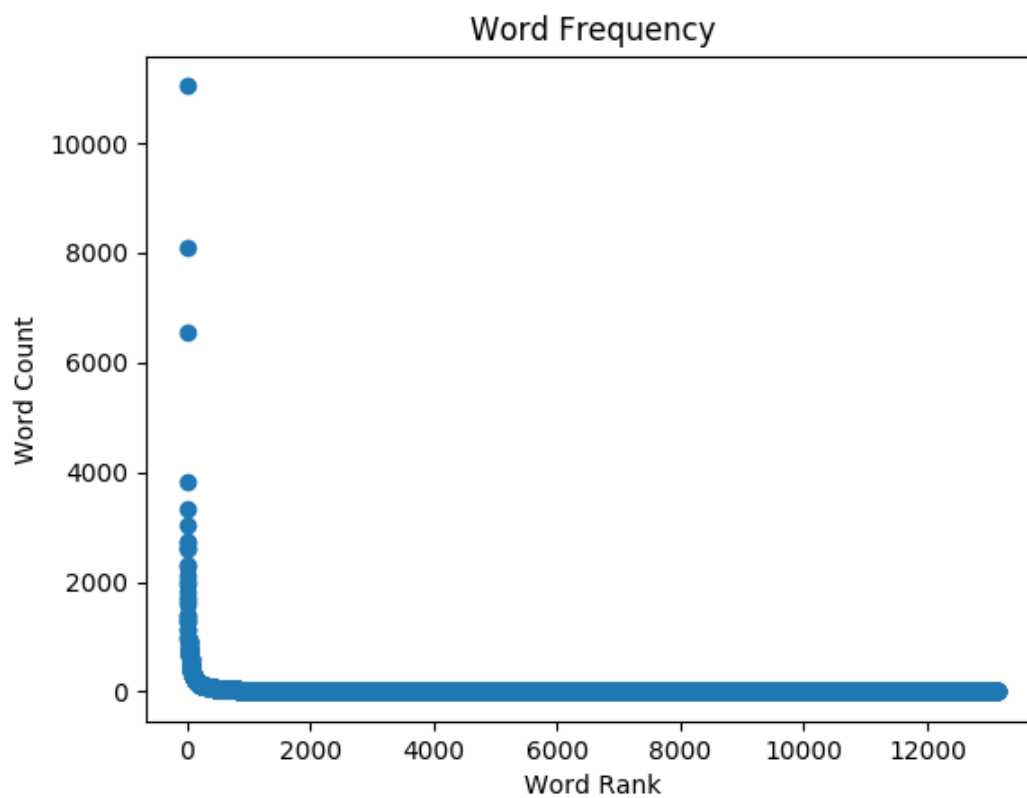


## Homework 7: Text Bag-of-Words Search and Classification

### Page 1 Distribution graph (5 points)

Show the distribution graph of words counts vs word rank.



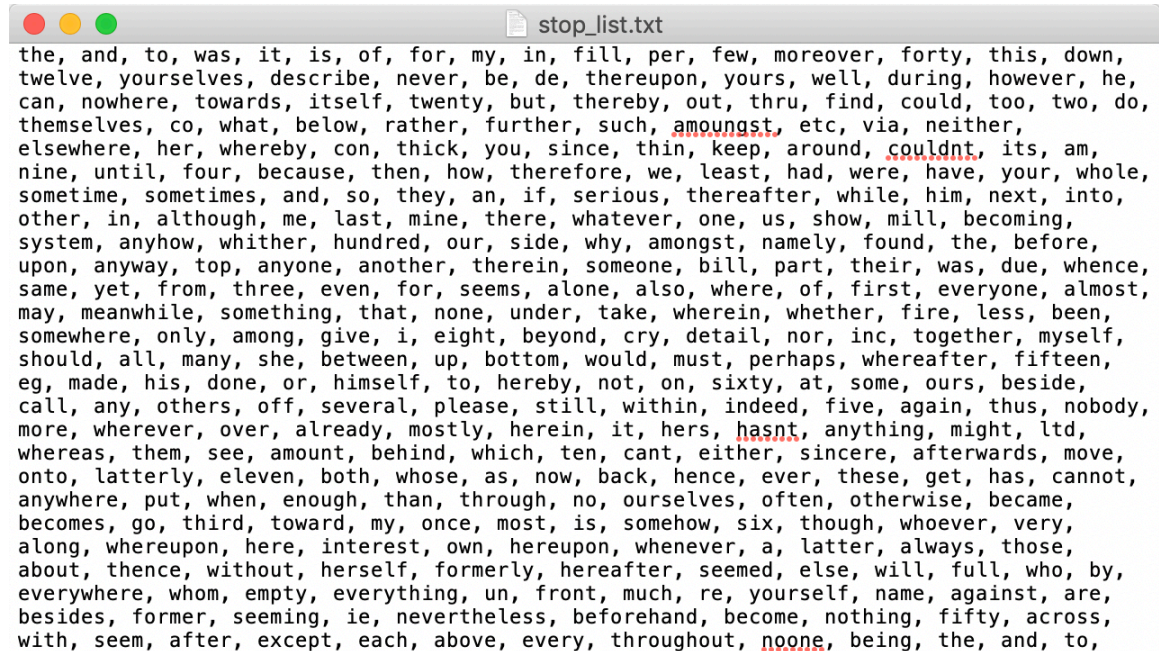
## Page 2 Identify the stop words (5 points)

List the stop words you choose as well as the frequency threshold.

The max document frequency threshold is 5000.

The minimum word occurrence threshold is 10.

Stop words (331 words):



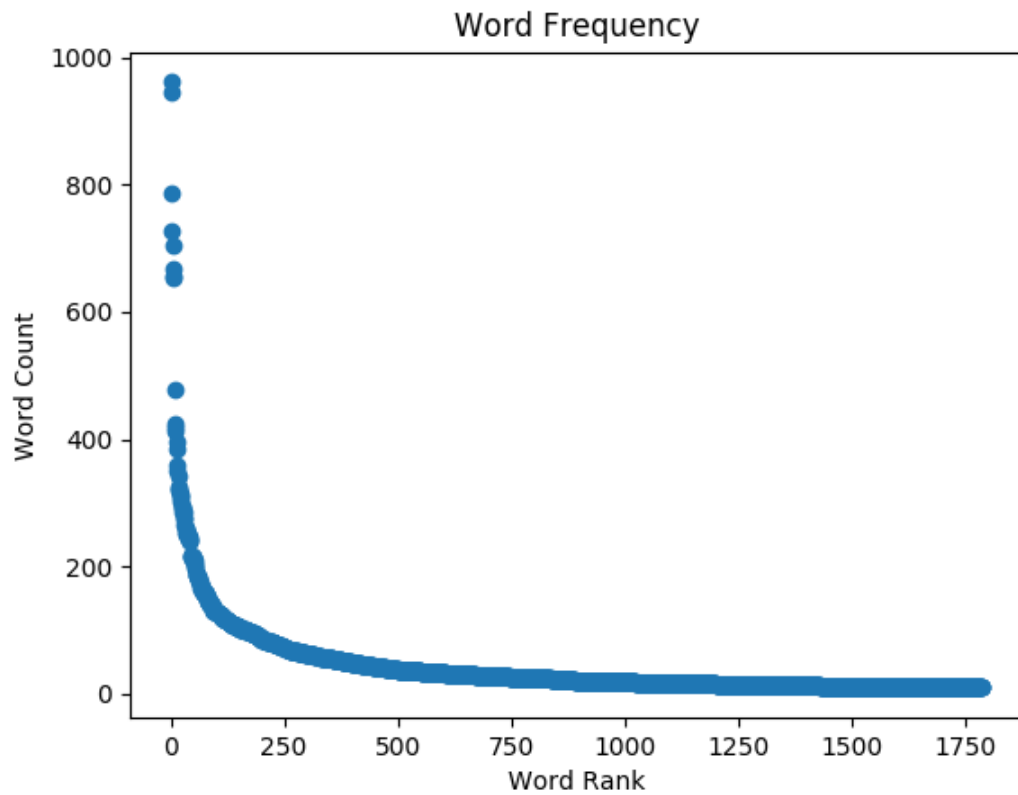
the, and, to, was, it, is, of, for, my, in, fill, per, few, moreover, forty, this, down, twelve, yourselves, describe, never, be, de, thereupon, yours, well, during, however, he, can, nowhere, towards, itself, twenty, but, thereby, out, thru, find, could, too, two, do, themselves, co, what, below, rather, further, such, amongst, etc, via, neither, elsewhere, her, whereby, con, thick, you, since, thin, keep, around, couldnt, its, am, nine, until, four, because, then, how, therefore, we, least, had, were, have, your, whole, sometime, sometimes, and, so, they, an, if, serious, thereafter, while, him, next, into, other, in, although, me, last, mine, there, whatever, one, us, show, mill, becoming, system, anyhow, whither, hundred, our, side, why, amongst, namely, found, the, before, upon, anyway, top, anyone, another, therein, someone, bill, part, their, was, due, whence, same, yet, from, three, even, for, seems, alone, also, where, of, first, everyone, almost, may, meanwhile, something, that, none, under, take, wherein, whether, fire, less, been, somewhere, only, among, give, i, eight, beyond, cry, detail, nor, inc, together, myself, should, all, many, she, between, up, bottom, would, must, perhaps, whereafter, fifteen, eg, made, his, done, or, himself, to, hereby, not, on, sixty, at, some, ours, beside, call, any, others, off, several, please, still, within, indeed, five, again, thus, nobody, more, wherever, over, already, mostly, herein, it, hers, hasnt, anything, might, ltd, whereas, them, see, amount, behind, which, ten, cant, either, sincere, afterwards, move, onto, latterly, eleven, both, whose, as, now, back, hence, ever, these, get, has, cannot, anywhere, put, when, enough, than, through, no, ourselves, often, otherwise, became, becomes, go, third, toward, my, once, most, is, somehow, six, though, whoever, very, along, whereupon, here, interest, own, hereupon, whenever, a, latter, always, those, about, thence, without, herself, formerly, hereafter, seemed, else, will, full, who, by, everywhere, whom, empty, everything, un, front, much, re, yourself, name, against, are, besides, former, seeming, ie, nevertheless, beforehand, become, nothing, fifty, across, with, seem, after, except, each, above, every, throughout, noone, being, the, and, to,

I choose Term Frequency-Inverse Document Frequency to decide the stop words and set threshold as 74.9.

And then add common English stop words and those words with word occurrence > 4000.

### Page 3 Distribution graph again (5 points)

After choosing the stop words, show the distribution graph of words counts vs word rank.



## Page 4 Code snippets (15 points)

Show the snippet of your code that you convert all the reviews into bag-of-words formulation using your chosen stop words and your code for nearest-neighbours with cos-distance.

```
cv_modify = CountVectorizer(stop_words=stop_list, lowercase=True, max_df=5000)
cv_fit_modify = cv_modify.fit_transform(text)
count_modify = cv_fit_modify.toarray()
count_sum_modify = count_modify.sum(axis=0)
count_ordered_modify = sorted(count_sum_modify, reverse=True)
vocabulary = cv_modify.get_feature_names()
feature = count_modify
```

```
sentence = ['Horrible customer service', ]
cv_sentence = CountVectorizer(lowercase=True, vocabulary=vocabulary)
cv_fit_sentence = cv_sentence.fit_transform(sentence)
vector_sentence = cv_fit_sentence.toarray()
# print(vector_sentence)

nbrs = NearestNeighbors(n_neighbors=1, algorithm='brute', metric='cosine')
nbrs.fit(vector_sentence)
distances, indices = nbrs.kneighbors(count_modify)
distances = distances.reshape(1, -1)
similarity = (1 - distances).reshape(1, -1)
```

## Page 5 Reviews with score (10 points)

Show the original reviews with the distance scores

The lowest 5 distance scores are:

0.22848325, 0.43408354, 0.49604737, 0.49604737, 0.51887478

The original reviews accordingly:

```

Rogers ...

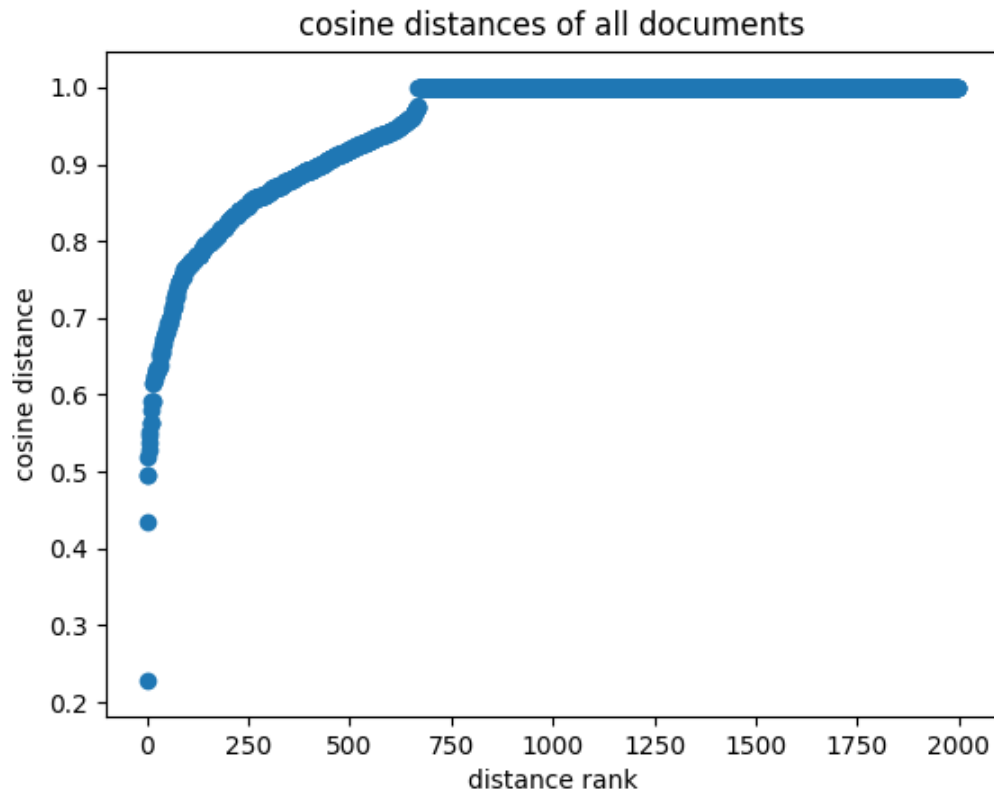
1) is over priced
2) have horrible customer service
3) faulty and incorrect billing
4) poor customer service
5) not enough options
6) never arrive for an appointment
#####
Horrible service, horrible customer service, and horrible quality of service! Do not
waste your time or money using this company for your pool needs. Dan (602)363-8267 broke
my pool filtration system and left it in a nonworking condition. He will not repair the
issue he caused, and told me to go somewhere else.

Save yourself the hassle, there are plenty of other quality pool companies out there.

Take care!
#####
Service was horrible came with a major attitude. Payed 30 for lasagna and was no where
worth it. Won't ever be going back and will NEVER recommend this place. was treated
absolutely horrible. Horrible.
#####
Horrible customer service! Been with them over 2 years, and after staying with them
during my last move they raised my bill almost double for the same services! Sent two
emails since I don't have time to call, not a single response. Will finally waste an
entire night to call to cancel my service.
#####
Went to Marca today to get a haircut and was given a great service both by front desk -
customer service and by Georgia, girl who did my hair. I guess I got lucky with her as she
has years of experience doing this job. She has excellent customer service skills and
takes excellent care of her customers.
#####
```

## Page 6 Query results (10 points)

Show your document results and explain the reasons that you choose them.



I choose 0.6 as the cosine distance threshold (i.e. 0.4 as the similarity threshold) and there are totally 16 documents matching the result.

## Page 7 Accuracy with threshold 0.5 (10 points)

Show your code for creating classifier. Report the accuracy on train and test dataset with threshold 0.5.

Accuracy of training set: 0.9988888888888889

Accuracy of test set: 0.945

```
random.seed(0)
index_test = random.sample(list(range(len(feature))), 200)
index_train = list(set(list(range(len(feature)))) - set(index_test))
train = feature[index_train, :]
test = feature[index_test, :]
train_label = np.array(label.iloc[index_train])
test_label = np.array(label.iloc[index_test])

lr = LogisticRegression(solver='lbfgs')
lr.fit(train, train_label)
predict1 = lr.predict(test)
print(lr.score(test, test_label))
print(lr.score(train, train_label))
```

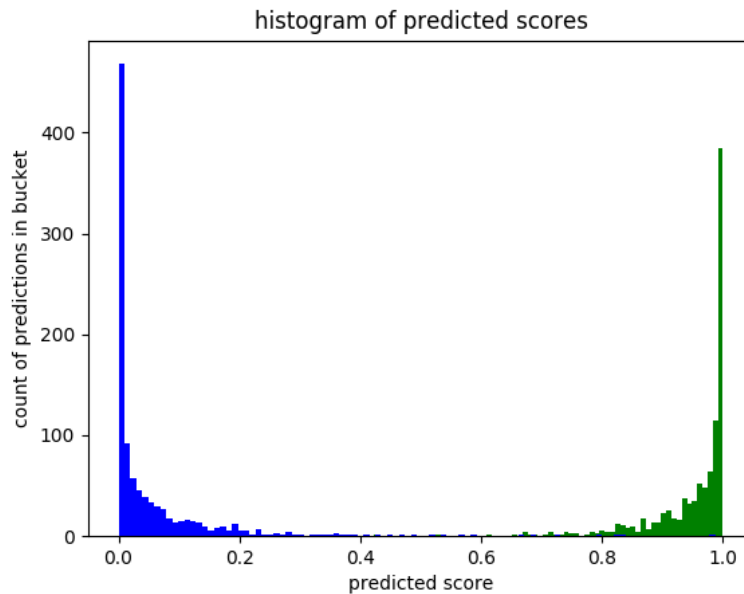
## Page 8 Predicted scores (10 points)

Show your code for plotting predicted scores and show the figure.

```
prob_all = lr.predict_proba(feature)
index_pos = np.where(label == 1)
pos = prob_all[index_pos]

index_neg = np.where(label == 0)
neg = prob_all[index_neg]

fig1 = plt.figure('prob')
plt.title('histogram of predicted scores')
plt.xlabel('predicted score')
plt.ylabel('count of predictions in bucket')
plt.hist(pos[:, 1], color='green', bins=100)
plt.hist(neg[:, 1], color='blue', bins=100)
plt.savefig('prob')
plt.show()
```





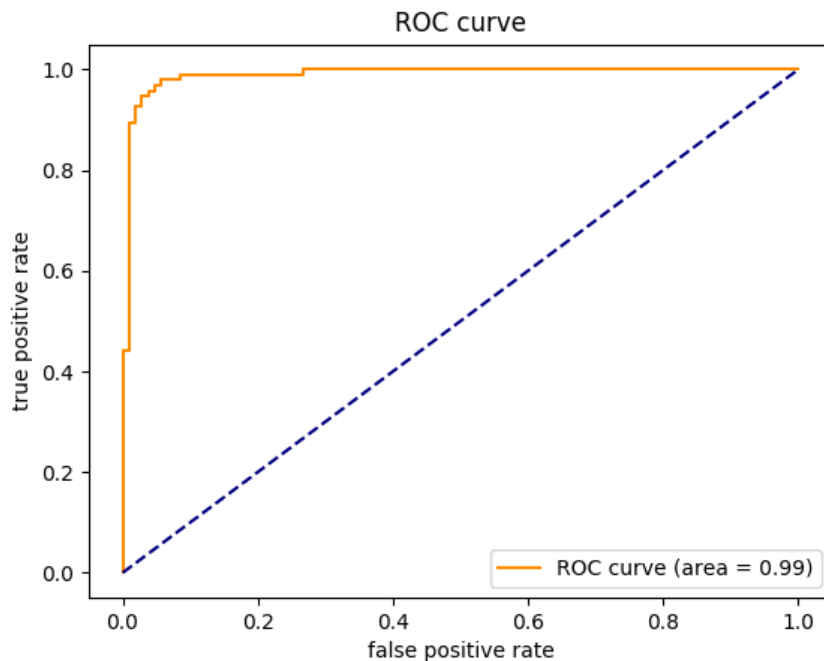
## Page 9 Accuracy again and curve (20 points)

Report the accuracy on train and test dataset with a different threshold. Explain why you choose that threshold.

Plot the ROC curve.

When threshold=0.6, accuracy on train set is 0.9983333333333333 and accuracy on test set is 0.96.

The reason to choose 0.6 is that according to the histogram of predicted scores, threshold should be between 0.4 and 0.6 so that we could better tell different labels apart. And after trying different values, 0.6 gives the best accuracy result.



## Page 10 Best threshold (10 points)

Choose the threshold that minimizes false positives while maximizing true positives. Explain your reason.

Threshold = 0.8789133271417255

False positive rate = 0.0380952380952381

True positive rate = 0.9578947368421052

The reason is that this result minimizes the distance between the corresponding point on the ROC curve and point (0,1) so that in this situation, it minimizes false positives and maximizes true positives.

```

import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import matplotlib.pyplot as plt
import pickle
from sklearn.feature_extraction import stop_words
from sklearn.metrics.pairwise import cosine_similarity
from scipy import spatial
import numpy as np
import random
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import NearestNeighbors

def accuracy_rate(model, threshold, feature, label):
    prob = model.predict_proba(feature)
    prob_pos = prob[:, 1]

    predict = []
    for p in prob_pos:
        if p > threshold:
            pre = 1
        else:
            pre = 0
        predict.append(pre)
    predict = np.array(predict)
    correct = np.count_nonzero(predict == label)
    rate = correct / len(label)

    return rate

```

```

if __name__ == '__main__':
    data_whole = pd.read_csv('yelp_2k.csv')
    data = data_whole.loc[:, ['stars', 'text']]
    data['label'] = data.stars.apply(lambda x: 0 if x==1 else 1)
    text = data.text
    label = data.label

##### first plot #####
cv = CountVectorizer(stop_words=None, lowercase=True)
cv_fit = cv.fit_transform(text)
count = cv_fit.toarray()
count_sum = count.sum(axis=0)
count_ordered = sorted(count_sum, reverse=True)
word = cv.get_feature_names()
word_dict = dict(zip(word, count_sum))
count_dict_sorted = sorted(word_dict.items(), key=lambda item: item[1])

plt.scatter(range(len(count_ordered)), count_ordered)
plt.title('Word Frequency')
plt.xlabel('Word Rank')
plt.ylabel('Word Count')
plt.savefig('word_frequency')
plt.close()

```

```

tf = TfidfTransformer()
tfidf = tf.fit_transform(cv_fit).toarray()
tfidf_sum = tfidf.sum(axis=0)
tfidf_dict = dict(zip(word, tfidf_sum))
tfidf_sorted = sorted(tfidf_dict.items(), key=lambda item: item[1], reverse=True)
print(tfidf_sorted[9:11])

stop_word = tfidf_sorted[0:10]
stop_list = []
for t in stop_word:
    stop_list.append(t[0])
max_count_stop = 4000
stop_list.extend(stop_words.ENGLISH_STOP_WORDS)

for i in range(len(count_dict_sorted) - 1, -1, -1):
    c = count_dict_sorted[i]
    if c[1] > max_count_stop:
        stop_list.append(c[0])
    else:
        break
# print(stop_list)
print(len(stop_list))
with open('stop_list.txt', 'w') as file:
    for word in stop_list:
        file.write(word)
        file.write(', ')
file.close()

```

```

min_list = []
min_count = 10
for c in count_dict_sorted:
    if c[1] < min_count:
        min_list.append(c[0])
    else:
        break
# print(len(min_list))
stop_list.extend(min_list)

##### modified plot #####
cv_modify = CountVectorizer(stop_words=stop_list, lowercase=True, max_df=5000)
cv_fit_modify = cv_modify.fit_transform(text)
count_modify = cv_fit_modify.toarray()
count_sum_modify = count_modify.sum(axis=0)
count_ordered_modify = sorted(count_sum_modify, reverse=True)
vocabulary = cv_modify.get_feature_names()
feature = count_modify
# print(len(vocabulary))

plt.scatter(range(len(count_ordered_modify)), count_ordered_modify)
plt.title('Word Frequency')
plt.xlabel('Word Rank')
plt.ylabel('Word Count')
plt.savefig('word_frequency_modify')
plt.close()

```

```

##### horrible service #####
sentence = ['Horrible customer service', ]
cv_sentence = CountVectorizer(lowercase=True, vocabulary=vocabulary)
cv_fit_sentence = cv_sentence.fit_transform(sentence)
vector_sentence = cv_fit_sentence.toarray()
# print(vector_sentence)

nbrs = NearestNeighbors(n_neighbors=1, algorithm='brute', metric='cosine')
nbrs.fit(vector_sentence)
distances, indices = nbrs.kneighbors(count_modify)
distances = distances.reshape(1, -1)
similarity = (1 - distances).reshape(1, -1)
index_sort = np.argsort(-similarity)
index_sort = index_sort[0]
index = index_sort[0:5]
distance_sort = distances[:, index]
print(distance_sort)

with open('text_horrible', 'a') as file:
    for i in index:
        # print(i)
        file.write(text.iloc[i])
        file.write('\n')
        file.write("#####")
        file.write('\n')
        # print(type(text.iloc[i]))
file.close()

```

```

# similarity_sort = np.sort(similarity)
# plt.scatter(range(similarity_sort.shape[1]), similarity_sort)
# plt.show()
# print(len(similarity_sort[similarity_sort > 0.4]))
distances_sort = np.sort(distances)
plt.scatter(range(distances_sort.shape[1]), distances_sort)
plt.title('cosine distances of all documents')
plt.xlabel('distance rank')
plt.ylabel('cosine distance')
plt.savefig('distances')
plt.show()
print(len(distances_sort[distances_sort < 0.6]))

##### part 3 #####
random.seed(0)
index_test = random.sample(list(range(len(feature))), 200)
index_train = list(set(list(range(len(feature)))) - set(index_test))
train = feature[index_train, :]
test = feature[index_test, :]
train_label = np.array(label.iloc[index_train])
test_label = np.array(label.iloc[index_test])

lr = LogisticRegression(solver='lbfgs')
lr.fit(train, train_label)
predict1 = lr.predict(test)
print(lr.score(test, test_label))
print(lr.score(train, train_label))

```

```

similarity_sort = np.sort(similarity)
plt.scatter(range(similarity_sort.shape[1]), similarity_sort)
plt.show()
print(len(similarity_sort[similarity_sort > 0.4]))

##### part 3 #####
random.seed(0)
index_test = random.sample(list(range(len(feature))), 200)
index_train = list(set(list(range(len(feature)))) - set(index_test))
train = feature[index_train, :]
test = feature[index_test, :]
train_label = np.array(label.iloc[index_train])
test_label = np.array(label.iloc[index_test])

lr = LogisticRegression(solver='lbfgs')
lr.fit(train, train_label)
predict1 = lr.predict(test)
print(lr.score(test, test_label))
print(lr.score(train, train_label))

```

```
##### plot #####
prob_all = lr.predict_proba(feature)
index_pos = np.where(label == 1)
pos = prob_all[index_pos]

index_neg = np.where(label == 0)
neg = prob_all[index_neg]

fig1 = plt.figure('prob')
plt.title('histogram of predicted scores')
plt.xlabel('predicted score')
plt.ylabel('count of predictions in bucket')
plt.hist(pos[:, 1], color='green', bins=100)
plt.hist(neg[:, 1], color='blue', bins=100)
plt.savefig('prob')
plt.show()

##### change threshold #####
threshold = 0.6

rate_train = accuracy_rate(lr, threshold, train, train_label)
rate_test = accuracy_rate(lr, threshold, test, test_label)

print(rate_train)
print(rate_test)
```

```
fpr, tpr, thresholds = roc_curve(test_label, lr.decision_function(test))
roc_auc = auc(fpr, tpr)
# print(len(threshold))
# print(type(fpr))
# print(fpr)
# print(fpr.shape)
# point = np.concatenate(fpr, tpr, axis=1)
# print(point)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.legend(loc="lower right")
plt.title('ROC curve')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.savefig('roc')
plt.show()

min_dist = float('inf')
for i in range(len(fpr)):
    dist = np.sqrt(np.square(fpr[i]-0) + np.square(tpr[i]-1))
    if dist < min_dist:
        min_dist = dist
        index = i
        fpr_min = fpr[i]
        tpr_min = tpr[i]
print(fpr_min)
print(tpr_min)
print(thresholds[index])
```