



# Solution Summary

## Detecting Similar Phrases in Financial Documents

Word2vec with AWS SageMaker BlazingText

Raghuveer Varahagiri  
December 14, 2018







## Problem

Use Amazon SageMaker and BlazingText algorithm to build a Machine Learning model that can identify similar phrases from finance domain documents.

## Steps

1. Training Data Preparation
2. Training & Tuning the Model
3. Deploying the model, Inference using real data
4. Results & Evaluation



# 1

## 1. Data Preparation

1. General English language corpus
2. Finance domain-specific corpus

All text was finally normalized, prepared as one sentence per line as required by BlazingText, merged into a single file and uploaded to S3

## 1.1 Data Preparation: English language corpus

- This is a generic english language corpus of 1 billion words available at <http://www.statmt.org/lm-benchmark/>
- The notebook downloads the corpus, normalizes the text using NLTK module, and writes the output in the format accepted by Amazon SageMaker BlazingText algorithm -- which is one sentence per line in a single input file
- All the data files are uploaded to S3
- This exercise was repeated once with all stop words removed as part of the normalization step, and once with all stop words retained.

[LM1B data set v4.ipynb](#)

# 1.1 Data Preparation: English language corpus [LM1B data set v4.ipynb](#)

## Data Ingestion

```
In [4]: %%time

# download the lm-1b corpus as tar.gz file

!pwd
!wget http://www.statmt.org/lm-benchmark/1-billion-word-language-modeling-benchmark-r13output.tar.gz
```

```
In [9]: # copy the data to /tmp folder where we have 20GB of space temporarily
!cp 1-billion-word-language-modeling-benchmark-r13output.tar.gz /tmp
```

```
In [14]: %%time

#uncompress

!tar -xvzf /tmp/1-billion-word-language-modeling-benchmark-r13output.tar.gz -C /tmp
```

Download and Uncompress  
The 1 Billion Words dataset

```
1-billion-word-language-modeling-benchmark-r13output/
1-billion-word-language-modeling-benchmark-r13output/training-monolingual.tokenized.shuffled/
1-billion-word-language-modeling-benchmark-r13output/training-monolingual.tokenized.shuffled/news.en-00024-of-00100
1-billion-word-language-modeling-benchmark-r13output/training-monolingual.tokenized.shuffled/news.en-00057-of-00100
.....
```

```
In [15]: #upload raw data to S3 (as a backup since /tmp folder will not persist after notebook instance is stopped and restarted)
sess.upload_data(path='/tmp/1-billion-word-language-modeling-benchmark-r13output.tar.gz', bucket=bucket, key_prefix='original')
```

```
Out[15]: 's3://sagemaker-wipro-raghuvv2/original/1-billion-word-language-modeling-benchmark-r13output.tar.gz'
```

# 1.1 Data Preparation: English language corpus [LM1B data set v4.ipynb](#)

```
In [5]: %%time
#Import NLTK (Natural Language Toolkit)
import nltk
nltk.download('all') #needs to be downloaded each time
```

```
[nltk_data] Downloading collection 'all'
[nltk_data] |
[nltk_data] | Downloading package abc to
[nltk_data] | /home/ec2-user/nltk data...
```

```
In [14]: %%time

# process the source files from input folder, normalize them using NLTK, and write the normalized text files to output folder

from nltk import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import os

input_directory = '/tmp/lm1b'
directory = os.fsencode(input_directory)
output_directory = '/tmp/lm1b-norm'

filecount = len(os.listdir(directory))
print ("Found " + str(filecount) + " input files in the input directory.")
counter = 0
```

Normalize the text using NLTK



# 1.1 Data Preparation: English language corpus [LM1B data set v4.ipynb](#)

```
for file in os.listdir(directory):
    filename = os.fsdecode(file)
    input_filename = os.path.join(input_directory, filename)
    counter += 1
    print("Processing source file " + str(counter) + " of " + str(filecount) + " : " + input_filename)

    # Load data
    input_file = open(input_filename, 'rt')
    text = input_file.read()
    input_file.close()

    output_sentences = []

    # split into sentences
    sentences = sent_tokenize(text)
    for sentence in sentences:
        # split into words
        tokens = word_tokenize(sentence)
        # convert to lower case
        tokens = [w.lower() for w in tokens]
        # remove punctuation from each word
        table = str.maketrans('', '', string.punctuation)
        stripped = [w.translate(table) for w in tokens]
        # remove remaining tokens that are not alphabetic
        words = [word for word in stripped if word.isalpha()]
        # DISABLED - filter out stop words
        #stop_words = set(stopwords.words('english'))
        #words = [w for w in words if not w in stop_words]
        #print(words[:100])
        #output sentence
        output_sentence = " ".join(words)
        output_sentences.append(output_sentence)

    #write output
    output_filename = os.path.join(output_directory, filename)
    print("Writing output file:" + output_filename)
    with open(output_filename, 'w') as output_file:
        for item in output_sentences:
            output_file.write("%s\n" % item)
```

Normalize the text using NLTK

```
Found 99 input files in the input directory.
Processing source file 1 of 99 : /tmp/lm1b/news.en-00057-of-00100
Writing output file:/tmp/lm1b-norm/news.en-00057-of-00100
```

## 1.2 Data Preparation: Finance domain-specific corpus (A)

- For financial and business domain specific corpus, the following data source was identified during research:
- The US S.E.C. receives annual reports and Form 10-K filings from all publicly listed companies every year and makes this data available as tagged XBRL data.
- <https://www.sec.gov/dera/data/financial-statement-and-notes-data-set.html>
- The Form 10-K data for last 10 years is made available by US S.E.C along with a guide to the data structure. For example each quarter's ZIP file contains multiple files that contain the various types of data submitted by the companies. Through a careful study of the data structures, it was possible to pick the right data files from each ZIP file and the right rows and columns of data from each CSV file. Only the field values corresponding to the XBRL tag names that end with "TextBlock" were picked to be part of the corpus as they represented the flowing english text descriptions in annual reports. This eliminated most of the tabular, numeric, and other non-textual information.
- This filtered corpus was processed through the same NLTK normalization steps as above and uploaded to S3.
- [\*\*\*Financial Data Sets \(SEC 10K Notes\) Scratchpad v3.ipynb\*\*\*](#)



# 1.2 Data Preparation: Finance domain-specific corpus (A)

## [Financial Data Sets \(SEC 10K Notes\) Scratchpad v3.ipynb](#)

```
In [2]: # source1: https://www.reddit.com/r/Python/comments/38jmis/download_zip_files_from_a_website_using_python/
# source2: https://code.tutsplus.com/tutorials/how-to-download-files-in-python--cms-30099
# modified by raghuveer 10/15/2018

#Author: Bellerophon_
#Date Created: 04/12/15
#Usage: Used to scrape a website for links that end in .zip and list them
#Requirements: BeautifulSoup lib
#Notes:

import urllib.request
import urllib.error
import urllib.parse
import urllib.robotparser

#from urllib import Request, urlopen, URLError
import urllib
import os
from bs4 import BeautifulSoup

newfile = open('seczipfiles.txt','w')
#print newfile

#print ("Running script.. ")
#Set variable for page to be open and url to be concatenated
url = "https://www.sec.gov"
page = urllib.request.urlopen('https://www.sec.gov/dera/data/financial-statement-and-notes-data-set.html').read()

#File extension to be looked for.
extension = ".zip"

#Use BeautifulSoup to clean up the page
soup = BeautifulSoup(page, "lxml")
soup.prettify()

#Find all the links on the page that end in .zip
for anchor in soup.findAll('a', href=True):
    links = url + anchor['href']
    if links.endswith(extension):
        newfile.write(links + '\n')
newfile.close()
```

```
!cat seczipfiles.txt
```

```
https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2018q3_notes.zip
https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2018q2_notes.zip
https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-sets/2018q1_notes.zip
```

Scrape SEC website using BeautifulSoup and collect the links for ZIP files

# 1.2 Data Preparation: Finance domain-specific corpus (A)

## [Financial Data Sets \(SEC 10K Notes\) Scratchpad v3.ipynb](#)

```
directory = os.fsencode('/tmp/sec')

#Read through the lines in the text file and download the zip files.
#Handle exceptions and print exceptions to the console

with open('seczipfiles.txt', 'r') as url:
    for line in url:
        if line:
            try:
                ziplink = line.rstrip()

                if line.find('//'):
                    zipfilename = line.rsplit('//', 1)[1].rstrip()
                    print ("Fetching: ", zipfilename)

                    response = urllib.request.urlopen(ziplink)
            except urllib.error.URLError as e:
                if hasattr(e, 'reason'):
                    print ('Failed to reach server.')
                    print ('Reason: ', e.reason)
                    continue
                elif hasattr(e, 'code'):
                    print ('The server failed to fulfill the request.')
                    print ('Error code: ', e.code)
                    continue
            else:
                zipcontent = response.read()
                completeName = os.path.join(directory, os.fsencode(zipfilename))
                with open (completeName, 'wb') as f:
                    print ("Downloading: ", os.fsdecode(completeName))
                    f.write(zipcontent)
                    f.close()

print ("Script completed")
```

Iterate through the text file for URLs  
and download the zip files

```
Fetching: 2018q3_notes.zip
Downloading: /tmp/sec/2018q3_notes.zip
Fetching: 2018q2_notes.zip
Downloading: /tmp/sec/2018q2_notes.zip
Fetching: 2018q1_notes.zip
```

# 1.2 Data Preparation: Finance domain-specific corpus (A)

## [Financial Data Sets \(SEC 10K Notes\) Scratchpad v3.ipynb](#)

```
In [11]: %%time
#Loop through each ZIP file and extract specifically "txt.tsv" and save it to a different folder with zip file name prefixed

# CAUTION: This command deals with ~10GB of text files - this can fillup the /tmp folder quickly.
# This can make saving the notebook impossible till you clear diskpace (/tmp)

!cd /tmp/sec && for f in *.zip; do unzip -p -j "$f" txt.tsv > /tmp/sectxttsv/"${f%.zip}.txt.tsv"; done

CPU times: user 3.43 s, sys: 0 ns, total: 3.43 s
Wall time: 2min 41s
```

Extract "txt.tsv" form each ZIP file

```
import sys
import csv

csv.field_size_limit(sys.maxsize)
```

Out[54]: 131072



# 1.2 Data Preparation: Finance domain-specific corpus (A)

```
In [71]: %%time

# read the TSV files and extract the "Value" columns, only with rows that have a "*TextBlock" tag

import csv
import os

input_directory = '/tmp/sectxttsv'
directory = os.fsencode(input_directory)
output_directory = '/tmp/sectxt'

filecount = len(os.listdir(directory))
print ("Found " + str(filecount) + " input files in the input directory.")
counter = 0

for file in os.listdir(directory):
    filename = os.fsdecode(file)
    input_filename = os.path.join(input_directory, filename)
    output_filename = os.path.join(output_directory, "textblock-"+filename[:-4])
    counter += 1
    print("Processing source file " + str(counter) + " of " + str(filecount) + " : " + input_filename)
    print("and writing to output file : " + output_filename)

    with open(input_filename, 'r', encoding='latin-1') as tsvin, open(output_filename, 'w') as txtout:
        reader = csv.DictReader(tsvin, dialect='excel-tab')
        #reader = csv.reader(tsvin, delimiter='\t', quotechar='\"')
        #tsvin = csv.reader(tsvin, delimiter='\t')
        #txtout = csv.writer(txtout)
        rowcount = 0

        for row in reader:
            try:
                rowcount += 1
                if row['tag'][-1*len('TextBlock'):] == 'TextBlock':
                    textblock = row['value']

                    # many of the textblocks start with the word "note" -
                    # detect and strip it to avoid overrepresenting this word in the corpus
                    if textblock[:4].lower() == 'note':
                        textblock = textblock[5:]
                        #print(textblock)
                    #print(row['value'])

                    # one particular file "2009q3_notes.txt.tsv" had inline CSS -
                    # detect and strip it to avoid the meaningless CSS terms polluting the corpus
                    inlinesss1 = '/*&lt;!-- /* Style Definitions */'
                    if inlinesss1 in textblock:
                        textblock = textblock[:textblock.find(inlinesss1)]
                        #print(textblock)
                    inlinesss2 = '/*&lt;!-- /* Font Definitions */'
                    if inlinesss2 in textblock:
                        textblock = textblock[:textblock.find(inlinesss2)]
                        #print(textblock)
                    txtout.write("%s\n" % textblock)

            except csv.Error as e:
                sys.exit('line %d: %s' % (reader.line_num, e))
```

```
Found 39 input files in the input directory.
Processing source file 1 of 39 : /tmp/sectxttsv/2012q4_notes.txt.tsv
```

Read each .TSV file and extract the values corresponding to "\*TextBlock" XBRL tags

# 1.2 Data Preparation: Finance domain-specific corpus (A)

## [Financial Data Sets \(SEC 10K Notes\) Scratchpad v3.ipynb](#)

```
In [ ]: %%time

# process the source files from input folder, normalize them using NLTK, and write the normalized text files to output folder

from nltk import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import os

input_directory = '/tmp/sectxt'
directory = os.fsencode(input_directory)
output_directory = '/tmp/sectxtnorm'

filecount = len(os.listdir(directory))
print ("Found " + str(filecount) + " input files in the input directory.")
counter = 0

for file in os.listdir(directory):
    filename = os.fsdecode(file)
    input_filename = os.path.join(input_directory, filename)
    counter += 1
    print("Processing source file " + str(counter) + " of " + str(filecount) + " : " + input_filename)

# Load data
input_file = open(input_filename, 'rt')
text = input_file.read()
input_file.close()

output_sentences = []
```

Normalize the text using NLTK

# 1.2 Data Preparation: Finance domain-specific corpus (A)

## [Financial Data Sets \(SEC 10K Notes\) Scratchpad v3.ipynb](#)

```
# split into sentences
sentences = sent_tokenize(text)
for sentence in sentences:
    # split into words
    tokens = word_tokenize(sentence)
    # convert to lower case
    tokens = [w.lower() for w in tokens]
    # remove punctuation from each word
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    words = [word for word in stripped if word.isalpha()]
    # DISABLED - filter out stop words
    #stop_words = set(stopwords.words('english'))
    #words = [w for w in words if not w in stop_words]
    #print(words[:100])
    #output sentence
    output_sentence = " ".join(words)
    output_sentences.append(output_sentence)

#write output
output_filename = os.path.join(output_directory, filename)
print("Writing output file:" + output_filename)
with open(output_filename, 'w') as output_file:
    for item in output_sentences:
        output_file.write("%s\n" % item)
```

```
Found 39 input files in the input directory.
Processing source file 1 of 39 : /tmp/sectxt/textblock-2016q3_notes.txt
Writing output file:/tmp/sectxtnorm/textblock-2016q3_notes.txt
Processing source file 2 of 39 : /tmp/sectxt/textblock-2010q2_notes.txt
Writing output file:/tmp/sectxtnorm/textblock-2010q2_notes.txt
Processing source file 3 of 39 : /tmp/sectxt/textblock-2010q3_notes.txt
Writing output file:/tmp/sectxtnorm/textblock-2010q3 notes.txt
```

Normalize the text using NLTK



## 1.3 Data Preparation: Finance domain-specific corpus (B)

- The US S.E.C also maintains a database of all "Annual Report to Shareholders" (ARS) documents submitted by various publicly traded entities and makes it available through an online search tool called the EDGAR full-text search.
- Using this as a source, it was possible to list the HTML files of all available ARS documents -- which was a rather limited collection.
- Using BeautifulSoup and NLTK modules these HTML files were normalized and merged into a single corpus file and uploaded to S3.
- [\*Financial Data Sets \(SEC EDGAR ARS\) Scratchpad v1.ipynb\*](#)

# 1.3 Data Preparation: Finance domain-specific corpus (B)

## [Financial Data Sets \(SEC EDGAR ARS\) Scratchpad v1.ipynb](#)

```
import urllib.request
import urllib.error
import urllib.parse
import urllib.robotparser

#from urllib import Request, urlopen, URLError
import urllib
import os
from bs4 import BeautifulSoup

newfile = open('edgar-ars-search2.txt','w')
#print newfile

#print ("Running script.. ")
#Set variable for page to be open and url to be concatenated
url = "https://www.sec.gov"
#Search for "ARS" forms for all time, sorted by latest, 100 per page (first 100 of 138 results shown in Page 1)
#page = urllib.request.urlopen('https://searchwww.sec.gov/EDGARFSCClient/jsp/EDGAR_MainAccess.jsp?search_text=ARS%20for&sort=Date&f
#Search for "ARS" forms for all time, sorted by oldest, 50 per page (last 50 of 138 results will available)
page = urllib.request.urlopen('https://searchwww.sec.gov/EDGARFSCClient/jsp/EDGAR_MainAccess.jsp?search_text=ARS%20for&sort=Reverse

#File extension to be looked for.
#extension = ".zip"

#start pattern
start_pattern='javascript:opennew('

#Use BeautifulSoup to clean up the page
soup = BeautifulSoup(page, "lxml")
soup.prettify()

#Find all the links on the page
for anchor in soup.findAll('a', href=True):
    link = anchor['href']
    if link.startswith(start_pattern):
        link = link[19:-5]
        parts = link.split(',')
        #newfile.write(parts[1][1:-1] + ' : ' + parts [0][1:-1] + '\n')
        newfile.write(parts [0][1:-1] + '\n')
newfile.close()
```

```
!cat edgar-ars-search.txt
```

```
http://www.sec.gov/Archives/edgar/data/910679/000119312518272895/d571416dars.htm
http://www.sec.gov/Archives/edgar/data/1507964/000117184318006330/ars_083118.htm
http://www.sec.gov/Archives/edgar/data/1076682/000149315218012233/formars.htm
http://www.sec.gov/Archives/edgar/data/806172/000117152018000331/eps8043.htm
http://www.sec.gov/Archives/edgar/data/1688568/000119312518209636/d563328dars.htm
http://www.sec.gov/Archives/edgar/data/1621906/000121390018007034/ars2017_westernuranium.htm
http://www.sec.gov/Archives/edgar/data/740037/000119312518161402/d540600dars.htm
```

Scrape the SEC EDGAR Search Results webpage for HTML links – each of which is one ARS document.

# 1.3 Data Preparation: Finance domain-specific corpus (B)

```
In [37]: %%time
#Read links from .txt and extract text from each webpage to individual text file
#This is done to create persistent data
#newfile = open('seczipfiles.txt', 'r')
#for line in newfile:
#    print line + '\n'
#newfile.close()

from nltk import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import os

directory = os.fsencode('/tmp/arsnorm')

#Read through the lines in the text file and download the zip files.
#Handle exceptions and print exceptions to the console

with open('edgar-ars-search.txt', 'r') as url:
    for line in url:
        if line:
            try:
                htmlfilelink = line.rstrip()

                if line.find('/'):
                    htmlfilename = line.rsplit('/', 1)[1].rstrip()
                    print ("Fetching: ", htmlfilename)

                    response = urllib.request.urlopen(htmlfilelink)
            except urllib.error.URLError as e:
                if hasattr(e, 'reason'):
                    print ('Failed to reach server.')
                    print ('Reason: ', e.reason)
                    continue
                elif hasattr(e, 'code'):
                    print ('The server failed to fulfill the request.')
                    print ('Error code: ', e.code)
                    continue
            else:
                #extract text from HTML
                htmlcontent = response.read()
                soup = BeautifulSoup(htmlcontent,"html5lib")
                htmltext = soup.get_text(strip=True)

                output_sentences = []
```

Iterate through the URLs and open them,  
Use BeautifulSoup to read text from each HTML page



# 1.3 Data Preparation: Finance domain-specific corpus (B)

[Financial Data Sets \(SEC EDGAR ARS\) Scratchpad v1.ipynb](#)

```
# split into sentences
sentences = sent_tokenize(htmltext)
for sentence in sentences:
    # split into words
    tokens = word_tokenize(sentence)
    # convert to lower case
    tokens = [w.lower() for w in tokens]
    # remove punctuation from each word
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    words = [word for word in stripped if word.isalpha()]
    # DISABLED - filter out stop words
    #stop_words = set(stopwords.words('english'))
    #words = [w for w in words if not w in stop_words]
    #print(words[:100])
    #output sentence
    output_sentence = " ".join(words)
    output_sentences.append(output_sentence)

#write output file
completeName = os.path.join(directory, os.fsencode(htmlfilename+'.txt'))
with open (completeName, 'w') as f:
    print ("Writing to: ", os.fsdecode(completeName))
    for item in output_sentences:
        f.write("%s\n" % item)
    #f.write(htmltext)
    #f.close()
```

Use NLTK to normalize the text

```
Fetching: d571416dars.htm
Writing to: /tmp/arsnorm/d571416dars.htm.txt
Fetching: ars_083118.htm
Writing to: /tmp/arsnorm/ars_083118.htm.txt
Fetching: formars.htm
```

# 2

## 2. Training and Tuning the Model

1. Training a BlazingText Model
2. Repeating the Training using different combinations of corpus  
*(Example: English only, English + SEC10K, English+SECARS, with and without stop words, etc)*
3. Using HyperParameter Tuning to find optimal parameters

## 2. Training and Tuning the Model

### [BlazingText Trainer Scratchpad v4.ipynb](#)

```
In [1]: import sagemaker, boto3, json
        from sagemaker import get_execution_role

        sess = sagemaker.Session()

        role = get_execution_role()

        bucket = 'sagemaker-wipro-raghuvv2'
        input_prefix = 'sec/lm1bsecmerged2' # existing input location of training data
        output_prefix = 'lm1bsecmerged2output-dim75' # NEW output location for model artifacts

        s3_train_data = 's3://{}/{}'.format(bucket, input_prefix)
        print (s3_train_data)

        s3_output_location = 's3://{}/{}'.format(bucket, output_prefix)
        print (s3_output_location)

        region_name = boto3.Session().region_name
        container = sagemaker.amazon.amazon_estimator.get_image_uri(region_name, "blazingtext", "latest")
        print('Using SageMaker BlazingText container: {} ({}).format(container, region_name))

        s3://sagemaker-wipro-raghuvv2/sec/lm1bsecmerged2
        s3://sagemaker-wipro-raghuvv2/lm1bsecmerged2output-dim75
        Using SageMaker BlazingText container: 825641698319.dkr.ecr.us-east-2.amazonaws.com/blazingtext:latest (us-east-2)
```



## 2. Training and Tuning the Model

### [BlazingText Trainer Scratchpad v4.ipynb](#)

```
In [2]: bt_model = sagemaker.estimator.Estimator(container,
                                                role,
                                                train_instance_count=1,
                                                train_instance_type='ml.c4.2xlarge', # Use of ml.p3.2xlarge is highly recommended for hig
                                                train_volume_size = 30,
                                                train_max_run = 360000,
                                                input_mode= 'File',
                                                output_path=s3_output_location,
                                                sagemaker_session=sess)

bt_model.set_hyperparameters(mode="skipgram",
                             epochs=5,
                             min_count=5,
                             sampling_threshold=0.0001,
                             learning_rate=0.05,
                             window_size=5,
                             vector_dim=75, # original --100
                             negative_samples=5,
                             subwords=True, # Enables learning of subword embeddings for OOV word vector generation
                             min_char=3, # min length of char ngrams
                             max_char=6, # max length of char ngrams
                             batch_size=11, # = (2*window_size + 1) (Preferred. Used only if mode is batch_skipgram)
                             evaluation=True)# Perform similarity evaluation on WS-353 dataset at the end of training

train_data = sagemaker.session.s3_input(s3_train_data, distribution='FullyReplicated',
                                       content_type='text/plain', s3_data_type='S3Prefix')
data_channels = {'train': train_data}
```

# 2. Training and Tuning the Model

## [BlazingText Trainer Scratchpad v4.ipynb](#)

```
In [3]: %%time
bt_model.fit(inputs=data_channels, logs=True)
```

```
INFO:sagemaker:Creating training-job with name: blazingtext-2018-10-23-02-57-26-800
```

```
2018-10-23 02:57:26 Starting - Starting the training job...
2018-10-23 02:57:27 Starting - Launching requested ML instances.....
2018-10-23 02:58:30 Starting - Preparing the instances for training...
2018-10-23 02:59:11 Downloading - Downloading input data.....
2018-10-23 03:01:17 Training - Training image download completed. Training in progress.
```

```
Arguments: train
```

```
[10/23/2018 03:01:17 WARNING 140557515609920] Loggers have already been setup.
```

```
[10/23/2018 03:01:17 WARNING 140557515609920] Loggers have already been setup.
```

```
[10/23/2018 03:01:17 INFO 140557515609920] nvidia-smi took: 0.025171995163 secs to identify 0 gpus
```

```
[10/23/2018 03:01:17 INFO 140557515609920] Running single machine CPU BlazingText training using skipgram mode.
```

```
[10/23/2018 03:01:17 INFO 140557515609920] Processing /opt/ml/input/data/train/lm1bsecmergedfile2 . File size: 6219 MB
```

```
Read 10M words
```

```
Read 20M words
```

```
Read 30M words
```

```
Read 40M words
```

```
##### Alpha: 0.0000 Progress: 100.00% Million Words/sec: 0.69 #####
```

```
Training finished.
```

```
Average throughput in Million words/sec: 0.69
```

```
Total training time in seconds: 7984.04
```

```
Evaluating word embeddings....
```

```
2018-10-23 05:16:46 Uploading - Uploading generated training modelVectors read from: /opt/ml/model/vectors.txt
```

```
},
"mean_rho": 0.5861497957171232
```

```
}
[10/23/2018 05:16:42 INFO 140557515609920] #mean_rho: 0.586149795717
```

```
2018-10-23 05:19:18 Completed - Training job completed
```

```
Billable seconds: 8408
```

```
CPU times: user 13.9 s, sys: 660 ms, total: 14.5 s
```

```
Wall time: 2h 22min 7s
```



## 2. Training and Tuning the Model

### Tuning the model:

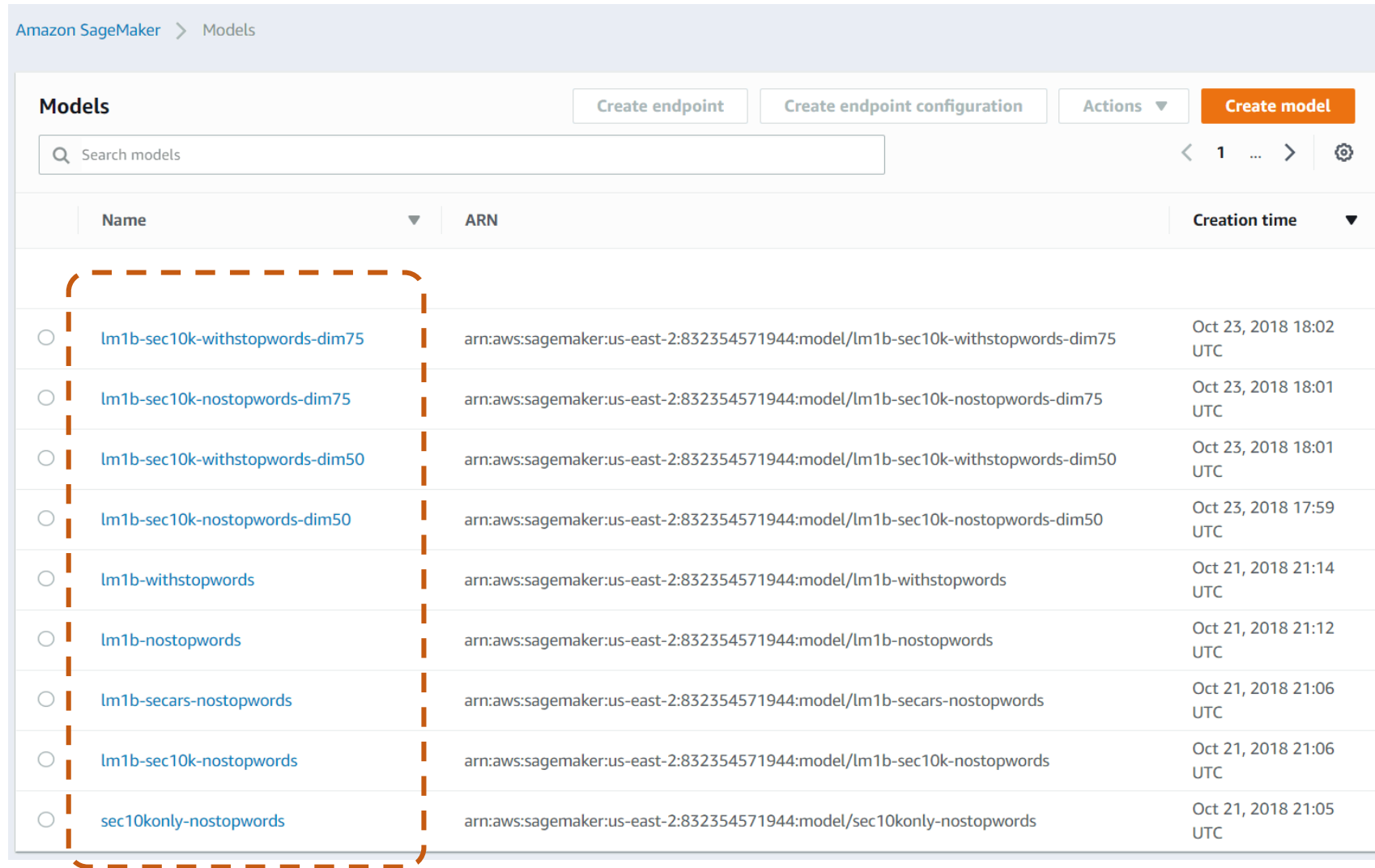
- a. **Manual** - iteration by repeating the training with different input corpus and different values of hyperparameters like vector dimension
- b. **Automatic** – using SageMaker’s hyperparameter tuning jobs to automatically run the training with a given input but using different combinations of hyperparameter values within configured constraints



## 2. Training and Tuning the Model

Tuning the model:

- a. **Manual** - iteration by repeating the training with different input corpus and different values of hyperparameters like vector dimension



Amazon SageMaker > Models

Models Create endpoint Create endpoint configuration Actions Create model

Search models

Name	ARN	Creation time
<input type="radio"/> <a href="#">lm1b-sec10k-withstopwords-dim75</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-sec10k-withstopwords-dim75	Oct 23, 2018 18:02 UTC
<input type="radio"/> <a href="#">lm1b-sec10k-nostopwords-dim75</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-sec10k-nostopwords-dim75	Oct 23, 2018 18:01 UTC
<input type="radio"/> <a href="#">lm1b-sec10k-withstopwords-dim50</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-sec10k-withstopwords-dim50	Oct 23, 2018 18:01 UTC
<input type="radio"/> <a href="#">lm1b-sec10k-nostopwords-dim50</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-sec10k-nostopwords-dim50	Oct 23, 2018 17:59 UTC
<input type="radio"/> <a href="#">lm1b-withstopwords</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-withstopwords	Oct 21, 2018 21:14 UTC
<input type="radio"/> <a href="#">lm1b-nostopwords</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-nostopwords	Oct 21, 2018 21:12 UTC
<input type="radio"/> <a href="#">lm1b-secars-nostopwords</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-secars-nostopwords	Oct 21, 2018 21:06 UTC
<input type="radio"/> <a href="#">lm1b-sec10k-nostopwords</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-sec10k-nostopwords	Oct 21, 2018 21:06 UTC
<input type="radio"/> <a href="#">sec10konly-nostopwords</a>	arn:aws:sagemaker:us-east-2:832354571944:model/sec10konly-nostopwords	Oct 21, 2018 21:05 UTC

## 2. Training and Tuning the Model

Tuning the model:

- b. Automatic** – using SageMaker’s hyperparameter tuning jobs to automatically run the training with a given input but using different combinations of hyperparameter values within configured constraints

Amazon SageMaker > Hyperparameter tuning jobs

**Hyperparameter tuning jobs** Clone Add/Edit tags Create hyperparameter tuning job

Search hyperparameter tuning jobs

Name	Status	Training completed/total	Creation time	Duration
<a href="#">hyperparameter-tuning-job-5</a>	Completed	25 / 25	Oct 24, 2018 23:34 UTC	11 hours
<a href="#">hyperparameter-tuning-job-4</a>	Completed	25 / 25	Oct 24, 2018 01:20 UTC	20 hours

Amazon SageMaker > Models

**Models** Create endpoint Create endpoint configuration Actions Create model

Search models

Name	ARN	Creation time
<a href="#">lm1b-sec10k-nostopwords-hypertune2-dim35</a>	arn:aws:sagemaker:us-east-2:832354571944:model/lm1b-sec10k-nostopwords-hypertune2-dim35	Oct 24, 2018 00:40 UTC

## 3

# 3. Deploying the Model & Inference

## Phrase Pair Comparison v2.ipynb

```
In [1]: # setup the input and output files

# MODELS FILE -- do not change this
# this reference CSV file contains metadata about the various ML models, their endpoint configs,
# vector dimensions, whether stop words were removed, and the suggested similarity_threshold value
modelsfile = 'mymodels.txt'

# SELECTED MODEL -- this variable simplifies and replaces the interactive selection (now disabled) in the middle of the notebook
# specify one of the 12 trained models from this list
# 1.  lm1b-nostopwords
# 2.  lm1b-withstopwords
# 3.  lm1b-secars-nostopwords
# 4.  sec10konly-nostopwords
# 5.  lm1b-sec10k-nostopwords
# 6.  sec10konly-withstopwords
# 7.  lm1b-sec10k-withstopwords
# 8.  lm1b-sec10k-nostopwords-dim50
# 9.  lm1b-sec10k-withstopwords-dim50
# 10. lm1b-sec10k-nostopwords-dim75
# 11. lm1b-sec10k-withstopwords-dim75
# 12. lm1b-sec10k-nostopwords-hypertune2-dim35
selectedmodels = ['lm1b-sec10k-withstopwords-dim50']

# PHRASE PAIRS FILE
# this is the INPUT file in CSV format contains pairs of input phrases that need to be compared
# Header: phrase1, phrase2
phrasepairsfile = 'phrasepairs3.txt'

# PHRASE PAIR COMPARISON FILE
# this is the OUTPUT file in CSV format that will contain the similar_bool = True/False
# output for each pair of phrases in the input file
# Header: phrase1, phrase2, similar_bool
paircomparisonfile = 'phrasepaircomparison9.txt'

# CUSTOM SIMILARITY THRESHOLD
# Use this if you want to override the suggested values of similarity_threshold with your own custom value.
# Any pair of phrases whose vectors' cosine similarity equals or exceeds the similarity_threshold are considered similar
# and similar_bool is set to True in the output
use_custom_similarity_threshold = False # default/recommended is False
custom_similarity_threshold = 0.685 # typical threshold values that give good results are in the range of 0.55 to 0.7
```



# 3. Deploying the Model & Inference

```
In [4]: import pandas, time, json, csv
        from pivottablejs import pivot_ui
        import sagemaker, boto3
        from nltk import sent_tokenize
        from nltk.tokenize import word_tokenize
        from nltk.corpus import stopwords
        import string
        import os
        import numpy as np

        # data processing functions

        def sentence_normalize(sentence, filterstopwords=True):
            # split into words
            tokens = word_tokenize(sentence)
            # convert to lower case
            tokens = [w.lower() for w in tokens]
            # remove punctuation from each word
            table = str.maketrans('', '', string.punctuation)
            stripped = [w.translate(table) for w in tokens]
            # remove remaining tokens that are not alphabetic
            words = [word for word in stripped if word.isalpha()]
            # filter out stop words
            if filterstopwords:
                stop_words = set(stopwords.words('english'))
                words = [w for w in words if not w in stop_words]
            #output sentence
            output_sentence = " ".join(words)
            return output_sentence
```

### 3. Deploying the Model & Inference

```
def wordvector(word,endpointname):
    smruntimeclient = boto3.client('sagemaker-runtime')
    payload = {"instances" : [word]}
    btrresponse = smruntimeclient.invoke_endpoint(
        EndpointName=endpointname,
        Body=json.dumps(payload),
        ContentType='application/json',
        Accept='application/json'
    )
    return json.loads(btrresponse['Body'].read())[0]['vector']

def phrasevector(phrase,endpointname,vector_dim):
    x = np.zeros((vector_dim,))
    for w in phrase.split():
        x = np.add(x,wordvector(w,endpointname))
    return x

def similarity(v1, v2):
    n1 = np.linalg.norm(v1)
    n2 = np.linalg.norm(v2)
    return np.dot(v1, v2) / n1 / n2

def compare_phrases(p1,p2,endpointname,vector_dim):
    return similarity(phrasevector(p1,endpointname,vector_dim),phrasevector(p2,endpointname,vector_dim))
```

# 3. Deploying the Model & Inference

```
In [5]: input_phrase_pairs=[]

print ('reading input phrase pairs from:', phrasepairsfile)

with open(phrasepairsfile,'r', encoding='latin-1') as tsvin:
    reader = csv.DictReader(tsvin, dialect='excel')
    rowcount = 0
    input_phrase_pairs=[]
    for row in reader:
        input_phrase_pairs.append(row)
        print (input_phrase_pairs[rowcount])
        rowcount += 1

print ('number of phrase pairs in file:', rowcount)
```

```
reading input phrase pairs from: phrasepairs3.txt
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'birth day')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'birth date')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'born on')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'year of incorporation')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'established in')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'founded in the year')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'incorporated in')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'home address')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'place of residence')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'residing at')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'residence address')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'house address')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'financial statement')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'balance sheet')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'high precipitation')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'heavy rainfall')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'counterfeit goods')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'fake products')])
OrderedDict([('phrase1', 'date of birth'), ('phrase2', 'shares')])
```

# 3. Deploying the Model & Inference

## Phrase Pair Comparison v2.ipynb

```
In [8]: %%time

print("writing to :", paircomparisonfile)

smclient = boto3.client('sagemaker')

modeldicts = input_models[input_models['model'].isin(selectedmodels)].to_dict(orient='records')

with open(paircomparisonfile,'w', encoding='latin-1') as csvout:
    csvwriter = csv.writer(csvout)
    # use this line if detailed output is desired
    # csvwriter.writerow(['model', 'vector_dim', 'removestopwords', 'phrase_pair_file', 'phrase1', 'phrase1_normalized', 'phrase2']
    csvwriter.writerow(['phrase1', 'phrase2', 'similar_bool'])

    for modeldict in modeldicts:

        modelname = modeldict['model']
        endpointname = modeldict['endpointname']
        endpointconfigname = modeldict['endpointconfigname']
        vector_dim = modeldict['vector_dim']
        removestopwords = modeldict['removestopwords']
        if use_custom_similarity_threshold==True:
            similarity_threshold = custom_similarity_threshold
            print ('using CUSTOM similarity threshold:', similarity_threshold)
        else:
            similarity_threshold = modeldict['similarity_threshold']
            print ('using similarity threshold:', similarity_threshold)

        print ("creating endpoint:", endpointname)

    btendpoint = smclient.create_endpoint(EndpointName=endpointname, EndpointConfigName=endpointconfigname)
```



# 3. Deploying the Model & Inference

## Phrase Pair Comparison v2.ipynb

```
while True:
    endpointstatus = smclient.describe_endpoint(EndpointName=endpointname)['EndpointStatus']
    #print(endpointstatus)
    if endpointstatus == 'InService':
        print(endpointstatus)
        break
    for i in range(3):
        print('.', end='', flush=True)
        time.sleep(3)

print ("finished creating endpoint:", endpointname)

print ("invoking endpoint:", endpointname)
for phrasepair in input_phrase_pairs:
    phrase1_normalized = sentence_normalize(phrasepair['phrase1'],removestopwords)
    phrase2_normalized = sentence_normalize(phrasepair['phrase2'],removestopwords)
    try:
        cos_similarity = compare_phrases(phrase1_normalized,phrase2_normalized,endpointname,vector_dim)
    except:
        cos_similarity = 'error'

    if cos_similarity >= similarity_threshold:
        similar_bool_output = True
    else:
        similar_bool_output = False

    # use this line if detailed output is desired
    # print (",".join([modelname, str(vector_dim), str(removestopwords), phrasepairsfile, phrasepair['phrase1'], phrase1_n
    # csvwriter.writerow([modelname, str(vector_dim), str(removestopwords), phrasepairsfile, phrasepair['phrase1'], phrase
    csvwriter.writerow([phrasepair['phrase1'], phrasepair['phrase2'], str(similar_bool_output)])

#delete endpoint
print ("deleting endpoint:", endpointname)
smclient.delete_endpoint(EndpointName=endpointname)

print ("completed writing comparison file")
```

```
writing to : phrasepaircomparison9.txt
using similarity threshold: 0.695
creating endpoint: lm1b-sec10k-withstopwords-dim50
.....
InService
finished creating endpoint: lm1b-sec10k-withstopwords-dim50
invoking endpoint: lm1b-sec10k-withstopwords-dim50
deleting endpoint: lm1b-sec10k-withstopwords-dim50
completed writing comparison file
CPU times: user 1min 30s, sys: 3.31 s, total: 1min 33s
Wall time: 9min 17s
```

# 4

## 4. Results / Evaluation

### [Phrase Pair Comparison v2.ipynb](#)

The notebook will accept the input as CSV (phrase1, phrase2) and provide another output file as CSV (phrase1, phrase2, similar\_bool)

By default the notebook uses the model '**lm1b-sec10k-withstopwords-dim50**' -- which was the one that was trained with both the LM-1-Billion-Words and the SEC 10K notes corpus, while retaining the stop words, with a **vector dimension of 50**. By default the notebook uses a "similarity threshold" of **0.695** that I assigned to this model. These defaults can be changed in the first cell of the notebook before running the rest of the notebook.

# 4. Results / Evaluation

Manually created list of phrases and similarity groups:

group	phrase
1	date of birth
1	birth day
1	birth date
1	born on
2	year of incorporation
2	established in
2	founded in the year
2	incorporated in
3	home address
3	place of residence
3	residing at
3	residence address
3	house address
4	financial statement
4	balance sheet
5	high precipitation
5	heavy rainfall
6	counterfeit goods
6	fake products

7	shares
7	common stock
8	solar power
8	renewable energy
9	driverless cars
9	self-driving cars
9	autonomous vehicles
10	take a nap
10	fall asleep
10	go to bed
11	flunk the test
11	fail the exam
12	express shipping
12	expedited delivery
13	Income from operations
13	Operating revenues
14	latest style
14	modern fashion
14	current trend
14	in vogue
15	late in the night
15	after dark
15	past midnight

# 4. Results / Evaluation

## Phrase pairs file used as input:

- 42x41 = **1722 total phrase pairs** generated from the 42 phrases
- 90 pairs (phrases from same group) are meant to be similar
- 1632 pairs (from different groups) are meant to be dissimilar

## Output file:

A boolean value is reported against each phrase pair indicating whether that is evaluated to be a “similar” or not, as per the selected model and similarity threshold.

```
jupyter phrasepairs3.txt 10/31/2018
File Edit View Language
1 phrase1,phrase2
2 date of birth,birth day
3 date of birth,birth date
4 date of birth,born on
5 date of birth,year of incorporation
6 date of birth,established in
7 date of birth,founded in the year
8 date of birth,incorporated in
9 date of birth,home address
10 date of birth,place of residence
11 date of birth,residing at
12 date of birth,residence address
13 date of birth,house address
14 date of birth,financial statement
15 date of birth,balance sheet
16 date of birth,high precipitation
17 date of birth,heavy rainfall
18 date of birth,counterfeit goods
19 date of birth,fake products
20 date of birth,shares
21 date of birth,common stock
22 date of birth,solar power
23 date of birth,renewable energy
24 date of birth,driverless cars
25 date of birth,self-driving cars
26 date of birth.autonomous vehicles
```

```
jupyter phrasepaircomparison6.txt 10/31/2018
File Edit View Language
1 phrase1,phrase2,similar_bool
2 date of birth,birth day,True
3 date of birth,birth date,True
4 date of birth,born on,True
5 date of birth,year of incorporation,True
6 date of birth,established in,False
7 date of birth,founded in the year,False
8 date of birth,incorporated in,False
9 date of birth,home address,False
10 date of birth,place of residence,False
11 date of birth,residing at,False
12 date of birth,residence address,False
13 date of birth,house address,False
14 date of birth,financial statement,False
15 date of birth,balance sheet,False
16 date of birth,high precipitation,False
17 date of birth,heavy rainfall,False
18 date of birth,counterfeit goods,False
19 date of birth,fake products,False
20 date of birth,shares,False
21 date of birth,common stock,False
22 date of birth,solar power,False
23 date of birth,renewable energy,False
24 date of birth,driverless cars,False
25 date of birth,self-driving cars,False
26 date of birth.autonomous vehicles.False
```



# 4. Results / Evaluation

Analysis of the output (for one specific run of a model and selected threshold) –

```
In [8]: #summary of evaluation as pivot
df = pandas.read_csv('lm1b-sec10k-withstopwords-phrase1.txt')

from pivottablejs import pivot_ui

pivot_ui(df[df['same_phrase']==False],
         rows=['phrase1_group'],
         cols=['phrase2_group'],
         aggregatorName='Average',
         vals=['cos_similarity'],
         rendererName='Heatmap')
```

Out[8]:

	phrase2_group	1	2	3	4	5	6	7	8	9	10	11	12	Totals
phrase1_group														
1		0.74	0.51	0.20	0.24	0.25	0.22	0.14	0.34	0.22	0.22	0.26	0.23	0.28
2		0.51	0.64	0.31	0.29	0.27	0.28	0.22	0.34	0.31	0.34	0.33	0.35	0.34
3		0.20	0.31	0.44	0.32	0.34	0.25	0.44	0.24	0.22	0.31	0.33	0.21	0.30
4		0.24	0.29	0.32	0.54	0.27	0.32	0.25	0.34	0.35	0.38	0.25	0.21	0.32
5		0.25	0.27	0.34	0.27	0.65	0.32	0.32	0.24	0.16	0.27	0.46	0.24	0.30
6		0.22	0.28	0.25	0.32	0.32	0.69	0.15	0.38	0.39	0.33	0.25	0.21	0.33
7		0.14	0.22	0.44	0.25	0.32	0.15	0.56	0.21	0.15	0.29	0.31	0.14	0.25
8		0.34	0.34	0.24	0.34	0.24	0.38	0.21	0.47	0.34	0.42	0.19	0.29	0.33
9		0.22	0.31	0.22	0.35	0.16	0.39	0.15	0.34	0.73	0.31	0.22	0.20	0.28
10		0.22	0.34	0.31	0.38	0.27	0.33	0.29	0.42	0.31	0.51	0.26	0.26	0.34
11		0.26	0.33	0.33	0.25	0.46	0.25	0.31	0.19	0.22	0.26	0.53	0.24	0.29
12		0.23	0.35	0.21	0.21	0.24	0.21	0.14	0.29	0.20	0.26	0.24	0.51	0.26
Totals		0.28	0.34	0.30	0.32	0.30	0.33	0.25	0.33	0.28	0.34	0.29	0.26	0.31

When we use the default ML model in my notebook (**lm1b-sec10k-withstopwords-dim50**) to compare these phrase vectors using a **similarity threshold of 0.695**, we get:

- 78 pairs identified as similar and 1644 pairs as dissimilar.
- This includes 16 false positives and 28 false negatives. False positives included pairs like "fall asleep" and "after dark" - which might appear in similar contexts.

# 4. Results / Evaluation

Studied how the cosine similarity varies when phrases are picked from within the same group ("intra-group similarity" : high similarity expected) versus when they are picked from different groups ("inter-group similarity" : low similarity expected).

same_group	(All)													
same_phrase	FALSE													
Average of cos_similarity	Phrase_group_2													
Phrase_group_1		1	2	3	4	5	6	7	8	9	10	11	12	Grand Total
1		0.735946317	0.50817738	0.197463222	0.238111874	0.251804225	0.217759011	0.137841364	0.34402418	0.221911948	0.224140275	0.264705243	0.230115142	0.282884097
2		0.50817738	0.643831398	0.306312908	0.286471734	0.270000469	0.279502366	0.224531844	0.343064811	0.310466139	0.344093794	0.326562162	0.349549981	0.342770905
3		0.197463222	0.306312908	0.440907547	0.322755104	0.341766904	0.246789676	0.438738729	0.237612435	0.21664868	0.309992739	0.328292769	0.212007658	0.295931933
4		0.238111874	0.286471734	0.322755104	0.5376954	0.266214313	0.318986008	0.246712237	0.34436502	0.353927204	0.376149754	0.254955849	0.214039403	0.316210701
5		0.251804225	0.270000469	0.341766904	0.266214313	0.651359173	0.317086609	0.31634583	0.244790073	0.156688179	0.272885496	0.461261233	0.241551741	0.30488511
6		0.217759011	0.279502366	0.246789676	0.318986008	0.317086609	0.685813095	0.149523062	0.376800339	0.39091283	0.331135336	0.245189293	0.206407548	0.326007708
7		0.137841364	0.224531844	0.438738729	0.246712237	0.31634583	0.149523062	0.561117252	0.207823023	0.149382176	0.288969623	0.305644906	0.141319858	0.251475073
8		0.34402418	0.343064811	0.237612435	0.34436502	0.244790073	0.376800339	0.207823023	0.469217056	0.336024974	0.422599553	0.191366329	0.289877802	0.325123792
9		0.221911948	0.310466139	0.21664868	0.353927204	0.156688179	0.39091283	0.149382176	0.336024974	0.732608993	0.311788786	0.219112888	0.204764338	0.284548005
10		0.224140275	0.344093794	0.309992739	0.376149754	0.272885496	0.331135336	0.288969623	0.422599553	0.311788786	0.507239392	0.259575943	0.263748214	0.335031124
11		0.264705243	0.326562162	0.328292769	0.254955849	0.461261233	0.245189293	0.305644906	0.191366329	0.219112888	0.259575943	0.527451658	0.243789187	0.290776534
12		0.230115142	0.349549981	0.212007658	0.214039403	0.241551741	0.206407548	0.141319858	0.289877802	0.204764338	0.263748214	0.243789187	0.508488221	0.25897597
<b>Grand Total</b>		<b>0.282884097</b>	<b>0.342770905</b>	<b>0.295931933</b>	<b>0.316210701</b>	<b>0.30488511</b>	<b>0.326007708</b>	<b>0.251475073</b>	<b>0.325123792</b>	<b>0.284548005</b>	<b>0.335031124</b>	<b>0.290776534</b>	<b>0.25897597</b>	<b>0.305420051</b>

# 4. Results / Evaluation

## Summarizing the results of one run across all models

	selected_model	removestopwords	vector_dim	intergroup_mean_similarity	intragroup_mean_similarity	phrase_group_distinction
1	lm1b-nostopwords	TRUE	100	0.282850412	0.655047415	0.372197002
2	lm1b-withstopwords	FALSE	100	0.335288274	0.69900345	0.363715176
3	lm1b-secars-nostopwords	TRUE	100	0.272254062	0.642257696	0.370003634
4	sec10konly-nostopwords	TRUE	100	0.305154081	0.518378851	0.21322477
5	lm1b-sec10k-nostopwords	TRUE	100	0.280681625	0.629299343	0.348617718
6	sec10konly-withstopwords	FALSE	100	0.350239696	0.575864411	0.225624715
7	lm1b-sec10k-withstopwords	FALSE	100	0.329241072	0.6823853	0.353144228
8	lm1b-sec10k-nostopwords-dim50	TRUE	50	0.323491151	0.684530934	0.361039783
9	lm1b-sec10k-withstopwords-dim50	FALSE	50	0.384631145	0.743129087	0.358497941
10	lm1b-sec10k-nostopwords-dim75	TRUE	75	0.294148292	0.645727607	0.351579315
11	lm1b-sec10k-withstopwords-dim75	FALSE	75	0.351164489	0.706446647	0.355282158
12	lm1b-sec10k-nostopwords-hypertune2-dim35	TRUE	35	0.352010871	0.709664651	0.35765378

```
In [9]: #summary of evaluation - overall metrics
intergroup_mean_similarity = df[(df['same_phrase']==False) & (df['same_group']==False)]['cos_similarity'].mean()
intragroup_mean_similarity = df[(df['same_phrase']==False) & (df['same_group']==True)]['cos_similarity'].mean()
phrase_group_distinction = intragroup_mean_similarity-intergroup_mean_similarity

print ("intergroup_mean_similarity:",intergroup_mean_similarity)
print ("intragroup_mean_similarity:",intragroup_mean_similarity)
print ("phrase group distinction:",phrase_group_distinction)

intergroup_mean_similarity: 0.2841687055976957
intragroup_mean_similarity: 0.5684828228858992
phrase group distinction: 0.28431411728820355
```



# 4. Results / Evaluation

The same input file was passed to all the models in my list and the output was analyzed to see how the models fare in terms of false positives and false negatives as opposed to returning the “correct” result.

	A	F	H	I	J	K	L	M	N	O	P
1	model	phrase1_normalized	phrase2_normalized	similar_bool_input	cos_similarity	similarity_threshold	similar_bool_output	false_positive	false_negative	correct_positive	correct_negative
6877	sec10konly-nostopwords	past midnight	go bed	FALSE	0.396467797	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6878	sec10konly-nostopwords	past midnight	flunk test	FALSE	0.368740483	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6879	sec10konly-nostopwords	past midnight	fail exam	FALSE	0.441440032	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6880	sec10konly-nostopwords	past midnight	express shipping	FALSE	0.389333787	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6881	sec10konly-nostopwords	past midnight	expedited delivery	FALSE	0.396447054	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6882	sec10konly-nostopwords	past midnight	income operations	FALSE	0.253140702	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6883	sec10konly-nostopwords	past midnight	operating revenues	FALSE	0.263389211	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6884	sec10konly-nostopwords	past midnight	latest style	FALSE	0.53042473	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6885	sec10konly-nostopwords	past midnight	modern fashion	FALSE	0.459967685	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6886	sec10konly-nostopwords	past midnight	current trend	FALSE	0.401974899	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6887	sec10konly-nostopwords	past midnight	vogue	FALSE	0.261112354	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6888	sec10konly-nostopwords	past midnight	late night	TRUE	0.569765617	0.685	FALSE	FALSE	TRUE	FALSE	FALSE
6889	sec10konly-nostopwords	past midnight	dark	TRUE	0.221954408	0.685	FALSE	FALSE	TRUE	FALSE	FALSE
6890	lm1b-sec10k-nostopwords	date birth	birth day	TRUE	0.80160878	0.685	TRUE	FALSE	FALSE	TRUE	FALSE
6891	lm1b-sec10k-nostopwords	date birth	birth date	TRUE	1	0.685	TRUE	FALSE	FALSE	TRUE	FALSE
6892	lm1b-sec10k-nostopwords	date birth	born	TRUE	0.565288792	0.685	FALSE	FALSE	TRUE	FALSE	FALSE
6893	lm1b-sec10k-nostopwords	date birth	year incorporation	FALSE	0.521534824	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6894	lm1b-sec10k-nostopwords	date birth	established	FALSE	0.211179167	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6895	lm1b-sec10k-nostopwords	date birth	founded year	FALSE	0.400194712	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6896	lm1b-sec10k-nostopwords	date birth	incorporated	FALSE	0.251563611	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6897	lm1b-sec10k-nostopwords	date birth	home address	FALSE	0.357606297	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6898	lm1b-sec10k-nostopwords	date birth	place residence	FALSE	0.391213589	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6899	lm1b-sec10k-nostopwords	date birth	residing	FALSE	0.198748686	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6900	lm1b-sec10k-nostopwords	date birth	residence address	FALSE	0.373420116	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6901	lm1b-sec10k-nostopwords	date birth	house address	FALSE	0.368268468	0.685	FALSE	FALSE	FALSE	FALSE	TRUE
6902	lm1b-sec10k-nostopwords	date birth	financial statement	FALSE	0.371300673	0.685	FALSE	FALSE	FALSE	FALSE	TRUE

Expected similarity

Similarity as per model

Categorization as correct / incorrect etc



## 4. Results / Evaluation

A rudimentary “score” was assigned to each model based on the number of “correct” and “incorrect” responses they returned for the given input file.

This in turn was translated into an “error rate” – which was simply a measure on number of incorrect responses normalized against the total number of input phrase pairs.

This gave an indication of which model is performing well.

	false_positive	false_negative	correct_positive	correct_negative	correct_result	score	missed matches	error_rate	
1 lm1b-nostopwords	2	26	64	1630	1694	1666	0.983739837	0.29	0.311
2 lm1b-withstopwords	6	18	72	1626	1698	1674	0.986062718	0.20	0.267
3 lm1b-secars-nostopwords	0	26	64	1632	1696	1670	0.984901278	0.29	0.289
4 sec10konly-nostopwords	10	48	42	1622	1664	1606	0.966318235	0.53	0.644
5 lm1b-sec10k-nostopwords	2	44	46	1630	1676	1630	0.973286876	0.49	0.511
6 sec10konly-withstopwords	0	56	34	1632	1666	1610	0.967479675	0.62	0.622
7 lm1b-sec10k-withstopwords	14	26	64	1618	1682	1642	0.976771196	0.29	0.444
8 lm1b-sec10k-nostopwords-dim50	4	46	44	1628	1672	1622	0.970963995	0.51	0.556
9 lm1b-sec10k-withstopwords-dim50	12	32	58	1620	1678	1634	0.974448316	0.36	0.489
10 lm1b-sec10k-nostopwords-dim75	0	46	44	1632	1676	1630	0.973286876	0.51	0.511
11 lm1b-sec10k-withstopwords-dim75	14	28	62	1618	1680	1638	0.975609756	0.31	0.467
12 lm1b-sec10k-nostopwords-hypertune2-dim35	4	46	44	1628	1672	1622	0.970963995	0.51	0.556
	68	442			20154	19644	0.975319396		

# 4. Results / Evaluation

The same input file was evaluated against all models – with different values of similarity threshold

And a “heatmap” was generated manually on excel to get a sense of where the “error rate” is lowest.

model	optimised	0.5	0.51	0.52	0.53	0.54	0.55	0.56	0.57	0.58	0.59	0.6	0.61	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.69	0.7	0.75	0.8	0.85	0.9
1 lm1b-nostopwords	0.585	0.556	0.467	0.422	0.400	0.422	0.422	0.444	0.333	0.311	0.311	0.356	0.400	0.400	0.422	0.444	0.467	0.489	0.511	0.578	0.600	0.622	0.733	0.778	0.978	0.978
2 lm1b-withstopwords	0.61	1.511	1.222	1.022	0.933	0.822	0.711	0.556	0.533	0.489	0.444	0.333	0.267	0.333	0.333	0.333	0.356	0.356	0.400	0.444	0.467	0.467	0.600	0.756	0.978	0.978
3 lm1b-secars-nostopwords	0.55	0.511	0.400	0.311	0.356	0.311	0.289	0.311	0.311	0.356	0.422	0.422	0.444	0.444	0.467	0.467	0.489	0.511	0.578	0.600	0.600	0.622	0.733	0.778	0.956	0.956
4 sec10konly-nostopwords	0.565	1.089	1.067	0.889	0.867	0.800	0.711	0.644	0.644	0.689	0.667	0.644	0.667	0.667	0.689	0.711	0.733	0.756	0.800	0.800	0.800	0.822	0.844	0.867	0.956	0.956
5 lm1b-sec10k-nostopwords	0.625	1.067	0.933	0.844	0.778	0.689	0.644	0.644	0.644	0.622	0.600	0.533	0.533	0.511	0.511	0.533	0.533	0.533	0.556	0.533	0.556	0.556	0.667	0.800	0.933	0.933
6 sec10konly-withstopwords	0.64	2.089	1.867	1.622	1.467	1.178	1.133	1.067	0.822	0.800	0.689	0.644	0.622	0.644	0.644	0.622	0.644	0.711	0.756	0.756	0.778	0.800	0.822	0.867	0.956	0.956
7 lm1b-sec10k-withstopwords	0.63	1.711	1.689	1.556	1.222	1.133	1.022	0.933	0.778	0.689	0.622	0.511	0.467	0.422	0.444	0.422	0.467	0.489	0.511	0.533	0.511	0.533	0.689	0.778	0.933	0.933
8 lm1b-sec10k-nostopwords-dim50	0.7	2.533	2.111	1.933	1.644	1.467	1.378	1.222	1.222	1.156	1.022	0.911	0.800	0.733	0.689	0.644	0.644	0.600	0.578	0.578	0.578	0.556	0.578	0.644	0.933	0.933
9 lm1b-sec10k-withstopwords-dim50	0.695	4.067	3.667	3.422	3.067	2.800	2.400	2.267	2.022	1.756	1.644	1.489	1.378	1.200	1.133	1.044	0.889	0.822	0.711	0.556	0.489	0.489	0.489	0.600	0.933	0.933
10 lm1b-sec10k-nostopwords-dim75	0.685	1.422	1.267	1.133	1.044	0.978	0.889	0.867	0.778	0.644	0.533	0.600	0.600	0.578	0.533	0.511	0.511	0.533	0.533	0.511	0.511	0.533	0.644	0.800	0.933	0.933
11 lm1b-sec10k-withstopwords-dim75	0.66	2.489	2.178	2.044	1.844	1.622	1.511	1.400	1.267	1.111	1.000	0.911	0.778	0.644	0.622	0.511	0.489	0.467	0.489	0.533	0.511	0.489	0.578	0.733	0.933	0.933
12 lm1b-sec10k-nostopwords-hypertune2-dim35	0.8	3.556	3.400	3.044	2.778	2.511	2.222	1.978	1.822	1.822	1.689	1.578	1.511	1.289	1.156	1.022	0.956	0.889	0.756	0.689	0.667	0.644	0.556	0.622	0.911	0.911

Based a visual analysis of this heatmap, an optimized value of “similarity threshold” was picked for each model.

These values are captured in mymodels.txt file, to be used as the default similarity thresholds for that model.

# 4. Results / Evaluation

Analysis of the output : of the model and similarity threshold handpicked from the pool

## INPUT

- 1722 total phrase pairs generated from the 42 phrases
- 90 pairs (phrases from same group) are meant to be similar
- 1632 pairs (from different groups) are meant to be dissimilar

## OUTPUT

Model used: **lm1b-sec10k-withstopwords-dim50**  
Similarity threshold of **0.695**

- 78 pairs identified as similar and 1644 pairs as dissimilar.
- This includes 16 false positives and 28 false negatives.  
False positives included pairs like "fall asleep" and "after dark" - which might appear in similar contexts.

44 errors out of 1722 inputs translates to:

Error Rate: **2.6%**

Success Rate: **97.4%**



# 4. Results / Evaluation

## Side Notes

- As the model artifacts generated by BlazingText are compatible with **Gensim**, I was able to load the models into memory on the notebook instance itself using gensim module and run inference against them – without needing to spin up an inference endpoint via sagemaker. This is very handy for quick non-production workloads.
- Gensim provides some enhanced functionality and I was able to use that library for interesting applications like “analogies”

France : Paris :: India : ?  
 Nearest word vector: “delhi”

USA:Obama :: India : ?  
 Nearest word vector: “manmohan”

```
In [49]: # model1 = Lm1b-sec10k-nostopwords-hypertune2-dim35 (vector_dim = 35)
# model2 = Lm1b-nostopwords (vector_dim = 100)
```

```
from gensim.models import KeyedVectors
word_vectors1 = KeyedVectors.load_word2vec_format('/tmp/model1/vectors.txt', binary=False)
word_vectors2 = KeyedVectors.load_word2vec_format('/tmp/model2/vectors.txt', binary=False)
#word_vectors = KeyedVectors.load_word2vec_format('/tmp/model/vectors.bin', binary=True)

2018-10-29 02:07:05,985 : INFO : loading projection weights from /tmp/model1/vectors.txt
2018-10-29 02:07:23,021 : INFO : loaded (480774, 35) matrix from /tmp/model1/vectors.txt
2018-10-29 02:07:23,090 : INFO : loading projection weights from /tmp/model2/vectors.txt
2018-10-29 02:07:57,870 : INFO : loaded (390131, 100) matrix from /tmp/model2/vectors.txt
```

```
In [50]: positive = ['india', 'paris']
negative=['france']
print (word_vectors1.most_similar(positive=positive, negative=negative))
print()
print (word_vectors2.most_similar(positive=positive, negative=negative))
```

```
2018-10-29 02:09:08,195 : INFO : precomputing L2-norms of word weight vectors
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issubdtype from `int` to `np.signedinteger` is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.
if np.issubdtype(vec.dtype, np.int):
2018-10-29 02:09:08,356 : INFO : precomputing L2-norms of word weight vectors
```

```
[('mumbai', 0.9106876850128174), ('delhi', 0.8844015598297119), ('bombay', 0.8769457340240479), ('jaipur', 0.8636860251426697), ('talegaon', 0.8574641346931458), ('shinjuku', 0.8501175045967102), ('akshaya', 0.8470752239227295), ('agra', 0.8468923568725586), ('kuala', 0.8466311097145081), ('lumpur', 0.844670295715332)]

[('delhi', 0.8931387066841125), ('mumbai', 0.8217498064041138), ('delhimumbai', 0.8153454065322876), ('jaipur', 0.7740672826766968), ('bharatpur', 0.7710288763046265), ('gandhinagar', 0.7609442472457886), ('bombay', 0.7564903497695923), ('gandhara', 0.7466191053390503), ('chandigarh', 0.74406784727287292), ('hyderabad', 0.7436448335647583)]
```

```
In [51]: positive = ['india', 'obama']
negative=['usa']
print (word_vectors1.most_similar(positive=positive, negative=negative))
print()
print (word_vectors2.most_similar(positive=positive, negative=negative))
```

```
[('posttaliban', 0.8417814373970032), ('rakhmon', 0.8133049011230469), ('eveofelection', 0.8021421432495117), ('governmentinwaiting', 0.8019393682479858), ('gandhi's', 0.8013871908187866), ('yudhoyono', 0.7967199683189392), ('postsaddam', 0.7944049835205078), ('democraticallyelected', 0.7921048402786255), ('indopakistani', 0.7907671928405762), ('shiiteled', 0.7904908657073975)]

[('manmohan', 0.65682053565979), ('indiapakistan', 0.648622989654541), ('mukherjee', 0.6287984848022461), ('barack', 0.6167978048324585), ('bjp', 0.614942729473114), ('indopakistani', 0.6147809028625488), ('pakistaniindian', 0.6100256443023682), ('yudhoyono', 0.6063039302825928), ('sinoindian', 0.6050500273704529), ('pranab', 0.6023163199424744)]
```