = np.append(train_labels_pen, [[0,0,0,0,0,1]],axis = 0

test_labels_pen = np.zeros((0, 6))

#Scores to Dummy Variable Conversion
#Testing Set
for item in test_set_dom1scores:
      if item==1:
          test_labels_pen = np.append(test_labels_pen, [[1,0,0,0,0,0]],axis = 0)
      elif item==2:
          test_labels_pen = np.append(test_labels_pen, [[0,1,0,0,0,0]],axis = 0)
      elif item==3:
          test_labels_pen = np.append(test_labels_pen, [[0,0,1,0,0,0]],axis = 0)
      elif item==4:
          test_labels_pen = np.append(test_labels_pen, [[0,0,0,1,0,0]],axis = 0)
      elif item==5:
          test_labels_pen = np.append(test_labels_pen, [[0,0,0,0,1,0]],axis = 0)
      else:
          test_labels_pen = np.append(test_labels_pen, [[0,0,0,0,0,1]],axis = 0)

print("Test Labels Shape:" ,test_labels_pen.shape)
```

```
Training Labels Shape: (1800, 6)
Test Labels Shape: (601, 6)
```

In [0]:
```
#TEST SET IS LEFT ALONE

val_set_essays = training_set_essays
val_set_dom1scores = training_set_dom1scores

#split coded scores
set_split_test = int((len(train_data_pen))*test_split)
training_set_essays_emb, val_set_essays_emb = train_data_pen[:set_split_test], tr
training_set_dom1scores_emb, val_set_dom1scores_emb = train_labels_pen[:set_split
#split the unencoded scores
training_set_dom1scores, val_set_dom1scores = training_set_dom1scores[:set_split_

test_set_essays_emb = test_data_pen
test_set_dom1scores_emb = test_labels_pen

print("\nTest Set Essays and matching Scores:")
print("Shape: ",test_set_essays_emb.shape, test_set_dom1scores_emb.shape)
```

```
Training Set Essays and matching Scores:
Shape:  (1440, 1118) (1440, 6)

Validation Set Essays and matching Scores:
Shape:  (360, 1118) (360, 6)

Test Set Essays and matching Scores:
Shape:  (601, 1118) (601, 6)
```

- Embedding Essays using GloVe Embedding

Load Black Box in

In [0]:
```
test_model_black_box = load_model(model_save_dir+'/D1_76_BLACKBOX_CNN.h5')
```

Anchor Identification Model

- Finding Anchor Words that predict high scores

In [0]:
```
!pip install anchor_exp
!pip install -q spacy && python -m spacy download en_core_web_lg && python -m spa

from anchor import anchor_text

import spacy
spacy_nlp = spacy.load('enlg')
```

In [0]:
```
sample_ids = [0]

for idx in sample_ids:
    print('Index: %d, Feature: %s' % (idx, test_set_essays[idx]))
    print('True Score: %s' % (test_set_dom1scores[idx]))
    estimatedscore=[calculate_score(make_prediction(test_model_black_box, idx))]
    def estimator(estimatedscore):
      estimator = np.asarray(estimatedscore)
      return estimator

    #classifier_fn([text])[0]

    explainer = anchor_text.AnchorText(spacy_nlp, [1,2,3,4,5,6], use_unk_distribu
    exp = explainer.explain_instance(test_set_essays[idx], estimator, threshold=0

    max_pred = 2
    print('Key Signal from Anchors: %s' % (' AND '.join(exp.names())))
    print(exp.features())
    print('Precision: %.2f' % exp.precision())
    print()

    #exp.show_in_notebook()
```

In [0]:
```
#using a real essay

sample_ids = [0]

for idx in sample_ids:
    print('Index: %d, Feature: %s' % (idx, training_set_essays[idx]))
    print('True Score: %s' % (training_set_dom1scores[idx]))
    estimatedscore=[calculate_score(make_prediction(test_model_black_box, idx))]
    def estimator(estimatedscore):
      estimator = np.asarray(estimatedscore)
      return estimator

    #classifier_fn([text])[0]

    explainer = anchor_text.AnchorText(spacy_nlp, [1,2,3,4,5,6], use_unk_distribu
    exp = explainer.explain_instance(training_set_essays[idx], estimator, thresho

    max_pred = 2
    print('Key Signal from Anchors: %s' % (' AND '.join(exp.names())))
    print('Precision: %.2f' % exp.precision())
    print()

    #exp.show_in_notebook()
```