DISTRACTOR GENERATION THROUGH TEXT RANKING

PROBLEM STATEMENT:

We are given a pair of question q answer a and Distractors d and we have to generate best pair of

for all q,a =>d in Range(i) where i is the number of question answer pair

where :

d = {d1,d2,d3},

d ~ a

d is related to q

MAPPING INTO ML-PROBLEM:

- The given problem falls under Natural Language Processing(NLP) where we have to rank th between question and answer as well as answer and distractor.
- The problem statement falls under category of Supervised Text-Ranking problem
- Based on the semantic similarity relations between and the distractor and answer as well as similarity ranking parameters will be considered as the best set of distractors

APPROACH :

For all given pair of question 'q' answer 'a' and distractors 'd' we are going to consider the followin

- Average-word2vec using NLTK - for embedding the text data
- cosine similarity - for finding the similarity metrics between docs
- Pos-tag similarity using NLTK - for structural similarity
- Token similarity using wordnet - for common tokens or words
- TFIDF - for finding most similar words in the documents
- Length similarity - for finding the similarity between word count in d,a
- Noun-phrase pos synonyms - for finding similar words for the NER
- semantic similarity using Wordnet
- EDIT distance = for number of edist needed to change the given d to a from the above featur such a way that for a given i d = (di1~~>di2>~~di3) ~ a or d ~ a and d ~ q where(>,<,~ is in terms
    - After the featurization techniques we are going to use 2-stage ranking :

1. In the 1st stage we are going to use Logistic regression for learn-to-rank and feature r
2. In the 2nd stage we are going to use Randomforest for learn-to-rank to increasing the

```
# Importing libraries
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
from textgenrnn import textgenrnn
```

```
from google.colab import drive
import pandas as pd
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
# Reading train and test files
train_data = pd.read_csv('Train.csv')
test_data = pd.read_csv('Test.csv')
result_data = pd.read_csv('Results.csv')
```

```
# Printing the shape of the train and test matrix
print(train_data.shape)
print(test_data.shape)
```

```
(31499, 3)
(13500, 2)
```

```
# printing sample data from train
train_data.head(10)
```

	question	answer_t
0	Meals can be served	in rooms at 9:00 p
1	It can be inferred from the passage that	The local government can deal with the proble
2	The author called Tommy 's parents in order to	help them realize their influence on Ton
3	It can be inferred from the passage that	the writer is not very willing to use idi
4	How can we deal with snake wounds according to...	Stay calm and do n't mo
5	What was the writer 's problem when she studie...	She missed her family very mu
6	Who were killed on February 5 in a small town ...	Chen Jianqing and one of her part
7	According to the writer , which of the followi...	Soccer is popular all over the world , but t
8	During a fire children often	pa
9	What 's the title of the passage ?	Five children died in a kindergarten bus acc

```
# printing sample test data
test_data.head(10)
```

	question	answer_text
0	What 'S the main idea of the text ?	The lack of career -- based courses in US high...
1	In the summer high season , Finland does nt se...	the sun is out at night
2	If you want to apply for Chinese Business Inte...	have to get confirmed at least twice
3	That afternoon , the boy 's clothes were dry b...	nobody made room for him in the water .
4	Which of the following statements is NOT true ?	There are twelve countries in the World Wildli...
5	The problem of " lock - in " can be dangerous ...	it may make it difficult for customers to reco...
6	The passage is mainly about	how Billy made blueberry juice with his uncle
7	Which of the following in not true ?	The snail 's teeth ca n't be worn out ..
8	What should you do at mealtime ?	Eat the food your host family gives you .
9	The part of " Only you can make a card like th...	how to make a meaningful DIY card

```
# checking for null values
train_data.isnull().sum()
```

```
question       0
answer_text    0
distractor     0
dtype: int64
```

```
# Converting the text data in dataframe from Nonetype to string
train_data['question'] =  train_data.question.astype('str')
train_data['answer_text'] = train_data.answer_text.astype('str')
train_data['distractor'] = train_data.distractor.astype('str')
```

TEXT PREPROCESSING

```
# @title Applying filtering of special charecters in the text and converti
from tqdm import tqdm
preprocessed_quesn = []
# tqdm is for printing the status bar
for phrase in tqdm(train_data['question']):
  phrase = re.sub(r"did n't",'did not',phrase)
  phrase = re.sub(r"won't", "will not", phrase)
  phrase = re.sub(r"can\'t", "can not", phrase)
  phrase = re.sub(r"\'s", " is", phrase)
  phrase = re.sub(r"\'d", " would", phrase)
  phrase = re.sub(r"\'ll", " will", phrase)
  phrase = re.sub(r"\'t", " not", phrase)
  phrase = re.sub(r"\'ve", " have", phrase)
  phrase = re.sub(r"\'m", " am", phrase)
  phrase = phrase.replace('\\r', ' ')
  phrase = phrase.replace('\\n', ' ')
  phrase = re.sub("[^A-Za-z0-9.,]+", ' ',phrase)
  phrase = phrase.lower()
  preprocessed_quesn.append(phrase)
```

Applyin
convert

```
100%|██████████| 31499/31499 [00:00<00:00, 76198.67it/s]
```

```
# printing sample of preprocessed question
preprocessed_quesn[1]
```

```
'it can be inferred from the passage that'
```

```
@title Preprocessing special charecters and converting into clean text
om tqdm import tqdm
eprocessed_ans = []
tqdm is for printing the status bar
r phrase in tqdm(train_data['answer_text'].astype('str')):
phrase = re.sub(r"did n't",'did not',phrase)
phrase = re.sub(r"won't", "will not", phrase)
phrase = re.sub(r"can\'t", "can not", phrase)
phrase = re.sub(r"\'s", " is", phrase)
phrase = re.sub(r"\'d", " would", phrase)
phrase = re.sub(r"\'ll", " will", phrase)
phrase = re.sub(r"\'t", " not", phrase)
```

Preproc

```
phrase = re.sub(r"\'ve", " have", phrase)
phrase = re.sub(r"\'m", " am", phrase)
phrase = phrase.replace('\\r', ' ')
phrase = phrase.replace('\\n', ' ')
phrase = re.sub("[^A-Za-z0-9.,]+", ' ',phrase)
phrase = phrase.lower()
preprocessed_ans.append(phrase)
```

```
100%|██████████| 31499/31499 [00:00<00:00, 77456.40it/s]
```

The distractor feature has set of 1-15 sentences which we need to preprocess and convert into se repeated sentences and dis-joint sentences .

```
# @title modelling the sentences and removing unneccessary punctuations
from nltk.tokenize import sent_tokenize
dist_vect = []
for phrase in tqdm(train_data['distractor']):
  phrase = re.sub(r"did n't",'did not',phrase)
  phrase = re.sub(r"won't", "will not", phrase)
  phrase = re.sub(r"can\'t", "can not", phrase)
  phrase = re.sub(r"\'s", " is", phrase)
  phrase = re.sub(r"\'d", " would", phrase)
  phrase = re.sub(r"\'ll", " will", phrase)
  phrase = re.sub(r"\'t", " not", phrase)
  phrase = re.sub(r"\'ve", " have", phrase)
  phrase = re.sub(r"\'m", " am", phrase)
  phrase = phrase.replace('\\r', ' ')
  phrase = phrase.replace('\\n', ' ')
  phrase = re.sub("[^A-Za-z0-9.,]+", ' ',phrase)
  phrase = phrase.lower()
  phrase = re.split(',',phrase)
  dist_vect.append(phrase)
```

modelli
punctu

```
100%|██████████| 31499/31499 [00:00<00:00, 42492.60it/s]
```

```
#Filling the incomplete distractors with suitable values
vel= []
vale =[]
for vale in dist_vect:
  if len(vale)==1:
    vale.append('Both the bove are correct')
    vale.append('None of the above')
  elif len(vale)==2:
    vale.append('None of the above')
  vel.append(vale)
```

```
# counting how many distractor list have distractors greater than 3
for i in range(4,16):
  t = len([id for id,v in enumerate(vel) if len(v)==i])
  print(i,t)
```

```
4 1219
5 318
6 290
7 41
8 26
9 23
10 7
11 1
12 10
13 0
14 0
15 2
```

```
temp = [i for i,v in enumerate(vel) if len(v)>5]
print(len(temp))
temp_df = pd.DataFrame()
temp_df['qstn'] = preprocessed_quesn
temp_df['ans'] = preprocessed_ans
temp_df['options'] = [v for v in vel]
```

```
401
```

```
#dropping rows with options more than 5
temp_df = temp_df.drop(index=temp)
train_data = train_data.drop(index=temp)
```

```
temp_df.shape
```

```
(31098, 3)
```

```
# @title selecting values whose distractor set is in range of 4-5
tem = [i for i,v in enumerate(temp_df['options']) if (len(v)>3 and len(v)<
tp_tem = pd.DataFrame(temp_df['options'].values[tem])
tp_tem.head()
```

selectir

	0
0	[if some tragedies occur again , relevant de...
1	[millions of people all over the world are pl...
2	[sun yukun had to change his residence status...
3	[1 , 500 people died on titanic is maiden vo...
4	[if the project is completed , the world is ...

```
#Clearing the duplicates in distractors
from collections import Counter
dl = []
dup = lambda x : Counter(x)
for i,tres in enumerate(tp_tem[0]):
  dpi = dup(tres)
  dd = [k for k,v in dpi.items() if(v==2)]
  dl.append(dd)
  dl = [j for j in dl if j]
```

```
# extracting all text in sequence
import itertools
from itertools import chain
ele = []
for el in chain.from_iterable(dl):
  ele.append(el)
```

```
# removing duplicate from the text
res = []
for i in temp_df['options']:
  if i not in res:
    res.append(i)
```

```
# Connecting the dissjoint sentences in distractors
tt = tp_tem[0]
con = [el for i,el in enumerate(tt)]

def cat(lst):
  if len(lst)==4:
    l = sorted(lst)
    m=[l[0]+l[1],l[2],l[-1]]
    return m
  if len(lst)==5:
    k = sorted(lst)
    a = k[0]+k[2]
    b = k[1]+k[3]
    if (k[0]+k[2])>(k[1]+k[3]) :
      n = [a,b,k[-1]]
      return n
    else :
      n = [a+k[1],k[2],k[-1]]
      return n

prep = []
for tp in tp_tem[0]:
   pre = cat(tp)
   prep.append(pre)
```

```
# generating and splitting into individual distractors
temp_df['options'].iloc[tem] = prep
d1 = [] # distractor 1
d2 = [] # distractor 2
d3 = [] # distractor 3
for tm in temp_df['options']:
  d1.append(tm[0])
  d2.append(tm[1])
  d3.append(tm[2])
  d3.append(tm[2])
```

```
# merging thee individual distractors
temp_df['d1'] = d1
temp_df['d2'] = d2
temp_df['d3'] = d3
```

```
# joining the text from distractors
d = []
for i in range(0,len(temp_df)):
  f = d1[i] + "," + d2[i] + "," + d3[i]
  d.append(f)

temp_df['d'] = d
```

FEATURIZATION TECHNIQUES

```
'''
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')
words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))
```

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)


'''
```

```
'\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open
```

```
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
 []; # @title the avg-w2v for each sentence/review is stored questions
m(temp_df['qstn']): # for each review/sentence
os(300) # as word vectors are of zero length
 @title num of words with a valid vector in the sentence/review
tence.split(): # for each word in a review/sentence
glove_words:
+= model[word]
ds += 1
 0:
nt_words
rs.append(vector)

_vectors))
_vectors[0]))
```

the avg

```
100%|██████████| 31098/31098 [00:00<00:00, 41983.77it/s]31098
300
```

```
[]; # @title the avg-w2v for each sentence/review is stored in answers
temp_df['ans']): # for each review/sentence
(300) # as word vectors are of zero length
m of words with a valid vector in the sentence/review
nce.split(): # for each word in a review/sentence
ove_words:
 model[words]
+= 1

_words
s.append(vector)

vectors))
vectors[0]))
```

the avg

```
100%|██████████| 31098/31098 [00:00<00:00, 39812.24it/s]31098
300
```

```
d1_avg_w2v_vectors = []; # @title the avg-w2v for each sentence/review is
for sent in tqdm(temp_df['d1']): # for each review/sentence
    vect = np.zeros(300) # as word vectors are of zero length
    cnt_wor =0; # num of words with a valid vector in the sentence/review
    for wor in sent.split(): # for each word in a review/sentence
        if wor in glove_words:
            vect += model[wor]
            cnt_wor += 1
    if cnt_wor != 0:
        vect /= cnt_wor
    d1_avg_w2v_vectors.append(vect)

print(len(d1_avg_w2v_vectors))
print(len(d1_avg_w2v_vectors[0]))
```

the avg
set

```
100%|██████████| 31098/31098 [00:00<00:00, 42280.09it/s]31098
300
```

```
d2_avg_w2v_vectors = []; # @title the avg-w2v for each sentence/review is
for senten in tqdm(temp_df['d2']): # for each review/sentence
    vecto = np.zeros(300) # as word vectors are of zero length
    cnt_worde =0; # num of words with a valid vector in the sentence/revie
    for worde in senten.split(): # for each word in a review/sentence
        if worde in glove_words:
            vecto += model[worde]
            cnt_worde += 1
    if cnt_worde != 0:
        vecto /= cnt_worde
    d2_avg_w2v_vectors.append(vect)

print(len(d2_avg_w2v_vectors))
print(len(d2_avg_w2v_vectors[0]))
```

the avg
distrac

```
100%|██████████| 31098/31098 [00:00<00:00, 49523.00it/s]31098
300
```

```
d3_avg_w2v_vectors = []; # @title the avg-w2v for each sentence/review is
for sentenc in tqdm(temp_df['d3']): # for each review/sentence
    vec = np.zeros(300) # as word vectors are of zero length
    cnt_wordes =0; # num of words with a valid vector in the sentence/revi
    for wordes in sentenc.split(): # for each word in a review/sentence
        if wordes in glove_words:
            vec += model[wordes]
            cnt_wordes += 1
    if cnt_wordes != 0:
        vec /= cnt_wordes
    d3_avg_w2v_vectors.append(vec)
```

the avg
set

```
print(len(d3_avg_w2v_vectors))
print(len(d3_avg_w2v_vectors[0]))
```

```
100%|██████████| 31098/31098 [00:00<00:00, 54834.67it/s]31098
300
```

```
d_avg_w2v_vectors = []; # @title the avg-w2v for each sentence/review is s
for sentences in tqdm(temp_df['d']): # for each review/sentence
    vectors = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/revie
    for words in sentence.split(): # for each word in a review/sentence
        if words in glove_words:
            vectors += model[words]
            cnt_words += 1
    if cnt_words != 0:
        vectors /= cnt_words
    d_avg_w2v_vectors.append(vectors)

print(len(d_avg_w2v_vectors))
print(len(d_avg_w2v_vectors[0]))
```

the avg

distrac

```
100%|██████████| 31098/31098 [00:00<00:00, 42446.78it/s]31098
300
```

```
qt_avg_w2v_vectors = []; # @title the avg-w2v for each sentence/review is
for sentence in tqdm(test_data['question']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_word =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_word += 1
    if cnt_word != 0:
        vector /= cnt_word
    qt_avg_w2v_vectors.append(vector)

print(len(qt_avg_w2v_vectors))
print(len(qt_avg_w2v_vectors[0]))
```

the avg

```
100%|██████████| 13500/13500 [00:00<00:00, 45897.80it/s]13500
300
```

```
# @title computing cosine similarity of two text documents
import math
def cosine_similar(v1,v2):
    "compute cosine similarity of v1 to v2: (v1 dot v2)/{||v1||*||v2||)"
    sumxx, sumxy, sumyy = 0, 0, 0
    for i in range(len(v1)):
        x = v1[i]; y = v2[i]
        sumxx += x*x
        sumyy += y*y
```

compu

```
        sumxy += x*y
    return sumxy/math.sqrt(sumxx*sumyy)

csad = [cosine_similar(ans_avg_w2v_vectors[i],d_avg_w2v_vectors[i]) for i
```

```
csaq = [cosine_similar(ans_avg_w2v_vectors[i],q_avg_w2v_vectors[i]) for i in range(0,le
csd1q = [cosine_similar(q_avg_w2v_vectors[i],d1_avg_w2v_vectors[i]) for i in range(0,le
csd2q = [cosine_similar(q_avg_w2v_vectors[i],d2_avg_w2v_vectors[i]) for i in range(0,le
csd3q = [cosine_similar(q_avg_w2v_vectors[i],d3_avg_w2v_vectors[i]) for i in range(0,le
csad1 = [cosine_similar(d1_avg_w2v_vectors[i],ans_avg_w2v_vectors[i]) for i in range(0,
csad2 = [cosine_similar(d2_avg_w2v_vectors[i],ans_avg_w2v_vectors[i]) for i in range(0,
csad3 = [cosine_similar(ans_avg_w2v_vectors[i],d3_avg_w2v_vectors[i]) for i in range(0,
```

```
# getting the lengths of the text in the fields
a = [len(ele) for ele in test_data['answer_text']]
b = {len(ale):t for t,ale in enumerate(d1)}
c = {len(ela):s for s,ela in enumerate(d2)}
d = {len(eal):r for r,eal in enumerate(d3)}
```

```
at_avg_w2v_vectors = []; # @title the avg-w2v for each sentence is stored
for sentences in tqdm(test_data['answer_text']): # for each review/sentenc
    vectors = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/revie
    for words in sentence.split(): # for each word in a review/sentence
        if words in glove_words:
            vectors += model[words]
            cnt_words += 1
    if cnt_words != 0:
        vectors /= cnt_words
    at_avg_w2v_vectors.append(vectors)

print(len(at_avg_w2v_vectors))
print(len(at_avg_w2v_vectors[0]))
```

the avg

```
100%|██████████| 13500/13500 [00:00<00:00, 49618.73it/s]13500
300
```

```
tcsaq = [cosine_similar(at_avg_w2v_vectors[i],qt_avg_w2v_vectors[i]) for i in range(0,l
```

```
tcsad1 = [cosine_similar(at_avg_w2v_vectors[i],d1_avg_w2v_vectors[i]) for i in range(0,
tcsad2 = [cosine_similar(at_avg_w2v_vectors[i],d2_avg_w2v_vectors[i]) for i in range(0,
tcsad3 = [cosine_similar(at_avg_w2v_vectors[i],d3_avg_w2v_vectors[i]) for i in range(0,
```

```
# appending the values to corresponding list
dt1 = []
dt2 = []
dt3 = []
for i in range(0,len(at_avg_w2v_vectors)):
  t1 = d1[i]
  dt1.append(t1)
  t2 = d2[i]
  dt2.append(t2)
  t3 = d3[i]
  dt3.append(t3)
```

```
# creating the dictionary out of the index values and text for hashing
td1 = {k:v for v in tcsad1 for k in dt1}
td2 = {a:b for b in tcsad2 for a in dt2}
td3 = {p:q for q in tcsad3 for p in dt3}
```

```
#dictionary with reference
td = dict()
td.update(td1)
td.update(td2)
td.update(td3)
```

```
# creating the negative or less raking values data
test_negative = tcsad1 or tcsad2 or tcsad3
test1 = tcsad1 and tcsad2
test2 = test1 and tcsad3
test_positive1 = test1 and test2
test_positive2 = test1 or test2
```

```
# @title pos-tagging similarity the text for structural analysis
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk import pos_tag, word_tokenize
w2va = temp_df['ans'].apply(lambda x: pos_tag(word_tokenize(x)))
w2vd = temp_df['d'].apply(lambda x: pos_tag(word_tokenize(x)))
w2vd1 = temp_df['d1'].apply(lambda x: pos_tag(word_tokenize(x)))
w2vd2 = temp_df['d2'].apply(lambda x: pos_tag(word_tokenize(x)))
w2vd3 = temp_df['d3'].apply(lambda x: pos_tag(word_tokenize(x)))
```

pos-tag

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
posans = list(w2va)
posd = list(w2vd)
posd1 = list(w2vd1)
posd2 = list(w2vd2)
posd3 = list(w2vd3)
```

```
dd1 = dict(zip(csad1,d1))
dd2 = dict(zip(csad2,d2))
dd3 = dict(zip(csad3,d3))
```

```
df = pd.DataFrame()
df['q'] = csaq
df['a'] = csad1
df['b'] = csad2
df['c'] = csad3
```

```
# creating the positive or high ranked text
positive = [csad1 and csad2 and csad3]
```

```
neg1 = [csad1 or csad2]
neg2 = [csad3 or d]
negative  =  neg1.append(neg2)
```

```
#Tfidf for frequency count
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
ans_tfidf = vectorizer.fit_transform(temp_df['ans'])
print("Shape of matrix after one hot encodig ",ans_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (31098, 2570)
```

```
# Tfidf on d1
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
d1_tfidf = vectorizer.fit_transform(temp_df['d1'])
print("Shape of matrix after one hot encodig ",d1_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (31098, 2604)
```

```
# tfidf on d2
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
d2_tfidf = vectorizer.fit_transform(temp_df['d2'])
print("Shape of matrix after one hot encodig ",d2_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (31098, 2040)
```

```
# tfidf on d3
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
d3_tfidf = vectorizer.fit_transform(temp_df['d3'])
print("Shape of matrix after one hot encodig ",d3_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (31098, 1437)
```

```
# tfidf on questions
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
q_tfidf = vectorizer.fit_transform(temp_df['qstn'])
print("Shape of matrix after one hot encodig ",q_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (31098, 1758)
```

```
# @title  Edit Distance on answer text
import editdistance
ed1a = []
ed2a = []
ed3a = []
for i in range(0,len(d2)):
  ea1 = editdistance.eval(temp_df['ans'].values[i],d1[i])
  ed1a.append(ea1)
  ea2 = editdistance.eval(temp_df['ans'].values[i],d2[i])
  ed2a.append(ea2)
  ea3 = editdistance.eval(temp_df['ans'].values[i],d3[i])
  ed3a.append(ea3)

print(len(ed1a))
```

Edit Dis

```
31098
```

```
# length of sentences
def length(s1,s2):
  l=len(s1)-len(s2)
  return l
lenad1 = []
lenad2 = []
lenad3 = []
for i in range(0,len(d1)):
  lad1 = length(temp_df['ans'].values[i],d1[i])
  lenad1.append(lad1)
  lad2 = length(temp_df['ans'].values[i],d2[i])
  lenad2.append(lad2)
  lad3 = length(temp_df['ans'].values[i],d3[i])
  lenad3.append(lad3)
```

```
# @title Sequence matching score
import difflib
def seqm(a,b):
  seq = difflib.SequenceMatcher(None,a,b)
  c = seq.ratio()
  return c

seqad1 = []
seqad2 = []
seqad3 = []
for i in range(0,len(temp_df)):
  sad1 = seqm(preprocessed_ans[i],d1[i])
  seqad1.append(sad1)
  sad2 = seqm(preprocessed_ans[i],d2[i])
  seqad2.append(sad2)
  sad3 = seqm(preprocessed_ans[i],d3[i])
  seqad3.append(sad3)
len(seqad3)
```

Sequer

```
31098
```

```
# @title Sentence similarity using wordnet
def length_dist(synset_1, synset_2):
    """
    Return a measure of the length of the shortest path in the semantic
    ontology (Wordnet in our case as well as the paper's) between two
    synsets.
    """
    l_dist = sys.maxint
    if synset_1 is None or synset_2 is None:
        return 0.0
    if synset_1 == synset_2:
        # if synset_1 and synset_2 are the same synset return 0
        l_dist = 0.0
    else:
        wset_1 = set([str(x.name()) for x in synset_1.lemmas()])
        wset_2 = set([str(x.name()) for x in synset_2.lemmas()])
        if len(wset_1.intersection(wset_2)) > 0:
            # if synset_1 != synset_2 but there is word overlap, return 1.
            l_dist = 1.0
        else:
            # just compute the shortest path between the two
            l_dist = synset_1.shortest_path_distance(synset_2)
            if l_dist is None:
                l_dist = 0.0
    # normalize path length to the range [0,1]
    return math.exp(-alpha * l_dist)
```

Senten

```
# @title noun phrase pos synonyms using wordnet
def length_dist(synset_1, synset_2):
    """
    Return a measure of the length of the shortest path in the semantic
    ontology (Wordnet in our case as well as the paper's) between two
    synsets.
    """
    l_dist = sys.maxint
    if synset_1 is None or synset_2 is None:
      return 0.0
    if synset_1 == synset_2:
      # if synset_1 and synset_2 are the same synset return 0
      l_dist = 0.0
    else:
      wset_1 = set([str(x.name()) for x in synset_1.lemmas()])
      wset_2 = set([str(x.name()) for x in synset_2.lemmas()])
      if len(wset_1.intersection(wset_2)) > 0:
        # if synset_1 != synset_2 but there is word overlap, return 1.0
        l_dist = 1.0
      else:
        # just compute the shortest path between the two
        l_dist = synset_1.shortest_path_distance(synset_2)
        if l_dist is None:
```

noun pl

```
        l_dist = 0.0
  # normalize path length to the range [0,1]
    return math.exp(-alpha * l_dist)
```

```
# @title Q-A analysis using wordnet
def _analyze_query(self):
  tagged = nltk.pos_tag(self.ir_query)
  ir_query_tagged = []
  for word, pos in tagged:
    pos = {pos.startswith('N'): wordnet.NOUN,
          pos.startswith('V'): wordnet.VERB,
          pos.startswith('J'): wordnet.ADJ,
          pos.startswith('R'): wordnet.ADV,}.get(pos, None)
    if pos:
      synsets = wordnet.synsets(word, pos=pos)
    else:
      synsets = wordnet.synsets(word)
    ir_query_tagged.append((word, synsets))

  # Add additional special hidden term
  ir_query_tagged.append(('cause', [wordnet.synset('cause.v.01')]))
  self.ir_query_tagged = ir_query_tagged
```

Q-A an

```
# @title short sentence similarity
nltk.download('wordnet')
from nltk.corpus import wordnet as wn
def get_best_synset_pair(word_1, word_2):
    """
    Choose the pair with highest path similarity among all pairs.
    Mimics pattern-seeking behavior of humans.
    """
    max_sim = -1.0
    synsets_1 = wn.synsets(word_1)
    synsets_2 = wn.synsets(word_2)
    if len(synsets_1) == 0 or len(synsets_2) == 0:
        return None, None
    else:
        max_sim = -1.0
        best_pair = None, None
        for synset_1 in synsets_1:
            for synset_2 in synsets_2:
               sim = wn.path_similarity(synset_1, synset_2)
               if sim is not None and sim > max_sim:
                   max_sim = sim
                   best_pair = synset_1, synset_2
        return best_pair
```

short s

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

```
test_q = list(test_data['question'])
test_a = list(test_data['answer_text'])
```

```
# @title Token similarity using wordnet
# @title Token similarity using nltk
import nltk.corpus
import nltk.tokenize.punkt
import nltk.stem.snowball
import string
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words('english')
stopwords.extend(string.punctuation)
stopwords.append('')
def token_set_match(a, b, threshold=0.5):
    """Check if a and b share token."""
    tokens_a = [token.lower().strip(string.punctuation) for token in word_
                    if token.lower().strip(string.punctuation) not in stop
    tokens_b = [token.lower().strip(string.punctuation) for token in word_
                    if token.lower().strip(string.punctuation) not in stop

    # Calculate Jaccard similarity
    ratio = len(set(tokens_a).intersection(tokens_b)) / float(len(set(toke
    if ratio >= threshold:
      return 1
    else :
      return 0
```

Token s

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
mda = []
md1a = []
md2a = []
md3a = []
mqd = []
md1a = []
md2a = []
md3a = []
mqd1 = []
mqd2 = []
mqd3 = []
for i in range(0,len(temp_df)):
  matchd1a = token_set_match(d1[i],temp_df['ans'].values[i])
  md1a.append(matchd1a)
  matchd2a = token_set_match(d2[i],temp_df['ans'].values[i])
  md2a.append(matchd2a)
  matchd3a = token_set_match(d3[i],temp_df['ans'].values[i])
  md3a.append(matchd3a)
  matchqd1 = token_set_match(temp_df['qstn'].values[i],d1[i])
  mqd1.append(matchqd1)
  matchqd2 = token_set_match(temp_df['qstn'].values[i],d2[i])
  mqd2.append(matchqd2)
  matchqd3 = token_set_match(temp_df['qstn'].values[i],d3[i])
  mqd3.append(matchqd3)
```

```
train = pd.DataFrame()
train['csad1'] = csad1
train['csad2'] = csad2
```

```
train['csad3'] = csad3
train['csd1q'] = csd1q
train['csd2q'] = csd2q
train['csd3q'] = csd3q
train['csaq'] = csaq
train['edit_d1a'] = ed1a
train['edit_d2a'] = ed2a
train['edit_d3a'] = ed3a
train['length_ad1'] = lenad1
train['length_ad2'] = lenad2
train['length_ad3'] = lenad3
train['seq_ad1'] = seqad1
train['seq_ad2'] = seqad2
train['seq_ad3'] = seqad3
train['md1a'] = md1a
train['md2a'] = md2a
train['md3a'] = md3a
train['mqd1'] = mqd1
train['mqd2'] = mqd2
train['mqd3'] = mqd3
```

```
train.shape
```

```
(31098, 22)
```

```
# filling nan values
train = train.fillna(0.9)
```

```
X = train.drop(['csad3'], axis=1)
Y = train['csad3']
Y = Y.astype(int)
Y = Y.values.reshape(-1,1)
```

```
X_train,Y_train =  X,Y
print(X_train.shape, Y_train.shape)
print("="*100)
```

```
(31098, 21) (31098, 1)
==================================================================================
```

```
test_data['csaq'] = tcsaq
X_test = test_data['csaq'].values.reshape(-1,1)
X_test = pd.DataFrame(X_test)
```

```
# taking opposite of answer as one distractor
tesd1 = []
for phrase in test_data['answer_text']:
  phrase = re.sub(r"did n't",'did',phrase)
  phrase = re.sub(r"won't", "will", phrase)
  phrase = re.sub(r"can\'t", "can", phrase)
  phrase = re.sub(r" not ", " ", phrase)
  phrase = re.sub(r" was ", " was not", phrase)
  phrase = re.sub(r"\'s", " is not", phrase)
  phrase = re.sub(r"\'d", " would not", phrase)
  phrase = re.sub(r"\'ll", " will not", phrase)
  phrase = re.sub(r"\'t", " not", phrase)
  phrase = re.sub(r"\'ve", " have not", phrase)
  phrase = re.sub(r"dis\w+", " ",phrase)
  phrase = re.sub(r"un\w+" , " ", phrase)
  phrase = re.sub(r"in\w+" , " ", phrase)
  phrase = re.sub(r"\'m", " am", phrase)
  tesd1.append(phrase.lower())
```

```
tp1 = []
tp2 = []
for i in range(0,len(test_data)):
  tp1.append(d3[i])
  tp2.append(d2[i])
```

```
tespos  = pd.DataFrame()
tespos['test_positive1'] = tp1
tespos['test_positive2'] = tp2
tespos['test_positive3'] = tesd1
```

MODELLING (LEARN TO RANK)

```
# @title Applying text to rank using Logistic Regression
from sklearn.linear_model import SGDRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import precision_score
from sklearn.utils import shuffle
def train_model(model, prediction_function, X_train, y_train, X_test):
  model.partial_fit(X_train, y_train)
  y_train_pred = prediction_function(model, X_train)
  print('train precision: ' + str(precision_score(y_train, y_train_pred)))
  print('train recall: ' + str(recall_score(y_train, y_train_pred)))
  print('train accuracy: ' + str(accuracy_score(y_train, y_train_pred)))
  y_test_pred = prediction_function(model, X_test)
  return model
def get_predicted_outcome(model, data):
    return np.argmax(model.predict(data), axis=1).astype(np.float32)
def get_predicted_rank(model, data):
    return model.predict_proba(data)[:, 1]

clf1 = train_model(SGDRegressor(), get_predicted_outcome, X_train, Y_train
```

Applyir

```
# @title Applying learn to rank using RandomForest classifier
from sklearn.ensemble import RandomForestClassifier
def train_model(model, prediction_function, X_train, y_train, X_test):
  model.partial_fit(X_train, y_train)
  y_train_pred = prediction_function(model, X_train)
```

Applyir

```
  print('train precision: ' + str(precision_score(y_train, y_train_pred)))
  print('train recall: ' + str(recall_score(y_train, y_train_pred)))
  print('train accuracy: ' + str(accuracy_score(y_train, y_train_pred)))
  y_test_pred = prediction_function(model, X_test)
  return model
def get_predicted_outcome(model, data):
    return np.argmax(model.predict(data), axis=1).astype(np.float32)
def get_predicted_rank(model, data):
    return model.predict_proba(data)

clf2 = train_model(RandomForestClassifier(), get_predicted_outcome, X_trai
```

```
#combining the best evaluated rank for the test distractors
test_positive1 = [clf1[0] or clf1[0]]
test_positive2 = [clf2[1] or clf1[1]]
test_positive1 = [clf2[2] or clf1[2]]
```

```
# @title combining all the options as mentioned using sepertor or
test_data['distractor'] = tespos.test_positive1.map(str) + " or " + tespos
```

combir

```
del test_data['csaq']
```

```
# Printing the sample data of the ranked distractors
test_data['distractor'].head()
```

```
0     in the dining room from 7 30 a. m. to 9 15 p....
1     the central government has established sound ...
2    None of the above or  blame tommy for his fail...
3     nothere are no ways to master idioms  or  non...
4    None of the above or Both the bove are correct...
Name: distractor, dtype: object
```

```
# storing the predictions in csv file
test_data.to_csv('predictions.csv')
```

CONCLUSION:

THE TEXT WITH SIMILAR LENGTHS AND SIMILAR SEMANTICS WITH ALL THE ABOVE FEATURE IN THE LEADERBOARD SCORE