1. For any information retrieval system, it is important benchmark it and one of the benchmarking measures is relevance of the results in accordance to the query. For this benchmarking of relevance, we need document collection, suite of queries and classifier that can classify the documents as relevant and irrelevant. Accuracy is one such way of benchmarking relevance, but it is sensitive to skewed data. Thus we use more data agnostic measures as recall and precision that clearly highlight fraction of relevant documents that were retrieved and fraction of the retrieved documents that were relevant respectively. Accuracy tells overall fraction of correct classifications. Consider the following confusion matrix-

	Correct	Incorrect
Selected	10	60
Not selected	100	600

   In this case accuracy of the classifier comes to (10 + 600) / 770 = ~80%.
   Whereas, Recall for this classifier will be 10 / (10 + 100) = ~9%
   Precision for this classifier will be 10 / (10 + 60) = ~14%
   Which are not very impressive.
   Thus, it is evident that precision and recall are better measures of the performance of the classifier and hence their use is justified over the use of accuracy.

2. P(period after a 3 letter word and this period indicates an abbreviation) =
   P(is-abbreviation | three-letter-word) * P(three-letter-word)
   - ⇨ 0.8*(0.0003)
   - ⇨ 0.00024

3. After removing the punctuation marks and other special characters but keeping only the space and English alphabets from the text, the entropy is 3.895 per character. As part of preprocessing, I also converted the text to lower case. I re-used the code from the previous homework and used a dictionary to first count the frequency of each character and then calculate the probability of each character. And finally using the calculated probability of each character to compute the entropy.

4. The vector space model is defined as the co-occurrence of counts of context items in a corpus for a target. The context items are lemmatized and used along with their part of speech tag. A context item is within a relevant context only if it appears in a 3-character wide window on each side of the target, without crossing the sentence boundary. The semantic similarity of two words based on their distributional similarity is measured as the cosine angle between the two vectors. The context words counts are transformed into association weights and PPMI is used to do so. Also, to reduce the dimensionality of the vector, SVD is employed. This model is used to predict the semantic similarity of two words based on their distributional similarity.

5.

a) Reweighting has proved advantageous to improve VSM. On the raw counts of word -document matrix, we are getting a Spearman r for wordsim353 of 0.095. However, when we reweight with PMI, we get 0.310, with PPMI, we get 0.326, and with TF-IDF, we get 0.198. since the aim is to

achieve the Spearman r as close to 1 as possible, the reweighting has improved the Spearman r and hence a better VSM. The best reweighting scheme for word-document matrix was PPMI. PMI reweighting was not that good because PMI is too sensitive to rare events. The code was taken from https://github.com/cgpotts/cs224u/blob/master/vsm.ipynb. To analyze the effect of different reweighting schemes, we used base code from the above-mentioned link. PMI has the problem of being biased toward infrequent events; very rare words tend to have very high PMI values.

possible solution is Laplace smoothing: Before computing PMI, a small constant k (values of 0.1-3 are common) is added to each of the counts, shrinking (discounting)all the non-zero values. The larger the k, the more the non-zero counts are discounted. For these reasons, it is more common to use Positive PMI (called PPMI) which replaces all negative PMI values with zero.

CODE SNIPPET –

```
ww = build(os.path.join(vsmdata_home, 'imdb-wordword.csv'))
wd = build(os.path.join(vsmdata_home, 'imdb-worddoc.csv'))

ww_ppmi = pmi(mat=ww[0], rownames=ww[1], positive=True)
wd_ppmi = pmi(mat=wd[0], rownames=wd[1], positive=True)

ww_pmi= pmi(mat=ww[0],rownames=ww[1],positive=False)
wd_pmi= pmi(mat=wd[0],rownames=wd[1],positive=False)

ww_tfidf = tfidf(mat=ww[0],rownames=ww[1])
wd_tfidf = tfidf(mat=wd[0],rownames=wd[1])
```

```
print('WORD WORD SIMILARITY')
print("Bare Matrix")
full_word_similarity_evaluation(ww[0],ww[1])
print("PMI Matrix")
full_word_similarity_evaluation(ww_pmi[0],ww_pmi[1])
print("Positive PMI Matrix")
full_word_similarity_evaluation(ww_ppmi[0],ww_ppmi[1])
print("TF-IDF Matrix")
full_word_similarity_evaluation(ww_tfidf[0],ww_tfidf[1])
print()
```

```
print('WORD DOCUMENT SIMILARITY')
print("Bare Matrix")
full_word_similarity_evaluation(wd[0],wd[1])
print("PMI Matrix")
full_word_similarity_evaluation(wd_pmi[0],wd_pmi[1])
print("Positive PMI Matrix")
full_word_similarity_evaluation(wd_ppmi[0],wd_ppmi[1])
print("TF-IDF Matrix")
full_word_similarity_evaluation(wd_tfidf[0],wd_tfidf[1])
```

b) Performing LSA on any of the matrices (with or without reweighting) degrades the performance. The Spearman r on bare matrix is dropped from 0.095 to 0.016, and the reweighting schemes PMI and TF-IDF reduced the Spearman r from .31 to .23, and .198 to .0028 respectively. This is likely because wordsim353 does not create a sparse word x doc matrix. The sparser a matrix is, the more likely a dimensionality reduction scheme like LSA is to help. When a matrix is sparse, LSA can take out the unnecessary dimensions in the matrix, yet preserving the underlying structure of the data and yield helpful estimations. But when the matrix is denser, LSA takes out

useful information. Also Using latent semantic analysis (LSA) when we construct a low rank approximation. SVD assumes normal distribution of data but in reality, term distribution is not normally distributed. Lastly matrix entries are weights, not counts, may be normally distributed even when counts are not and so LSA might sometimes prove beneficial too.

CODE SNIPPET –

```
ww_lsa = lsa(mat=ww[0],rownames=ww[1])
wd_lsa = lsa(mat=wd[0],rownames=wd[1])

ww_pmi_lsa = lsa(mat=ww_pmi[0],rownames=ww_pmi[1])
ww_ppmi_lsa = lsa(mat=ww_ppmi[0],rownames=ww_ppmi[1])
ww_tfidf_lsa = lsa(mat=ww_tfidf[0],rownames=ww_tfidf[1])

wd_pmi_lsa = lsa(mat=wd_pmi[0],rownames=wd_pmi[1])
wd_ppmi_lsa = lsa(mat=wd_ppmi[0],rownames=wd_ppmi[1])
wd_tfidf_lsa = lsa(mat=wd_tfidf[0],rownames=wd_tfidf[1])
print('WORD WORD SIMILARITY')
print("Bare Matrix LSA")
full_word_similarity_evaluation(ww_lsa[0],ww_lsa[1])
print("PMI LSA Matrix")
full_word_similarity_evaluation(ww_pmi_lsa[0],ww_pmi_lsa[1])
print("PPMI LSA Matrix")
full_word_similarity_evaluation(ww_ppmi_lsa[0],ww_ppmi_lsa[1])
print("TF-IDF LSA Matrix")
full_word_similarity_evaluation(ww_tfidf_lsa[0],ww_tfidf_lsa[1])
print('WORD DOCUMENT SIMILARITY')
print("Bare Matrix LSA")
full_word_similarity_evaluation(wd_lsa[0],wd_lsa[1])
print("PMI LSA Matrix")
full_word_similarity_evaluation(wd_pmi_lsa[0],wd_pmi_lsa[1])
print("PPMI LSA Matrix")
full_word_similarity_evaluation(wd_ppmi_lsa[0],wd_ppmi_lsa[1])
print("TF-IDF LSA Matrix")
full_word_similarity_evaluation(wd_tfidf_lsa[0],wd_tfidf_lsa[1])
```

c) Looking solely at wordsim353, we found the best results when constructing the VSM in the following way: using a word x word matrix (instead of a word x document matrix where we got a Spearman r value of 0.326), using PPMI as a reweighting scheme (instead of PMI or TF-IDF), using cosine for vector comparison (instead of Euclidian distance or Jaccard), and not using any dimensionality reduction (LSA). This gave us a Spearman r value of 0.566. As discussed above, dimensionality reduction did not prove beneficial in this case. And PPMI helped because we did not have an enormous corpus to understand the granularity of the probability of co-occurrence of words in case of rare events, required for PMI to be effective.

CODE SNIPPET –

```
def word_similarity_evaluation(reader, mat, rownames, distfunc=cosine):

ww = build(os.path.join(vsmdata_home, 'imdb-wordword.csv'))
ww_ppmi = pmi(mat=ww[0], rownames=ww[1], positive=True)
wd = build(os.path.join(vsmdata_home, 'imdb-worddoc.csv'))
wd_ppmi = pmi(mat=wd[0], rownames=wd[1], positive=True)
```

```
print('WORD WORD SIMILARITY')
print("Positive PMI Matrix")
full_word_similarity_evaluation(ww_ppmi[0],ww_ppmi[1])
```

SUMMARY -

Matrix	**WORD WORD**	**WORD DOCUMENT**
	Cosine Distance	
BARE	Spearman r: -0.036	Spearman r: 0.095
PMI	Spearman r: 0.519	Spearman r: 0.310
PPMI	Spearman r: 0.566	Spearman r: 0.326
TF-IDF	Spearman r: 0.139	Spearman r: 0.198
	Euclidean distance	
BARE	Spearman r: -0.053	Spearman r: -0.114
PMI	Spearman r: 0.034	Spearman r: 0.169
PPMI	Spearman r: 0.035	Spearman r: 0.239
TF-IDF	Spearman r: -0.136	Spearman r: -0.204
	LSA – Euclidean distance	
BARE	Spearman r: -0.056	Spearman r: -0.131
PMI	Spearman r: 0.209	Spearman r: 0.164
PPMI	Spearman r: 0.139	Spearman r: 0.213
TF-IDF	Spearman r: -0.068	Spearman r: -0.019
	LSA – cosine Distance	
BARE	Spearman r: 0.004	Spearman r: 0.016
PMI	Spearman r: 0.502	Spearman r: 0.231
PPMI	Spearman r: 0.467	Spearman r: 0.324
TF-IDF	Spearman r: 0.091	Spearman r: 0.028