Introduction

In this project, I classify Yelp round-10 review datasets. The reviews contain a lot of metadata that can be mined and used to infer meaning, business attributes, and sentiment. For simplicity, I classify the review comments into two class: either as positive or negative. Reviews that have star higher than three are regarded as positive while the reviews with star less than or equal to 3 are negative. Therefore, the problem is a supervised learning. To build and train the model, I first tokenize the text and convert them to sequences. Each review comment is limited to 50 words. As a result, short texts less than 50 words are padded with zeros, and long ones are truncated. After processing the review comments, I trained three model in three different ways:

- Model-1: In this model, a neural network with LSTM and a single embedding layer were used.
- Model-2: In Model-1, an extra 1D convolutional layer has been added on top of LSTM layer to reduce the training time.
- Model-3: In this model, I use the same network architecture as Model-2, but use the pre-trained glove 100 dimension word embeddings as initial input.

Since there are about 1.6 million input comments, it takes a while to train the models. To reduce the training time step, I limit the training epoch to three. After three epochs, it is evident that Model-2 is better regarding both training time and validation accuracy.

Project Outline

In this project I will cover the follwouings :

- **Download data from yelp and process them**
- **Build neural network with LSTM**
- **Build neural network with LSTM and CNN**
- **Use pre-trained GloVe word embeddings**
- **Word Embeddings from Word2Vec**

Import libraries

```
# Keras
from keras.preprocessing.text import Tokenizer
# from keras.preprocessing.sequence import pad_sequences
from keras.utils import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, LSTM, Conv1D, MaxPooling1D, Dropout, Activation
from keras.layers import Embedding

## Plot
import plotly.offline as py
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)
import matplotlib as plt

# NLTK
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer

# Other
import re
import string
import numpy as np
import pandas as pd
from sklearn.manifold import TSNE
```

Data Processing

```
# df = pd.read_csv('yelp.csv', sep = '|', names = ['stars', 'text'], error_bad_lines=False)
df = pd.read_csv('yelp.csv')
```

```
df.tail()
```

	business_id	date	review_id	stars	
9995	VY_tvNUCCXGXQeSvJl757Q	2012-07-28	Ubyfp2RSDYW0g7Mbr8N3iA	3	First lunch he use
9996	EKzMHI1tip8rC1-ZAy64yg	2012-01-18	2XyIOQKbVFb6uXQdJ0RzlQ	4	Should delicious
9997	53YGfwmbW73JhFiemNeyzQ	2010-11-16	jyznYkIbpqVmlsZxSDSypA	4	I recer Olive a bı
9998	9SKdOoDHcFoxK5ZtsgHJoA	2012-12-02	5UKq9WQE1qQbJ0DJbc-B6Q	2	My ne

```
df= df.dropna()
# df = df[df.stars.apply(lambda x: x.isnumeric())]
df = df[df.stars.apply(lambda x: x !="")]
df = df[df.text.apply(lambda x: x !="")]
```

```
df.describe()
```

	stars	text
count	1673870	1673870
unique	5	1673452
top	5	Good stuff
freq	709732	6

```
df.head()
```

	business_id	date	review_id	stars	text	ty
0	9yKzy9PApeiPPOUJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	My wife took me here on my birthday for breakf...	rev
1	ZRJwVLyzEJq1VAihDhYiow	2011-	IjZ33sJrzXqU-0X6U8NwyA	5	I have no idea why some	rev

Convert five classes into two classes (positive = 1 and negative = 0)

Since the main purpose is to identify positive or negative comments, I convert five class star category into two classes:

- (1) Positive: comments with stars > 3 and
- (2) Negative: comments with stars <= 3

```
labels = df['stars'].map(lambda x : 1 if int(x) > 3 else 0)
```

Tokenize text data

Because of the computational expenses, I use the top 20000 unique words. First, tokenize the comments then convert those into sequences. I keep 50 words to limit the number of words in each comment.

```
stopwords
```

```
<WordListCorpusReader in '.../corpora/stopwords' (not loaded yet)>
```

```
def clean_text(text):

    ## Remove puncuation
    text = text.translate(string.punctuation)

    ## Convert words to lower case and split them
```

```
    ## Convert words to lower case and split them
    text = text.lower().split()

    ## Remove stop words
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops and len(w) >= 3]

    text = " ".join(text)

    # Clean the text
    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r",", " ", text)
    text = re.sub(r"\.", " ", text)
    text = re.sub(r"!", " ! ", text)
    text = re.sub(r"\/", " ", text)
    text = re.sub(r"\^", " ^ ", text)
    text = re.sub(r"\+", " + ", text)
    text = re.sub(r"\-", " - ", text)
    text = re.sub(r"\=", " = ", text)
    text = re.sub(r"'", " ", text)
    text = re.sub(r"(\d+)(k)", r"\g<1>000", text)
    text = re.sub(r":", " : ", text)
    text = re.sub(r" e g ", " eg ", text)
    text = re.sub(r" b g ", " bg ", text)
    text = re.sub(r" u s ", " american ", text)
    text = re.sub(r"\0s", "0", text)
    text = re.sub(r" 9 11 ", "911", text)
    text = re.sub(r"e - mail", "email", text)
    text = re.sub(r"j k", "jk", text)
    text = re.sub(r"\s{2,}", " ", text)

    text = text.split()
    stemmer = SnowballStemmer('english')
    stemmed_words = [stemmer.stem(word) for word in text]
    text = " ".join(stemmed_words)

    return text
```

```
nltk.download('stopwords')
df['text'] = df['text'].map(lambda x: clean_text(x))
```

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

```
df.head(10)
```

	stars	text
0	5	minut realiz conflict block away visit way lea...
2	5	love conflict kitchen food fantast downsid rot...
3	4	holi moli addict first heard conflict kitchen ...
4	4	great persian food though cheap fill street fo...
5	4	yummi food good price encourag tri new thing w...
6	5	passion peopl passion food ask question relat ...
7	1	first food never mix polit second hummus serv ...
8	5	past review discuss concept need countri exper...
9	5	premi food countri conflict delici creativ qui...
10	5	sixth star joke place everyth right give flip ...

```
vocabulary_size = 20000
tokenizer = Tokenizer(num_words= vocabulary_size)
```

```
tokenizer.fit_on_texts(df['text'])

sequences = tokenizer.texts_to_sequences(df['text'])
data = pad_sequences(sequences, maxlen=50)
```

```
print(data.shape)
```

```
(10000, 50)
```

Build neural network with LSTM

Network Architechture

The network starts with an embedding layer. The layer lets the system expand each token to a more massive vector, allowing the network to represent a word in a meaningful way. The layer takes 20000 as the first argument, which is the size of our vocabulary, and 100 as the second input parameter, which is the dimension of the embeddings. The third parameter is the input_length of 50, which is the length of each comment sequence.

```
model_lstm = Sequential()
model_lstm.add(Embedding(20000, 100, input_length=50))
model_lstm.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model_lstm.add(Dense(1, activation='sigmoid'))
model_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Train the network

There are about 1.6 million comments, and it takes a while to train the model in a MacBook Pro. To save time I have used only three epochs. GPU machines can be used to accelerate the training with more time steps. I split the whole datasets as 60% for training and 40% for validation.

```
model_lstm.fit(data, np.array(labels), validation_split=0.4, epochs=3)
```

```
Epoch 1/3
188/188 [==============================] - 25s 118ms/step - loss: 0.5358 - accuracy: 0.7358 - val_loss: 0.4567 - val_accurac
Epoch 2/3
188/188 [==============================] - 19s 103ms/step - loss: 0.3123 - accuracy: 0.8710 - val_loss: 0.4915 - val_accurac
Epoch 3/3
188/188 [==============================] - 19s 101ms/step - loss: 0.1851 - accuracy: 0.9333 - val_loss: 0.6261 - val_accurac
<keras.callbacks.History at 0x7fe905996b50>
```

Build neural network with LSTM and CNN

The LSTM model worked well. However, it takes forever to train three epochs. One way to speed up the training time is to improve the network adding “Convolutional” layer. Convolutional Neural Networks (CNN) come from image processing. They pass a “filter” over the data and calculate a higher-level representation. They have been shown to work surprisingly well for text, even though they have none of the sequence processing ability of LSTMs.

```
def create_conv_model():
    model_conv = Sequential()
    model_conv.add(Embedding(vocabulary_size, 100, input_length=50))
    model_conv.add(Dropout(0.2))
    model_conv.add(Conv1D(64, 5, activation='relu'))
    model_conv.add(MaxPooling1D(pool_size=4))
    model_conv.add(LSTM(100))
    model_conv.add(Dense(1, activation='sigmoid'))
    model_conv.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model_conv
```

```
model_conv = create_conv_model()
model_conv.fit(data, np.array(labels), validation_split=0.4, epochs = 3)
```

```
Epoch 1/3
188/188 [==============================] - 9s 38ms/step - loss: 0.5504 - accuracy: 0.7243 - val_loss: 0.4565 - val_accuracy:
Epoch 2/3
188/188 [==============================] - 6s 31ms/step - loss: 0.3205 - accuracy: 0.8638 - val_loss: 0.4903 - val_accuracy:
Epoch 3/3
```

```
188/188 [==============================] - 7s 38ms/step - loss: 0.1686 - accuracy: 0.9407 - val_loss: 0.6308 - val_accuracy:
<keras.callbacks.History at 0x7fe902989760>
```

Save processed Data

```
df_save = pd.DataFrame(data)
df_label = pd.DataFrame(np.array(labels))
```

```
result = pd.concat([df_save, df_label], axis = 1)
```

```
result.to_csv('train_dense_word_vectors.csv', index=False)
```

Use pre-trained Glove word embeddings

In this subsection, I want to use word embeddings from pre-trained Glove. It was trained on a dataset of one billion tokens (words) with a vocabulary of 400 thousand words. The glove has embedding vector sizes, including 50, 100, 200 and 300 dimensions. I chose the 100-dimensional version. I also want to see the model behavior in case the learned word weights do not get updated. I, therefore, set the trainable attribute for the model to be False.

Get embeddings from Glove

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
```

```
--2023-04-25 02:42:47--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2023-04-25 02:42:48--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2023-04-25 02:42:48--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip        100%[===================>] 822.24M  5.06MB/s    in 2m 39s

2023-04-25 02:45:27 (5.17 MB/s) - 'glove.6B.zip' saved [862182613/862182613]
```

```
!unzip glove*.zip
```

```
Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

```
embeddings_index = dict()
f = open('glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

```
Loaded 400000 word vectors.
```

```
# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocabulary_size, 100))
for word, index in tokenizer.word_index.items():
    if index > vocabulary_size - 1:
```

```
        break
    else:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
```

Develop model

I use the same model architecture with a convolutional layer on top of the LSTM layer.

```
model_glove = Sequential()
model_glove.add(Embedding(vocabulary_size, 100, input_length=50, weights=[embedding_matrix], trainable=False))
model_glove.add(Dropout(0.2))
model_glove.add(Conv1D(64, 5, activation='relu'))
model_glove.add(MaxPooling1D(pool_size=4))
model_glove.add(LSTM(100))
model_glove.add(Dense(1, activation='sigmoid'))
model_glove.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
model_glove.fit(data, np.array(labels), validation_split=0.4, epochs = 3)
```

```
Epoch 1/3
188/188 [==============================] - 7s 25ms/step - loss: 0.6074 - accuracy: 0.6862 - val_loss: 0.5725 - val_accuracy:
Epoch 2/3
188/188 [==============================] - 4s 22ms/step - loss: 0.5487 - accuracy: 0.7167 - val_loss: 0.5568 - val_accuracy:
Epoch 3/3
188/188 [==============================] - 4s 19ms/step - loss: 0.5118 - accuracy: 0.7450 - val_loss: 0.5381 - val_accuracy:
<keras.callbacks.History at 0x7fe8ee936910>
```

Word embedding visialization

In this subsection, I want to visualize word embedding weights obtained from trained models. Word embeddings with 100 dimensions are first reduced to 2 dimensions using t-SNE. Tensorflow has an excellent tool to visualize the embeddings in a great way, but here I just want to visualize the word relationship.

Get embedding weights from glove

```
lstm_embds = model_lstm.layers[0].get_weights()[0]
```

```
conv_embds = model_conv.layers[0].get_weights()[0]
```

```
glove_emds = model_glove.layers[0].get_weights()[0]
```

Get word list

```
word_list = []
for word, i in tokenizer.word_index.items():
    word_list.append(word)
```

Scatter plot of first two components of TSNE

```
import plotly.io as pio
pio.renderers.default = "colab"
```

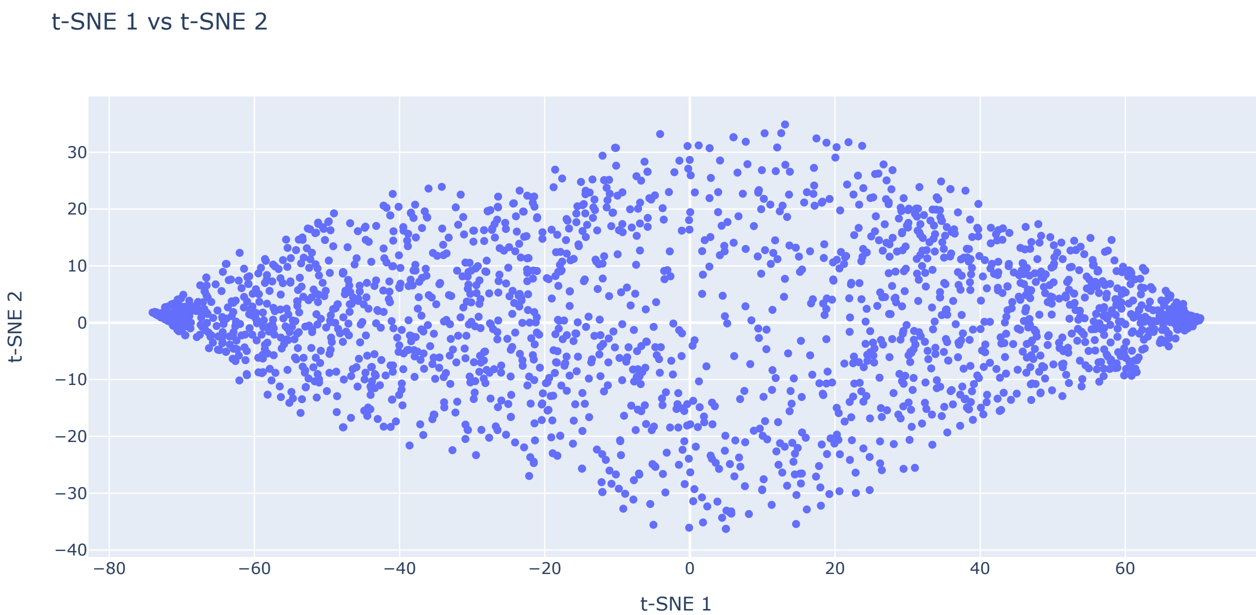
```
def plot_words(data, start, stop, step):
    trace = go.Scatter(
        x = data[start:stop:step,0],
        y = data[start:stop:step, 1],
        mode = 'markers',
        text= word_list[start:stop:step]
    )
    layout = dict(title= 't-SNE 1 vs t-SNE 2',
                  yaxis = dict(title='t-SNE 2'),
```

```
                xaxis = dict(title='t-SNE 1'),
                hovermode= 'closest')
    fig = dict(data = [trace], layout= layout)
    py.iplot(fig)
```

1. LSTM

```
number_of_words = 2000
lstm_tsne_embds = TSNE(n_components=2).fit_transform(lstm_embds)
```

```
plot_words(lstm_tsne_embds, 0, number_of_words, 1)
```



2. CNN + LSTM

```
conv_tsne_embds = TSNE(n_components=2).fit_transform(conv_embds)
```

```
plot_words(conv_tsne_embds, 0, number_of_words, 1)
```

t-SNE 1 vs t-SNE 2

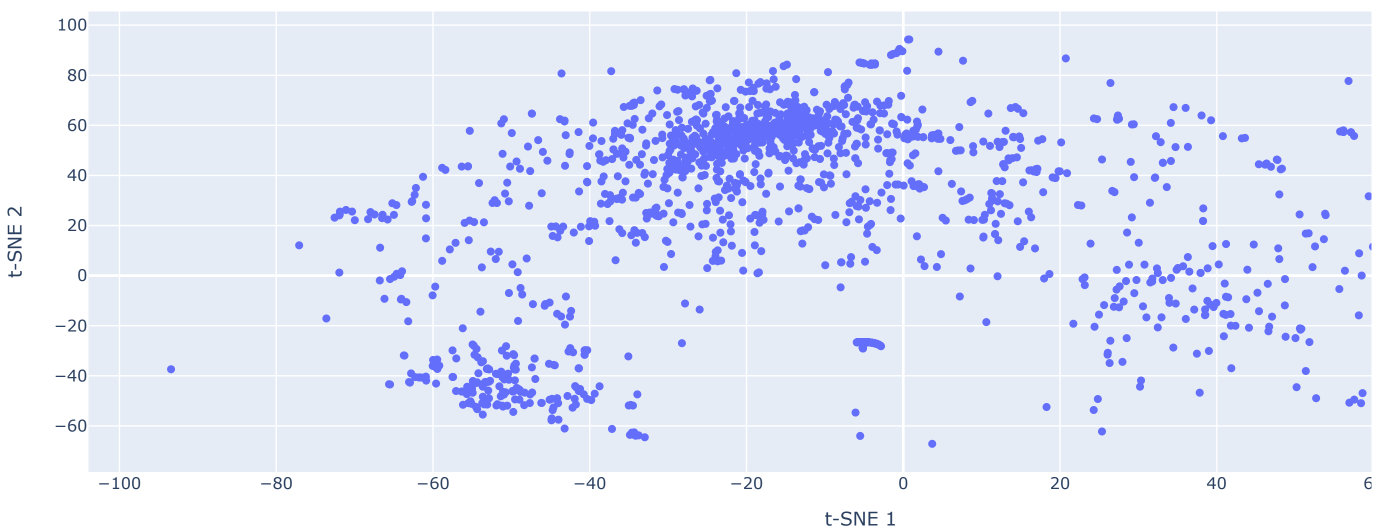
3. Glove

```
glove_tsne_embds = TSNE(n_components=2).fit_transform(glove_emds)
```

```
# Import the necessaries libraries
import plotly.offline as pyo
import plotly.graph_objs as go
# Set notebook mode to work in offline
pyo.init_notebook_mode()
```

```
plot_words(glove_tsne_embds, 0, number_of_words, 1)
```

t-SNE 1 vs t-SNE 2



Word Embeddings from Word2Vec

In this subsection, I use word2vec to create word embeddings from the review comments. Word2vec is one algorithm for learning a word embedding from a text corpus.

```
from gensim.models import Word2Vec
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

Tokenize the reviews coments.

```
df['tokenized'] = df.apply(lambda row : nltk.word_tokenize(row['text']), axis=1)
```

```
df.head()
```

	business_id	date	review_id	stars	text	type	user_id	cool	useful	funny	tok
0	9yKzy9PApeiPPOUJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	wife took birthday breakfast excel weather per...	review	rLtl8ZkDX5vH5nAx9C3q5Q	2	5	0	[wi l bı
1	ZRJwVLyzEJq1VAihDhYiow	2011-07-27	IjZ33sJrzXqU-0X6U8NwyA	5	idea peopl give bad review place goe show you ...	review	0a2KyEL0d3Yb1V6aivbIuQ	0	0	0	peo bad pla

Train word2vec model

```
model_w2v = Word2Vec(df['tokenized'])
```

```
model_w2v.wv.index_to_key
```

```
from gensim.models.keyedvectors import KeyedVectors
# Convert to KeyedVectors format
kv = KeyedVectors(vector_size=model_w2v.vector_size)
kv.add_vectors(model_w2v.wv.index_to_key, model_w2v.wv.vectors)

# Get the vectors for all words in the vocabulary
X = kv[kv.index_to_key]
```

```
X = [model_w2v.wv.key_to_index]
```

Plot Word Vectors Using PCA

```
from sklearn.decomposition import TruncatedSVD
```

```
tsvd = TruncatedSVD(n_components=5, n_iter=10)
result = tsvd.fit_transform(X)
```

```
result.shape
```

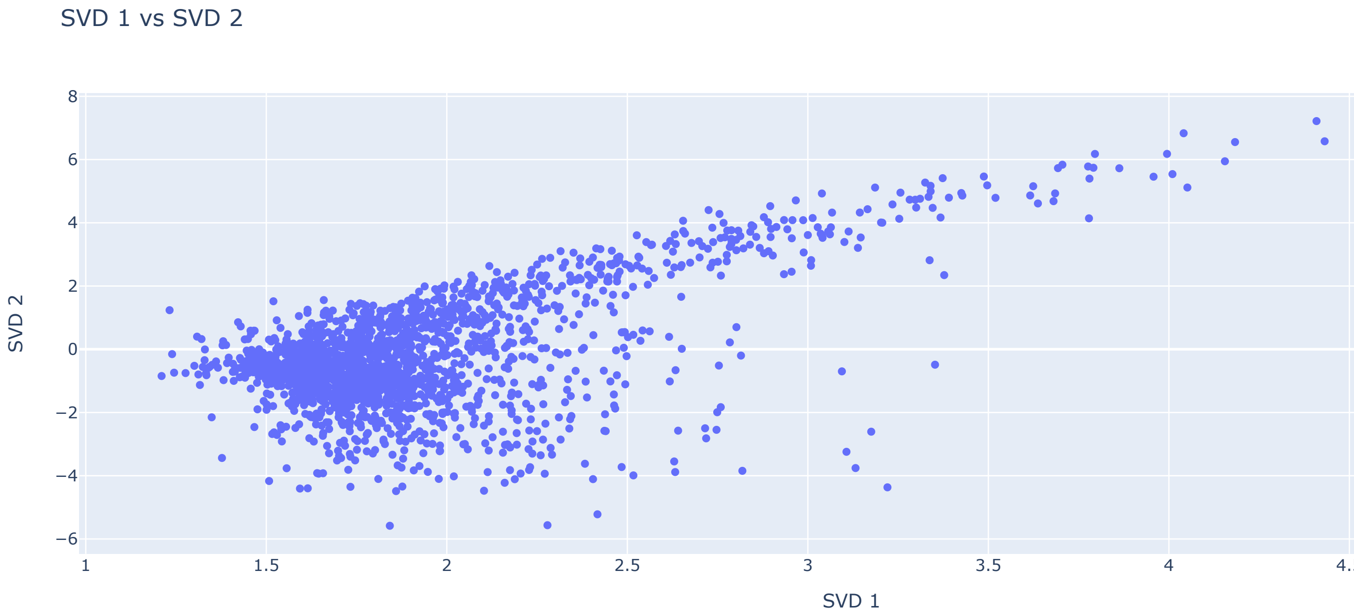
```
(7007, 5)
```

```
tsvd_word_list = []
words = list(model_w2v.wv.index_to_key)
for i, word in enumerate(words):
    tsvd_word_list.append(word)

trace = go.Scatter(
    x = result[0:number_of_words, 0],
    y = result[0:number_of_words, 1],
    mode = 'markers',
    text= tsvd_word_list[0:number_of_words]
)

layout = dict(title= 'SVD 1 vs SVD 2',
              yaxis = dict(title='SVD 2'),
              xaxis = dict(title='SVD 1'),
              hovermode= 'closest')

fig = dict(data = [trace], layout= layout)
py.iplot(fig)
```



✓ 0s completed at 11:19 PM