AML_HW2_Write up

Name: Scarlett Huang NetID: sh2557 Program: CM

Name:Zihan Zhang NetID: zz698 Program: ORIE

Programming

(a) Download the data

Download the train and test data from kaggle.

(b) Split the training data.

```
import pandas as pd
from sklearn.model_selection import train_test_split

train=pd.read_csv("train.csv")
x_train=train.drop('target',axis=1)#dataframe with index
y_train=train.target
X_train, X_test, y_train, y_test = train_test_split(x_train, y_train,
test_size=0.3, random_state=42)
Train = X_train.join(y_train)
Train.to_csv('split_train.csv', sep=',',header=True, index=False)
Test = X_test.join(y_test)
Test.to_csv('dev.csv', sep=',',header=True, index=False)
```

(c) Preprocess the Data.

I did the following:

1. Convert all the words to lowercase.
2. Lemmatize all the words
3. Strip punctuation.
4. Strip the stop words, e.g., “the”, “and”, “or”.
5. Srtip the urls.

```
import pandas as pd
import re

from nltk.tokenize import TweetTokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords

train=pd.read_csv("split_train.csv")
x_train=train.drop('target',axis=1)#dataframe with index
y_train=train.target

test=pd.read_csv("dev.csv")
x_test=test.drop('target',axis=1)#dataframe with index
y_test=test.target

tweettoken = TweetTokenizer(strip_handles=True, reduce_len=True)
stemmer=PorterStemmer() # stemming

train_pre=[]
test_pre=[]
data=[]
def preprocess(text,dataclass):
    url = re.compile(r'https?://\S+|www\.\S+')#delete url
    text_new=url.sub(r'',text)
    text_new=re.sub('[^a-zA-Z]'," ",text_new)#delete punctuation
    text_new=text_new.lower()#lowercase
    text_rest=tweettoken.tokenize(text_new)
    for i in text_rest:
        if i in stopwords.words('english'):#delete stopwords
            text_rest.remove(i)
    rest=[]
    for k in text_rest:
        rest.append(stemmer.stem(k))#lemmatize
    ret=" ".join(rest)
    if dataclass==1:
        train_pre.append(ret)
    elif dataclass==0:
        test_pre.append(ret)

def splitclass(data,q,m):
         for j in range(q):
                 preprocess(data["text"].iloc[j],m)

splitclass(x_train,5329,1)
splitclass(x_test,2284,0)

train_result = pd.DataFrame(train_pre)
train_result = train_result.join(y_train)
test_result = pd.DataFrame(test_pre)
test_result = test_result.join(y_test)

train_result.to_csv('train_pre.csv', sep=',',header=True, index=False)
test_result.to_csv('test_pre.csv', sep=',',header=True, index=False)
```

Before:

Ashes 2015: Australia Ì Ûªs collapse at Trent Bridge among worst in history: England bundled out Australia for 60 ... http://t.co/t5TrhjUAU0
GREAT MICHIGAN TECHNIQUE CAMP B1G THANKS TO @bmurph1019 @hail_Youtsey . @termn8r13 #GoBlue #WrestleOn http://t.co/OasKgki6Qj
CNN: Tennessee movie theater shooting suspect killed by police http://t.co/dI8ElZsWNR
Still rioting in a couple of hours left until I have to be up for class.
Crack in the path where I wiped out this morning during beach run. Surface wounds on left elbow and right knee. http://t.co/yaqRSximph

After:

ash australia collaps trent bridg among worst histori england bundl australia
great michigan techniqu camp b g thank bmurph hail youtsey termn r goblu wrestleon
cnn tennesse movi theater shoot suspect kill polic
still riot a coupl hour left i to up class
crack the path i wipe thi morn beach run surfac wound left elbow right knee

(d) Bag of Words model

description: use bag of words model for text vectorization

d-1: build the bag of words feature vectors for both the train and dev sets, and report the total number of features in these vectors.

- answer: total number of features = 2991.
- codes: see the codes and results in **Cell [4], [5]**.

d-2: decide on an appropriate threshold M , and discuss how you made this decision.

- answer: best threshold M=3. To decide on the best M, I use F1 Score to be the metric, i.e., the value that leads to the highest F1 Score is my best M. I set the value of M from 1 to 10 and run each of the four classification models respectively. Results show that when M=3, the Bernoulli Naive Bayes model gets the highest F1 Score (0.7611518915866743) among all models.
- codes: see codes and results in **Cell [15], [16], [17], [18]**.

d-1:

```
#[4]
# d-1: use bag of words model on train set.

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=3,
                             binary=True) #instantiate the CountVectorizer
class, set M=3
train_corpus = train_pre

train_bow_vectors = vectorizer.fit_transform(train_corpus) #use fit() to
create index, transform each document into a word frequency vector
print(type(train_bow_vectors)) #type: sparse vector
#print (train_bow_vectors)

train_bow_vectors = train_bow_vectors.toarray()
print (train_bow_vectors)
print(type(train_bow_vectors)) #type: np array

train_bow_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
train_bow_voca_dic = vectorizer.vocabulary_
print(train_bow_voca_list[:50])
print(list(train_bow_voca_dic.items())[:50])
#print('vocabulary list of trainset:', train_bow_voca_list)
#print( 'vocabulary dic of trainset:', train_bow_voca_dic)

print(train_bow_vectors.sum(axis=0)) #count each word' frequency in the corpus

X1 = train_bow_vectors.shape
```

```
print (X1)
train_bow_feature_num = X1[1] #vector length/number of features
print ('total number of features in trainset:', train_bow_feature_num)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
<class 'numpy.ndarray'>
['aa', 'aba', 'abandon', 'abbswinston', 'abc', 'abcnew', 'abil', 'abl',
'ablaz', 'about', 'absolut', 'abstorm', 'abus', 'accept', 'access', 'accid',
'accident', 'accionempresa', 'accord', 'account', 'accus', 'achiev', 'acid',
'acr', 'across', 'act', 'action', 'activ', 'actual', 'ad', 'add', 'address',
'admit', 'adopt', 'adult', 'advanc', 'advisori', 'af', 'affect', 'affili',
'afghan', 'afghanistan', 'afp', 'afraid', 'africa', 'after', 'afternoon',
'aftershock', 'ag', 'again']
[('ash', 169), ('australia', 192), ('collaps', 532), ('trent', 2721),
('bridg', 355), ('among', 99), ('worst', 2941), ('histori', 1240), ('england',
859), ('bundl', 381), ('great', 1140), ('michigan', 1652), ('techniqu', 2592),
('camp', 408), ('thank', 2617), ('hail', 1171), ('cnn', 523), ('tennesse',
2602), ('movi', 1714), ('theater', 2621), ('shoot', 2346), ('suspect', 2555),
('kill', 1428), ('polic', 1984), ('still', 2494), ('riot', 2205), ('coupl',
594), ('hour', 1264), ('left', 1483), ('to', 2665), ('up', 2788), ('class',
503), ('crack', 601), ('the', 2620), ('path', 1917), ('thi', 2631), ('morn',
1700), ('beach', 245), ('run', 2238), ('surfac', 2549), ('wound', 2945),
('right', 2203), ('knee', 1438), ('expert', 914), ('franc', 1042), ('begin',
262), ('examin', 897), ('airplan', 64), ('debri', 667), ('found', 1035)]
[ 3  8  8 ...  6  6 24]
(5329, 2991)
total number of features in trainset: 2991
```

```
#[5]
# d-1: use bag of words model on dev set.

test_corpus = test_pre

test_bow_vectors = vectorizer.transform(test_corpus) #use the same set of
tokens as trainset, transform each document into a word frequency vector
print(type(test_bow_vectors)) #type: sparse vector
#print (test_bow_vectors)

test_bow_vectors = test_bow_vectors.toarray()
print (test_bow_vectors)
print(type(test_bow_vectors)) #type: np array
```

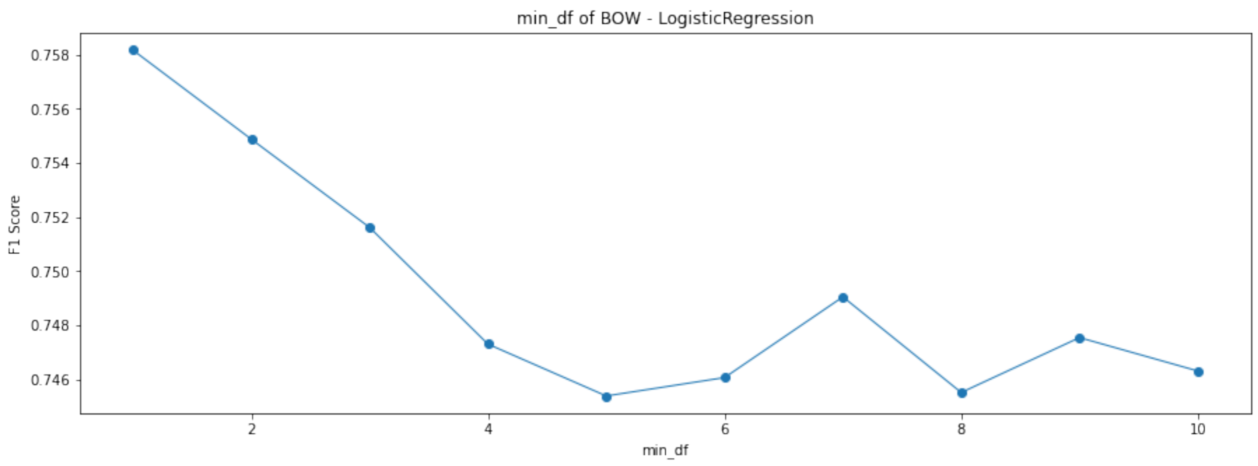
```
test_bow_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
test_bow_voca_dic = vectorizer.vocabulary_
print(test_bow_voca_list[:50])
print(list(test_bow_voca_dic.items())[:50])
#print('vocabulary list of dev set:', test_bow_voca_list)
#print( 'vocabulary dic of dev set:', test_bow_voca_dic)

print(test_bow_vectors.sum(axis=0)) #count each word' frequency in the corpus

X2 = test_bow_vectors.shape
print (X2)
test_bow_feature_num = X2[1] #vector length/number of features
print ('total number of features in dev set:', test_bow_feature_num)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
<class 'numpy.ndarray'>
['aa', 'aba', 'abandon', 'abbswinston', 'abc', 'abcnew', 'abil', 'abl',
'ablaz', 'about', 'absolut', 'abstorm', 'abus', 'accept', 'access', 'accid',
'accident', 'accionempresa', 'accord', 'account', 'accus', 'achiev', 'acid',
'acr', 'across', 'act', 'action', 'activ', 'actual', 'ad', 'add', 'address',
'admit', 'adopt', 'adult', 'advanc', 'advisori', 'af', 'affect', 'affili',
'afghan', 'afghanistan', 'afp', 'afraid', 'africa', 'after', 'afternoon',
'aftershock', 'ag', 'again']
[('ash', 169), ('australia', 192), ('collaps', 532), ('trent', 2721),
('bridg', 355), ('among', 99), ('worst', 2941), ('histori', 1240), ('england',
859), ('bundl', 381), ('great', 1140), ('michigan', 1652), ('techniqu', 2592),
('camp', 408), ('thank', 2617), ('hail', 1171), ('cnn', 523), ('tennesse',
2602), ('movi', 1714), ('theater', 2621), ('shoot', 2346), ('suspect', 2555),
('kill', 1428), ('polic', 1984), ('still', 2494), ('riot', 2205), ('coupl',
594), ('hour', 1264), ('left', 1483), ('to', 2665), ('up', 2788), ('class',
503), ('crack', 601), ('the', 2620), ('path', 1917), ('thi', 2631), ('morn',
1700), ('beach', 245), ('run', 2238), ('surfac', 2549), ('wound', 2945),
('right', 2203), ('knee', 1438), ('expert', 914), ('franc', 1042), ('begin',
262), ('examin', 897), ('airplan', 64), ('debri', 667), ('found', 1035)]
[ 0  6  6 ...  0  0 11]
(2284, 2991)
total number of features in dev set: 2991
```

d-2:

```
#[15]
# d-2: Decide on an appropriate threshold M for Bag of Words Model.

# 1.min_df of BOW - LogisticRegression

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt

f1_score_list = []
m_list = list(range(1,11))

for m in m_list:
    vectorizer = CountVectorizer(min_df=m,binary=True)
    train_bow_vectors = vectorizer.fit_transform(train_corpus)
    train_bow_vectors = train_bow_vectors.toarray()
    test_corpus = test_pre
    test_bow_vectors = vectorizer.transform(test_corpus)
    test_bow_vectors = test_bow_vectors.toarray()

    LR = LogisticRegression()
    LR.fit(train_bow_vectors,y_train)
    predict_lr=LR.predict(test_bow_vectors)

    f1 = f1_score(y_test,predict_lr)
    f1_score_list.append(f1)

print(f1_score_list)
max_f1 = max(f1_score_list)
print ('Max F1 Score =', max_f1)
print ('best threshold M =',f1_score_list.index(max_f1) + 1)

#best threshold M=1
#plot
a1 = m_list
b1 = f1_score_list
plt.figure(figsize=(15,5))
plt.plot(a1,b1,'o-',linewidth=1)
plt.xlabel("min_df")
plt.ylabel("F1 Score")
plt.title("min_df of BOW - LogisticRegression")
plt.show()
```

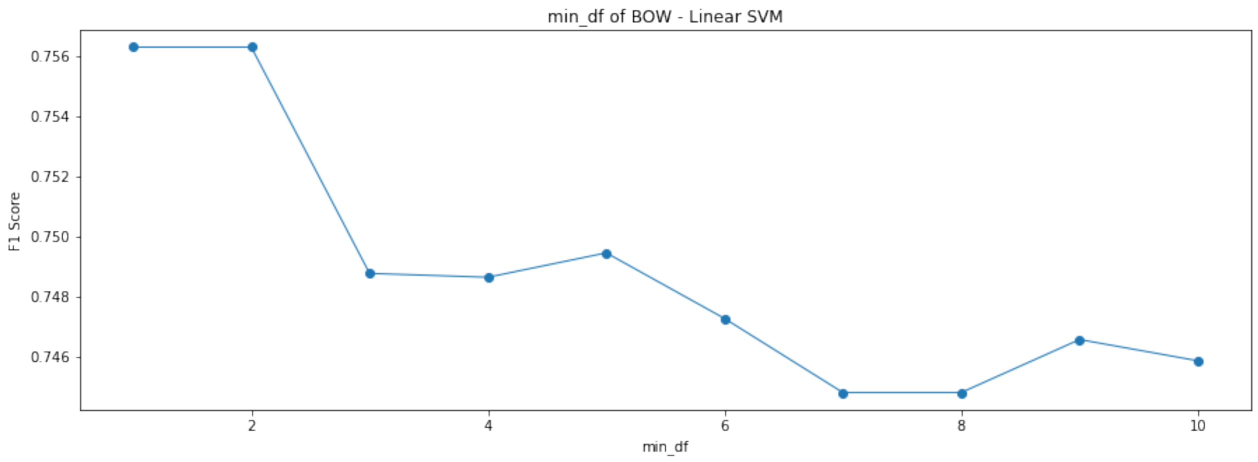
```
[0.758169934640523, 0.754880694143167, 0.7516198704103672, 0.7473118279569891,
0.7453954496208018, 0.7460747157552788, 0.749055585536967, 0.7455236028214868,
0.7475516866158868, 0.7463175122749591]
Max F1 Score = 0.758169934640523
best threshold M = 1
```



```python
#[16]
# d-2: Decide on an appropriate threshold M for Bag of Words Model.

# 2.min_df of BOW - Linear SVM

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt

f1_score_list = []
m_list = list(range(1,11))

for m in m_list:
    vectorizer = CountVectorizer(min_df=m,binary=True)
    train_bow_vectors = vectorizer.fit_transform(train_corpus)
    train_bow_vectors = train_bow_vectors.toarray()
    test_corpus = test_pre
    test_bow_vectors = vectorizer.transform(test_corpus)
    test_bow_vectors = test_bow_vectors.toarray()

    lsvm_model=LinearSVC(C=0.1)
    lsvm_model.fit(train_bow_vectors,y_train)
    predict_lsvm=lsvm_model.predict(test_bow_vectors)

    f1 = f1_score(y_test,predict_lsvm)
    f1_score_list.append(f1)
```

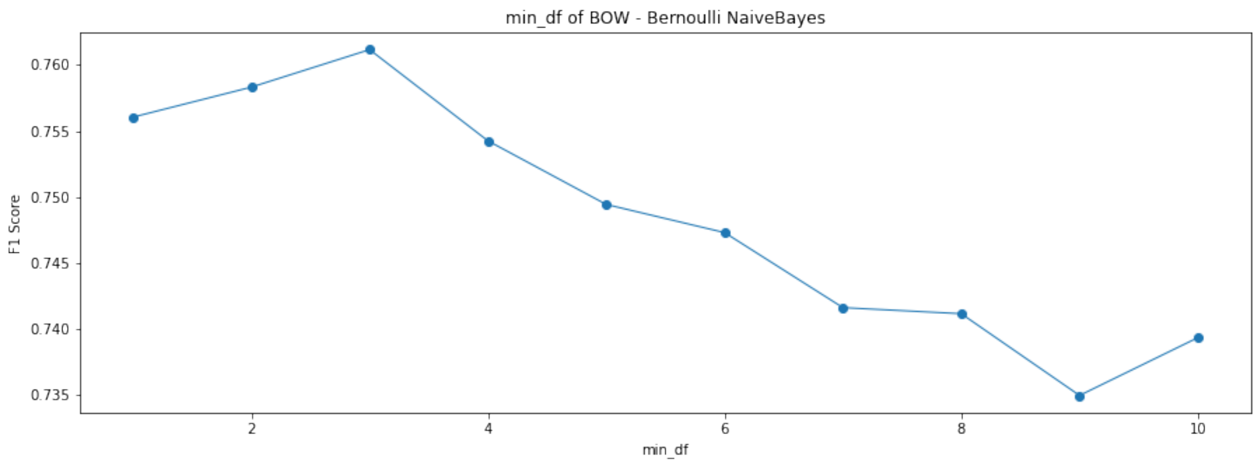
```
print(f1_score_list)
max_f1 = max(f1_score_list)
print ('Max F1 Score =', max_f1)
print ('best threshold M =',f1_score_list.index(max_f1) + 1)

#best threshold M=1
#plot
a2 = m_list
b2 = f1_score_list
plt.figure(figsize=(15,5))
plt.plot(a2,b2,'o-',linewidth=1)
plt.xlabel("min_df")
plt.ylabel("F1 Score")
plt.title("min_df of BOW - Linear SVM")
plt.show()
```

```
[0.7562841530054644, 0.7562841530054644, 0.7487738419618528,
0.7486457204767064, 0.7494553376906319, 0.747276688453159, 0.7448200654307524,
0.7448200654307524, 0.7465790914066777, 0.7458745874587459]
Max F1 Score = 0.7562841530054644
best threshold M = 1
```



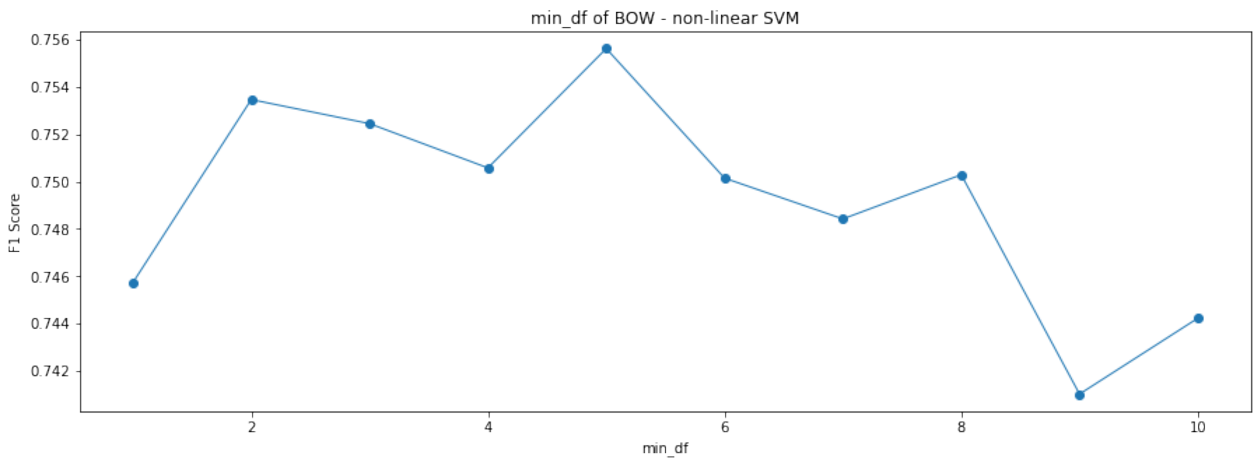
```
#[17]
# d-2: Decide on an appropriate threshold M for Bag of Words Model.

# 3.min_df of BOW - Bernoulli NaiveBayes

from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt

f1_score_list = []
m_list = list(range(1,11))

for m in m_list:
    vectorizer = CountVectorizer(min_df=m,binary=True)
    train_bow_vectors = vectorizer.fit_transform(train_corpus)
    train_bow_vectors = train_bow_vectors.toarray()
    test_corpus = test_pre
    test_bow_vectors = vectorizer.transform(test_corpus)
    test_bow_vectors = test_bow_vectors.toarray()

    BernoulliNB = Bernoulli_NaiveBayes()
    BernoulliNB.train(train_bow_vectors,y_train)
    y_pred = BernoulliNB.predict(test_bow_vectors)

    f1 = BernoulliNB.cal_f1_score(y_test,y_pred)
    f1_score_list.append(f1)

print(f1_score_list)
max_f1 = max(f1_score_list)
print ('Max F1 Score =', max_f1)
print ('best threshold M =',f1_score_list.index(max_f1) + 1)

#best threshold M=3
#plot
a2 = m_list
b2 = f1_score_list
plt.figure(figsize=(15,5))
plt.plot(a2,b2,'o-',linewidth=1)
plt.xlabel("min_df")
plt.ylabel("F1 Score")
plt.title("min_df of BOW - Bernoulli NaiveBayes")
plt.show()
```

```
[0.7560414269275029, 0.7583098591549297, 0.7611518915866743,
0.7542468856172139, 0.7494331065759636, 0.7473026689381034,
0.7416240772288472, 0.7411630558722919, 0.7349742415569548,
0.7393526405451448]
Max F1 Score = 0.7611518915866743
best threshold M = 3
```



```python
#[18]
# d-2: Decide on an appropriate threshold M for Bag of Words Model.

# 4.min_df of BOW - non-linear SVM

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt

f1_score_list = []
m_list = list(range(1,11))

for m in m_list:
    vectorizer = CountVectorizer(min_df=m,binary=True)
    train_bow_vectors = vectorizer.fit_transform(train_corpus)
    train_bow_vectors = train_bow_vectors.toarray()
    test_corpus = test_pre
    test_bow_vectors = vectorizer.transform(test_corpus)
    test_bow_vectors = test_bow_vectors.toarray()

    SVM=SVC(C=1,kernel='rbf',gamma=0.1)
    SVM.fit(train_bow_vectors,y_train)
    predict_svm=SVM.predict(test_bow_vectors)

    f1 = f1_score(y_test,predict_svm)
    f1_score_list.append(f1)

print(f1_score_list)
max_f1 = max(f1_score_list)
print ('Max F1 Score =', max_f1)
print ('best threshold M =',f1_score_list.index(max_f1) + 1)

#best threshold M=5
#plot
a4 = m_list
b4 = f1_score_list
plt.figure(figsize=(15,5))
plt.plot(a4,b4,'o-',linewidth=1)
plt.xlabel("min_df")
plt.ylabel("F1 Score")
plt.title("min_df of BOW - non-linear SVM")
plt.show()
```

```
[0.7457428068115091, 0.7534562211981567, 0.752441125789776,
0.7505747126436781, 0.7556066705002876, 0.7501435956346928,
0.7484240687679083, 0.7502876869965478, 0.7410404624277457,
0.7442396313364056]
Max F1 Score = 0.7556066705002876
best threshold M = 5
```



(e) Implement a naive Bayes classifier - BernoulliNB Naive Bayes

e-1: implement the BernoulliNB Naive Bayes classifier, without using any existing machine learning libraries.

- codes: see the codes in **Cell [6]**.

e-2: Train this classifier on the train set, and report its mean F 1-score on the dev set.

- codes: see the codes and results in **Cell [7]**.

```
#[6]
# e-1: implement the BernoulliNB Naive Bayes classifier, without using any
existing machine learning libraries.

import numpy as np

class Bernoulli_NaiveBayes:

    def __init__(self):
        self.alpha = 1 # set smoothing factor=1(Laplace Smoothing), to avoid
zero probability problems

    def _cal_prior_prob_log(self, y, classes): # calculate the logarithm of
prior probability of each class, P(y=c_k)
        self.classes = np.unique(y)
        class_num = len(self.classes) #count the number of possible types of y
        sample_num = len(y)

        c_num = np.count_nonzero(y == classes[:, None], axis=1) #count sample
amount of each class
        prior_prob = (c_num + self.alpha) / (sample_num + class_num *
self.alpha) #calculate prior probabilities(add smoothing correction)
        prior_prob_log = np.log(prior_prob) #calculate logarithm

        return prior_prob_log

    def _cal_condi_prob_log(self, X, y, classes): #calculate the logarithm of
all conditional probabilities P(x^(j)|y=c_k)

        n = (X.shape)[1]
        K = len(classes)

        #create an empty multidimensional array
        #prob_log: logarithmic matrix of two conditional probabilities
        condi_prob_log = np.empty((2, K, n))

        for k, c in enumerate(classes):
            X_c = X[np.equal(y, c)] #acquire all samples of class c_k
            total_num = len(X_c)
            num_f1 = np.count_nonzero(X_c, axis=0) #count the number of
samples of which feature value is 1
            condi_prob_f1 = (num_f1 + self.alpha) / (total_num + self.alpha *
2) #calculate conditional probability P(x^(j)=1|y=c_k)

            #calculate and store logarithm into matrix
            #prob_log[0]: store all values of log(P(x^(j)=0|y=c_k))
            #prob_log[1]: store all values of log(P(x^(j)=1|y=c_k))
            condi_prob_log[0, k] = np.log(1 - condi_prob_f1)
            condi_prob_log[1, k] = np.log(condi_prob_f1)

        return condi_prob_log

    def train(self, x_train, y_train): #train the model
        self.classes = np.unique(y_train) #acquire all classes
        self.prior_prob_log = self._cal_prior_prob_log(y_train, self.classes)
#calculate and store the logarithm of all prior probabilities
        self.condi_prob_log = self._cal_condi_prob_log(x_train, y_train,
self.classes) #calculate and store the logarithm of all conditional
probabilities

    def _predict_single_sample(self, x): #predict the label of single sample

        K = len(self.classes)
        po_prob_log = np.empty(K) #create an empty multidimensional array
        index_f1 = x == 1 #acquire index of feature value=1
        index_f0 = ~index_f1 #acquire index of feature value=0

        for k in range(K): #iterate each class
            #calculate the logarithm of the numerator of the posterior
probability
            po_prob_log[k] = self.prior_prob_log[k] \
                                 + np.sum(self.condi_prob_log[0, k][index_f0])
\
                                 + np.sum(self.condi_prob_log[1, k][index_f1])

        label = np.argmax(po_prob_log) #get the class with the highest
posterior probability
        return label

    def predict(self, X): #predict samples (include single sample)

        if X.ndim == 1: #if only predict single sample (the dimension of the
array = 1), invoke _predict_single_sample()
            return self._predict_single_sample(X)
        else:
            #if predict multiple samples, loop call _predict_single_sample()
and return a list of the predicted results
            labels = []
            for j in range(X.shape[0]):
                label = self._predict_single_sample(X[j])
                labels.append(label)
            return labels

    def cal_f1_score(self,true,predict):

        true = list(true)
        num = len(true)
        TP = 0
        FP = 0
        TN = 0
        FN = 0

        for i in range(num):
            if true[i] != predict[i]:
                if true[i] == 1:
                    FN += 1
                else:
                    FP += 1
            else:
                if true[i] == 1:
                    TP += 1
                else:
                    TN += 1
        precision = TP / (TP + FP)
        recall = TP / (TP + FN)
        F1_Score = 2 * (precision * recall) / (precision + recall)
        return F1_Score
```

```
#[7]
# e-2: Train the Bernoulli_NaiveBayes classifier on the train set, and report
its mean F 1-score on the dev set.

x_train = train_bow_vectors
y_train = np.array(y_train)
x_test = test_bow_vectors

BernoulliNB = Bernoulli_NaiveBayes()
BernoulliNB.train(x_train,y_train)
y_pred = BernoulliNB.predict(x_test)

print (BernoulliNB.cal_f1_score(y_test,y_pred))
```

```
0.7611518915866743
```

(f) Logistic regression prediction

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

#LR model
LR = LogisticRegression()
LR.fit(train_vectors,y_train)
predict_lr=LR.predict(test_vectors)
print(f1_score(y_test,predict_lr))
#0.7566684812193795

#Coefficient
conf_mat = confusion_matrix(y_test, predict_lr)
print(conf_mat)
print(classification_report(y_test, predict_lr))
coef_lr=LR.coef_[0]
maxindex_lr = np.argmax(coef_lr )
minindex_lr = np.argmin(coef_lr )

for i in train_voca_dic.keys():
    if train_voca_dic[i]==maxindex_lr:
        print('is real',i)
    if train_voca_dic[i]==minindex_lr:
        print('not real',i)
# is real: hiroshima
# not real: better
```

Result:

```
0.7566684812193795
[[1142  176]
 [ 271  695]]
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      1318
           1       0.80      0.72      0.76       966

    accuracy                           0.80      2284
   macro avg       0.80      0.79      0.80      2284
weighted avg       0.80      0.80      0.80      2284

not real better
is real hiroshima
```

The F1-score is 0.7566684812193795.

The most important words for deciding whether a tweet is about a real diaster or not using logistic regression is "better" and "hiroshima"

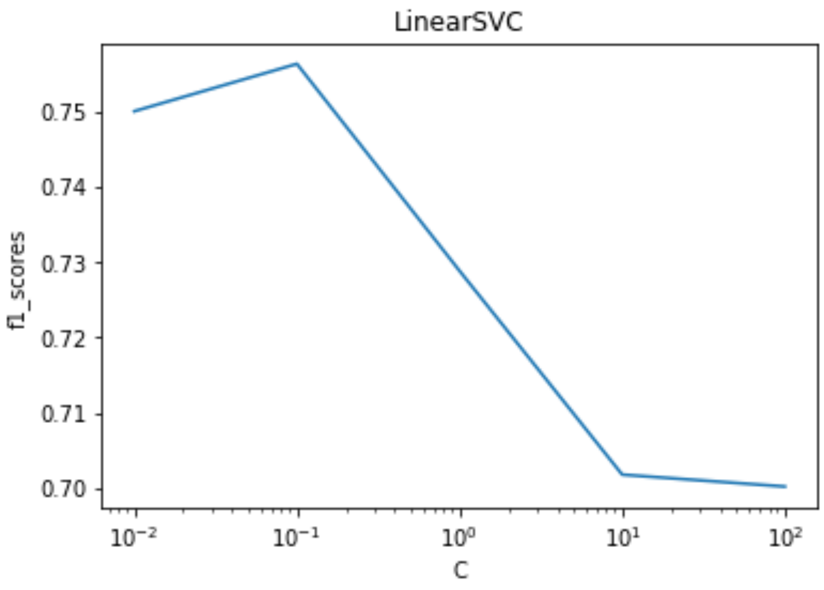
(g) Linear SVM prediction

```
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score

Cs=[0.01, 0.1, 1.0, 10.0, 100.0]
f1_scores_l=[]

for C in Cs:
    LSVM=LinearSVC(C=C,max_iter=100000)
    LSVM.fit(train_vectors,y_train)
    predict_lsvm=LSVM.predict(test_vectors)
    f1_scores_l.append(f1_score(y_test,predict_lsvm))
    print(f1_score(y_test,predict_lsvm))

## plot
fig1=plt.figure()
ax=fig1.add_subplot(1,1,1)
ax.plot(Cs,f1_scores_l)
ax.set_xlabel(r"C")
```

```python
ax.set_ylabel(r"f1_scores")
ax.set_xscale('log')
ax.set_title("LinearSVC")
plt.show()

# Linear SVM model
lsvm_model=LinearSVC(C=0.1)
lsvm_model.fit(train_vectors,y_train)
predict_lsvm=lsvm_model.predict(test_vectors)
print(f1_score(y_test,predict_lsvm))

# coefficient
coef_lsvm=lsvm_model.coef_[0]
maxindex_lsvm = np.argmax(coef_lsvm )
minindex_lsvm = np.argmin(coef_lsvm )
for i in train_voca_dic.keys():
    if train_voca_dic[i]==maxindex_lr:
        print('is real',i)
    if train_voca_dic[i]==minindex_lr:
        print('not real',i)

# 0.7562841530054644
# is real: hiroshima
# not real: better
```

Result:

```
0.75
0.7562841530054644
0.7288225892381459
0.7017913593256059
0.70020964360587
```



C=0.1results in the best classification performance, and the best linear SVM classifier is a little bit worse tan the logistic regression classifer.

The F1-score is 0.7562841530054644

The most important words for deciding whether a tweet is about a real diaster or not using logistic regression is "better" and "hiroshima". These are the same from the most important words for logistic regression.

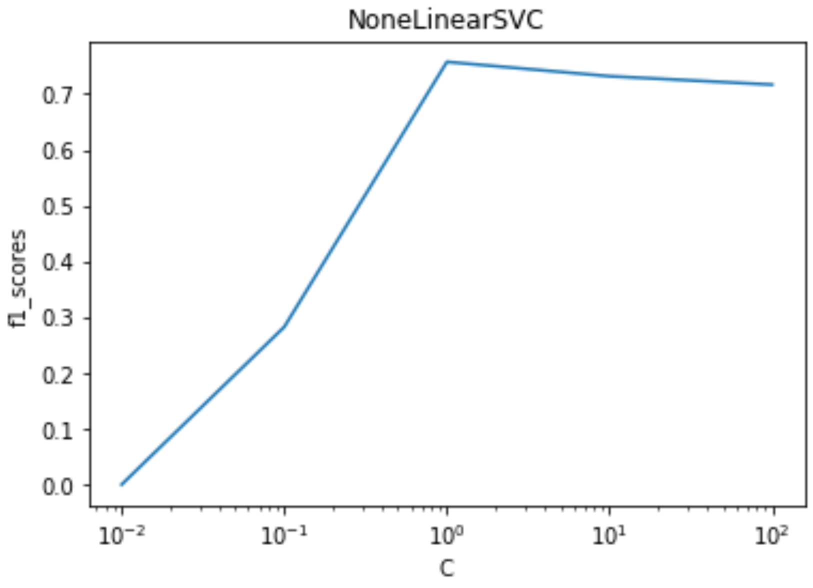
(h) Non-linear SVM prediction

```
from sklearn.svm import SVC
from sklearn.metrics import f1_score

G=[0.01,0.1,1,5,10]
for g in G:
    SVM=SVC(C=1,kernel='rbf',gamma=g)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    print(f1_score(y_test,predict_svm))

G=[0.05,0.08,0.11,0.14,0.17]
for g in G:
    SVM=SVC(C=1,kernel='rbf',gamma=g)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    print(f1_score(y_test,predict_svm))

G=[0.135,0.15,0.16]
for g in G:
    SVM=SVC(C=1,kernel='rbf',gamma=g)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    print(f1_score(y_test,predict_svm))
# best gamma=0.15

Cs_svm=[0.01, 0.1, 1.0, 10.0, 100.0]
f1_scores_nl=[]

for c in Cs_svm:
    SVM=SVC(C=c,kernel='rbf',gamma=0.15,max_iter=100000)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    f1_scores_nl.append(f1_score(y_test,predict_svm))
    print(f1_score(y_test,predict_svm))
# best C=1

## plot
```

```
fig2=plt.figure()
ax=fig2.add_subplot(1,1,1)
ax.plot(Cs_svm,f1_scores_nl)
ax.set_xlabel(r"C")
ax.set_ylabel(r"f1_scores")
ax.set_xscale('log')
ax.set_title("NoneLinearSVC")
plt.show()

# SVM model
SVM=SVC(C=1,kernel='rbf',gamma=0.15)
SVM.fit(train_vectors,y_train)
predict_svm=SVM.predict(test_vectors)
print(f1_score(y_test,predict_svm))
# 0.7573149741824441
```

```
0.6614069690992767
0.7556066705002876
0.3588039867109634
0.3012048192771084
0.3012048192771084
0.7491207502930832
0.7534562211981567
0.7558874210224009
0.756880733944954
0.7551963048498845
```

```
0.7571592210767468
0.7573149741824441
0.7556066705002876
```



```
0.0
0.2824156305506217
0.7573149741824441
0.7315993359158826
0.7165483342435827
```

Gamma=0.15 and C=1results in the best classification performance.

The F1-score is 0.7573149741824441

(i) N-gram model

description: use N-gram model for text vectorization

i-1: Using N = 2, construct feature representations of the tweets in the train and dev tweets.

- answer: total number of features = 4849.
- codes: see the codes and results in **Cell [8], [9]**.

i-2: Randomly sample 10 2-grams from your vocabulary, and print them out.

- codes: see the codes and results in **Cell [10]**.

i-3: Choose a threshold M, and only include symbols in the vocabulary that occur in at least M different tweets in the train set. Discuss how you chose the threshold M.

- answer: best threshold M=3. To decide on the best M, I use F1 Score to be the metric, i.e., the value that leads to the highest F1 Score is my best M. I set the value of M from 1 to 10 and run each of the four classification models respectively. Results show that when M=3, the Non-linear SVM model gets the highest F1 Score (0.7555040556199305) among all models.
- codes: see the codes and results in **Cell [19], [20], [21], [22]**.

i-4: Repeat parts (e)-(h), and report the results.

- codes: see the codes and results in **Cell [11], [12], [13], [14]**.

i-5: Do these results differ significantly from those using the bag of words model? Discuss what this implies about the task.

- answer: According to the results printed in Cell [15]~[22], the F1 scores of using the N-gram model are generally slightly lower than those using the bag of words model. The range of this difference is not significantly large, not greater than 0.04. This implies that Bag of Words model and N-gram model have similar performance on this short text classification task.

```
#[8]
# i-1: use N-gram model (N=2) on train set.

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=3,
                             ngram_range=(1,2),
                             binary=True) #instantiate the CountVectorizer
class, set M=3

train_corpus = train_pre
train_ng_vectors = vectorizer.fit_transform(train_corpus) #use fit() to create
index,transform each document into a word frequency vector
print(type(train_ng_vectors)) #type: sparse vector
#print (train_ng_vectors)

train_ng_vectors = train_ng_vectors.toarray()
print (train_ng_vectors)
print(type(train_ng_vectors)) #type: np array

train_ng_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
train_ng_voca_dic = vectorizer.vocabulary_
print(train_ng_voca_list[:50])
print(list(train_ng_voca_dic.items())[:50])
#print('vocabulary list of trainset:', train_ng_voca_list)
#print( 'vocabulary dic of trainset:', train_ng_voca_dic)

print(train_ng_vectors.sum(axis=0)) #count each word' frequency in the corpus

X3 = train_ng_vectors.shape
print (X3)
train_ng_feature_num = X3[1] #vector length/number of features
print ('total number of features in trainset:', train_ng_feature_num)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
<class 'numpy.ndarray'>
['aa', 'aba', 'aba woman', 'abandon', 'abandon aircraft', 'abbswinston',
'abbswinston zionist', 'abc', 'abc news', 'abcnew', 'abil', 'abl', 'ablaz',
'about', 'absolut', 'abstorm', 'abus', 'accept', 'access', 'access the',
'accid', 'accid expert', 'accid indian', 'accid man', 'accid properti',
'accident', 'accionempresa', 'accord', 'account', 'account hiroshima',
'accus', 'achiev', 'acid', 'acr', 'across', 'act', 'action', 'action hostag',
'action year', 'activ', 'activ municip', 'actual', 'ad', 'ad video', 'add',
'address', 'admit', 'admit arson', 'adopt', 'adult']
```

```
[('ash', 252), ('australia', 287), ('collaps', 820), ('trent', 4372),
('bridg', 547), ('among', 135), ('worst', 4752), ('histori', 1944),
('england', 1312), ('bundl', 585), ('ash australia', 253), ('australia
collaps', 288), ('collaps trent', 824), ('trent bridg', 4373), ('among worst',
136), ('great', 1782), ('michigan', 2634), ('techniqu', 4082), ('camp', 634),
('thank', 4120), ('hail', 1826), ('cnn', 805), ('tennesse', 4095), ('movi',
2728), ('theater', 4209), ('shoot', 3728), ('suspect', 4025), ('kill', 2250),
('polic', 3192), ('movi theater', 2729), ('theater shoot', 4211), ('suspect
kill', 4027), ('still', 3937), ('riot', 3508), ('coupl', 919), ('hour', 1993),
('left', 2353), ('to', 4271), ('up', 4478), ('class', 784), ('crack', 926),
('the', 4126), ('path', 3084), ('thi', 4224), ('morn', 2705), ('beach', 373),
('run', 3556), ('surfac', 4017), ('wound', 4758), ('right', 3506)]
[ 3  8  8 ...  6  4 24]
(5329, 4849)
total number of features in trainset: 4849
```

```
#[9]
# i-1: use N-gram model (N=2) on dev set.

test_corpus = test_pre
test_ng_vectors = vectorizer.transform(test_corpus) #use the same set of
tokens as trainset, transform each document into a word frequency vector
print(type(test_ng_vectors)) #type: sparse vector
#print (test_ng_vectors)

test_ng_vectors = test_ng_vectors.toarray()
print (test_ng_vectors)
print(type(test_ng_vectors)) #type: np array

test_ng_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
test_ng_voca_dic = vectorizer.vocabulary_
print(test_ng_voca_list[:50])
print(list(test_ng_voca_dic.items())[:50])
#print('vocabulary list of trainset:', test_ng_voca_list)
#print( 'vocabulary dic of trainset:', test_ng_voca_dic)

print(test_ng_vectors.sum(axis=0)) #count each word' frequency in the corpus

X4 = test_ng_vectors.shape
print (X4)
test_ng_feature_num = X4[1] #vector length/number of features
print ('total number of features in dev set:', test_ng_feature_num)
```

```
<class 'scipy.sparse.csr.csr_matrix'>
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
<class 'numpy.ndarray'>
['aa', 'aba', 'aba woman', 'abandon', 'abandon aircraft', 'abbswinston',
'abbswinston zionist', 'abc', 'abc news', 'abcnew', 'abil', 'abl', 'ablaz',
'about', 'absolut', 'abstorm', 'abus', 'accept', 'access', 'access the',
'accid', 'accid expert', 'accid indian', 'accid man', 'accid properti',
'accident', 'accionempresa', 'accord', 'account', 'account hiroshima',
'accus', 'achiev', 'acid', 'acr', 'across', 'act', 'action', 'action hostag',
'action year', 'activ', 'activ municip', 'actual', 'ad', 'ad video', 'add',
'address', 'admit', 'admit arson', 'adopt', 'adult']
[('ash', 252), ('australia', 287), ('collaps', 820), ('trent', 4372),
('bridg', 547), ('among', 135), ('worst', 4752), ('histori', 1944),
('england', 1312), ('bundl', 585), ('ash australia', 253), ('australia
collaps', 288), ('collaps trent', 824), ('trent bridg', 4373), ('among worst',
136), ('great', 1782), ('michigan', 2634), ('techniqu', 4082), ('camp', 634),
('thank', 4120), ('hail', 1826), ('cnn', 805), ('tennesse', 4095), ('movi',
2728), ('theater', 4209), ('shoot', 3728), ('suspect', 4025), ('kill', 2250),
('polic', 3192), ('movi theater', 2729), ('theater shoot', 4211), ('suspect
kill', 4027), ('still', 3937), ('riot', 3508), ('coupl', 919), ('hour', 1993),
('left', 2353), ('to', 4271), ('up', 4478), ('class', 784), ('crack', 926),
('the', 4126), ('path', 3084), ('thi', 4224), ('morn', 2705), ('beach', 373),
('run', 3556), ('surfac', 4017), ('wound', 4758), ('right', 3506)]
[ 0  6  5 ...  0  0 11]
(2284, 4849)
total number of features in dev set: 4849
```

```
#[10]
# i-2: Randomly sample 10 2-grams from your vocabulary, and print them out.

import random

slice = []
for word in random.sample(test_ng_voca_list,100):
    if bool(' ' in word) == True:
        slice.append(word)
        if len(slice) == 10:
            break
print (slice)
```

```
['get destroy', 'evacu order', 'take wht', 'you go', 'hurrican drought', 'plan
hijack', 'lightn caus', 'buse arrest', 'crash there', 'saw coach']
```

```
#[11]
# i-4: use N-gram model, repeat parts (e)-(h), and report the results.

# 1.Bernoulli NaiveBayes model
y_train = np.array(y_train)
BernoulliNB = Bernoulli_NaiveBayes()
BernoulliNB.train(train_ng_vectors,y_train)
y_pred = BernoulliNB.predict(test_ng_vectors)
print (BernoulliNB.cal_f1_score(y_test,y_pred))
```

```
0.7282809611829946
```

```
#[12]
# i-4: use N-gram model, repeat parts (e)-(h), and report the results.

# 2.LR model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

LR = LogisticRegression()
LR.fit(train_ng_vectors,y_train)
predict_lr=LR.predict(test_ng_vectors)
print(f1_score(y_test,predict_lr))
```

```
0.7546346782988005
```

```
#[13]
# i-4: use N-gram model, repeat parts (e)-(h), and report the results.

# 3.Linear SVM model
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score

lsvm_model=LinearSVC(C=0.1)
lsvm_model.fit(train_ng_vectors,y_train)
predict_lsvm=lsvm_model.predict(test_ng_vectors)
print(f1_score(y_test,predict_lsvm))
```

```
0.7526055951727921
```

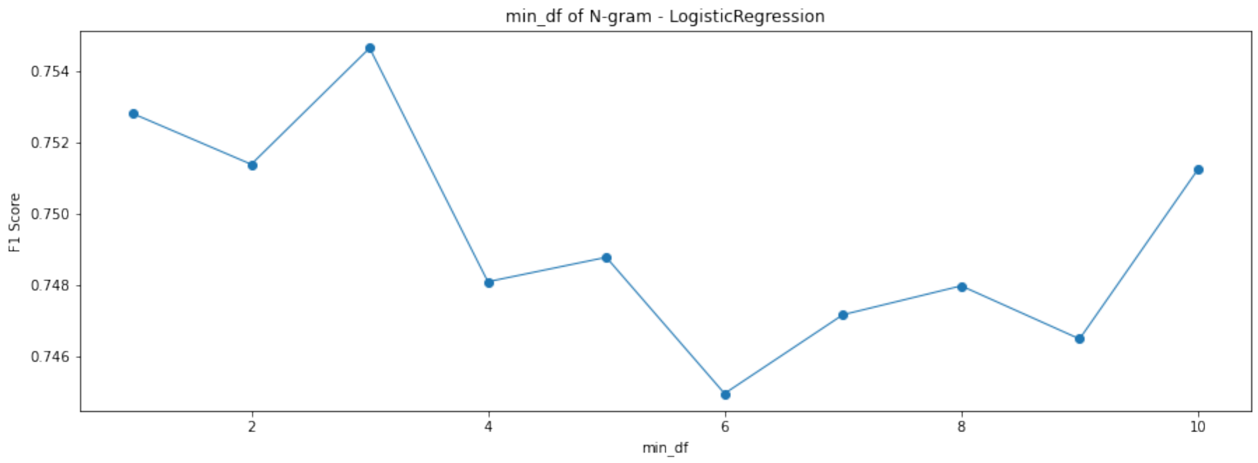
```
#[14]
# i-4: use N-gram model, repeat parts (e)-(h), and report the results.

# 4.Non-linear SVM model
from sklearn.svm import SVC
from sklearn.metrics import f1_score

SVM=SVC(C=1,kernel='rbf',gamma=0.1)
SVM.fit(train_ng_vectors,y_train)
predict_svm=SVM.predict(test_ng_vectors)
print(f1_score(y_test,predict_svm))
```

```
0.7555040556199305
```

```
#[19]
# i-3: Decide on an appropriate threshold M for N-gram Model

# 1.min_df of N-gram - LogisticRegression

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt

f1_score_list = []
m_list = list(range(1,11))

for m in m_list:
    vectorizer = CountVectorizer(min_df=m,ngram_range=(1,2),binary=True)
    train_ng_vectors = vectorizer.fit_transform(train_corpus)
    train_ng_vectors = train_ng_vectors.toarray()
    test_corpus = test_pre
    test_ng_vectors = vectorizer.transform(test_corpus)
    test_ng_vectors = test_ng_vectors.toarray()

    LR = LogisticRegression()
    LR.fit(train_ng_vectors,y_train)
    predict_lr=LR.predict(test_ng_vectors)

    f1 = f1_score(y_test,predict_lr)
    f1_score_list.append(f1)

print(f1_score_list)
max_f1 = max(f1_score_list)
print ('Max F1 Score =', max_f1)
print ('best threshold M =',f1_score_list.index(max_f1) + 1)

#best threshold M=3
```

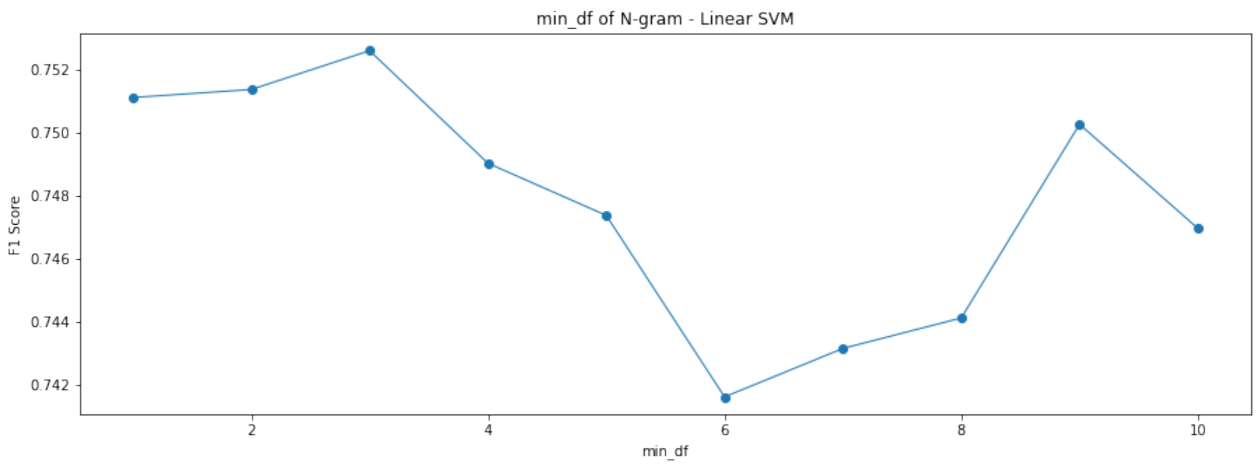
```
#plot
a1 = m_list
b1 = f1_score_list
plt.figure(figsize=(15,5))
plt.plot(a1,b1,'o-',linewidth=1)
plt.xlabel("min_df")
plt.ylabel("F1 Score")
plt.title("min_df of N-gram - LogisticRegression")
plt.show()
```

```
[0.7527964205816554, 0.7513751375137514, 0.7546346782988005,
0.7480832420591457, 0.748768472906404, 0.7449481157837247, 0.7471575527883053,
0.7479674796747968, 0.7464788732394366, 0.7512301804264625]
Max F1 Score = 0.7546346782988005
best threshold M = 3
```



```
#[20]
# i-3: Decide on an appropriate threshold M for N-gram Model

# 2.min_df of N-gram - Linear SVM

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt

f1_score_list = []
m_list = list(range(1,11))

for m in m_list:
    vectorizer = CountVectorizer(min_df=m,ngram_range=(1,2),binary=True)
    train_ng_vectors = vectorizer.fit_transform(train_corpus)
    train_ng_vectors = train_ng_vectors.toarray()
    test_corpus = test_pre
    test_ng_vectors = vectorizer.transform(test_corpus)
    test_ng_vectors = test_ng_vectors.toarray()
    lsvm_model=LinearSVC(C=0.1)
    lsvm_model.fit(train_ng_vectors,y_train)
    predict_lsvm=lsvm_model.predict(test_ng_vectors)

    f1 = f1_score(y_test,predict_lsvm)
    f1_score_list.append(f1)

print(f1_score_list)
max_f1 = max(f1_score_list)
print ('Max F1 Score =', max_f1)
print ('best threshold M =',f1_score_list.index(max_f1) + 1)

#best threshold M=3
#plot
a2 = m_list
b2 = f1_score_list
plt.figure(figsize=(15,5))
plt.plot(a2,b2,'o-',linewidth=1)
plt.xlabel("min_df")
plt.ylabel("F1 Score")
plt.title("min_df of N-gram - Linear SVM")
plt.show()
```

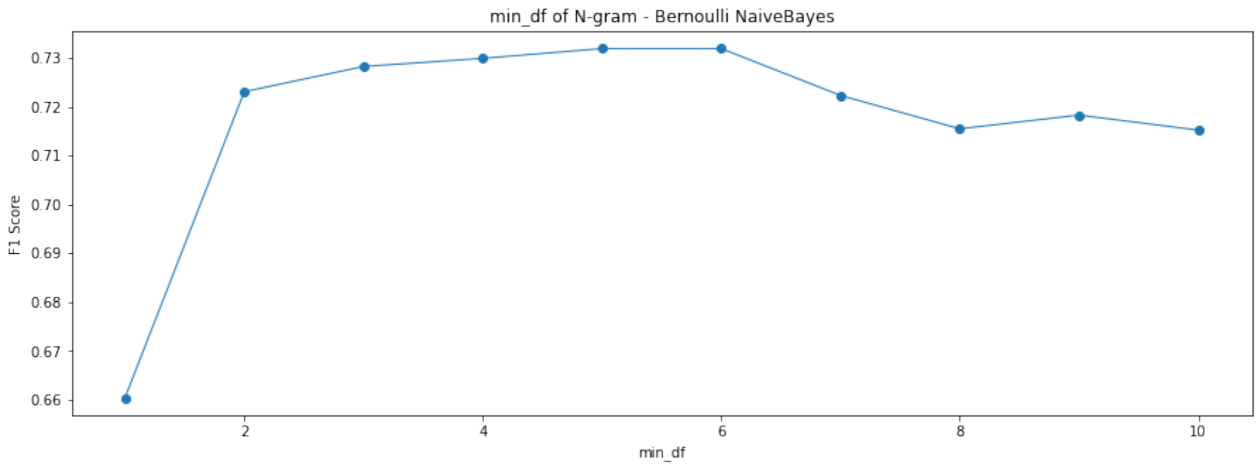
```
[0.7511210762331839, 0.7513751375137514, 0.7526055951727921,
0.7490389895661723, 0.7473858007705009, 0.741634668129457, 0.7431693989071039,
0.7441352973267866, 0.7502726281352236, 0.746974697469747]
Max F1 Score = 0.7526055951727921
best threshold M = 3
```



```
#[21]
# i-3: Decide on an appropriate threshold M for N-gram Model

# 3.min_df of N-gram - Bernoulli NaiveBayes

from sklearn.feature_extraction.text import CountVectorizer
```

```python
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt

f1_score_list = []
m_list = list(range(1,11))

for m in m_list:
    vectorizer = CountVectorizer(min_df=m,ngram_range=(1,2),binary=True)
    train_ng_vectors = vectorizer.fit_transform(train_corpus)
    train_ng_vectors = train_ng_vectors.toarray()
    test_corpus = test_pre
    test_ng_vectors = vectorizer.transform(test_corpus)
    test_ng_vectors = test_ng_vectors.toarray()

    BernoulliNB = Bernoulli_NaiveBayes()
    BernoulliNB.train(train_ng_vectors,y_train)
    y_pred = BernoulliNB.predict(test_ng_vectors)

    f1 = BernoulliNB.cal_f1_score(y_test,y_pred)
    f1_score_list.append(f1)

print(f1_score_list)
max_f1 = max(f1_score_list)
print ('Max F1 Score =', max_f1)
print ('best threshold M =',f1_score_list.index(max_f1) + 1)

#best threshold M=5
#plot
a2 = m_list
b2 = f1_score_list
plt.figure(figsize=(15,5))
plt.plot(a2,b2,'o-',linewidth=1)
plt.xlabel("min_df")
plt.ylabel("F1 Score")
plt.title("min_df of N-gram - Bernoulli NaiveBayes")
plt.show()
```

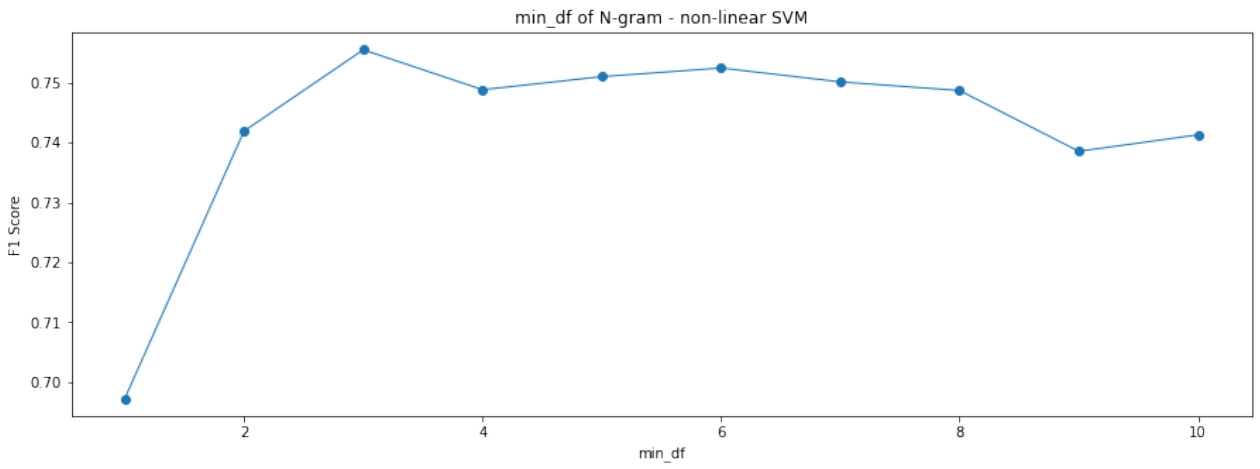
```
[0.660427807486631, 0.723114956736712, 0.7282809611829946, 0.7299448867115738,
0.7319461444308445, 0.7319461444308445, 0.7223587223587224,
0.7155172413793105, 0.7183271832718328, 0.715248009797918]
Max F1 Score = 0.7319461444308445
best threshold M = 5
```



```
#[22]
# i-3: Decide on an appropriate threshold M for N-gram Model

# 4.min_df of N-gram - non-linear SVM

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.svm import SVC
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt

f1_score_list = []
m_list = list(range(1,11))

for m in m_list:
    vectorizer = CountVectorizer(min_df=m,ngram_range=(1,2),binary=True)
    train_ng_vectors = vectorizer.fit_transform(train_corpus)
    train_ng_vectors = train_ng_vectors.toarray()
    test_corpus = test_pre
    test_ng_vectors = vectorizer.transform(test_corpus)
    test_ng_vectors = test_ng_vectors.toarray()

    SVM=SVC(C=1,kernel='rbf',gamma=0.1)
    SVM.fit(train_ng_vectors,y_train)
    predict_svm=SVM.predict(test_ng_vectors)

    f1 = f1_score(y_test,predict_svm)
    f1_score_list.append(f1)

print(f1_score_list)
max_f1 = max(f1_score_list)
print ('Max F1 Score =', max_f1)
print ('best threshold M =',f1_score_list.index(max_f1) + 1)

#best threshold M=3
#plot
a4 = m_list
```

```
b4 = f1_score_list
plt.figure(figsize=(15,5))
plt.plot(a4,b4,'o-',linewidth=1)
plt.xlabel("min_df")
plt.ylabel("F1 Score")
plt.title("min_df of N-gram - non-linear SVM")
plt.show()
```

```
[0.6972361809045227, 0.7419165196942975, 0.7555040556199305,
0.7488372093023256, 0.751015670342426, 0.7524637681159421, 0.7501442585112522,
0.7487001733102254, 0.7385507246376811, 0.7412993039443155]
Max F1 Score = 0.7555040556199305
best threshold M = 3
```



(j) Incorporating the additional columns

I append "keywords", "location" and "keywords & location" in "text" separately. According to (d) and (i), I choose to use bag of words for the text features. And the min_df for logistic regression and linear SVM is 1, for naive bayes is 3 and for non-linear SVM is 5.

1. keyword+text

```
import pandas as pd
import re

from nltk.tokenize import TweetTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

train=pd.read_csv("split_train.csv")
x_train=train.drop('target',axis=1)#dataframe with index
x_train=x_train.where(x_train.notnull(), '0')
x_train=x_train.values.tolist()
for i in range(5329):
    x_train[i][3]=str(x_train[i][1])+' '+x_train[i][3]
```

```
x_train=pd.DataFrame(x_train,columns=['id', 'keyword', 'location','text'])
y_train=train.target

test=pd.read_csv("dev.csv")
x_test=test.drop('target',axis=1)#dataframe with index
x_test=x_test.where(x_test.notnull(), '0')
x_test=x_test.values.tolist()
for i in range(2284):
    x_test[i][3]=str(x_test[i][1])+' '+x_test[i][3]
x_test=pd.DataFrame(x_test,columns=['id', 'keyword', 'location','text'])
y_test=test.target

tweettoken = TweetTokenizer(strip_handles=True, reduce_len=True)
lemmatizer=WordNetLemmatizer()

train_pre_new=[]
test_pre_new=[]
data=[]
def preprocess(text,dataclass):
    url = re.compile(r'https?://\S+|www\.\S+')#delete url
    text_new=url.sub(r'',text)
    text_new=re.sub('[^a-zA-Z]'," ",text_new)#delete punctuation
    text_new=text_new.lower()#lowercase
    text_rest=tweettoken.tokenize(text_new)
    for i in text_rest:
        if i in stopwords.words('english'):#delete stopwords
            text_rest.remove(i)
    rest=[]
    for k in text_rest:
        rest.append(lemmatizer.lemmatize(k))#lemmatize
    ret=" ".join(rest)
    if dataclass==1:
        train_pre_new.append(ret)
    elif dataclass==0:
        test_pre_new.append(ret)

def splitclass(data,q,m):
         for j in range(q):
                 preprocess(data["text"].iloc[j],m)

splitclass(x_train,5329,1)
splitclass(x_test,2284,0)

train_result = pd.DataFrame(train_pre_new)
train_result = train_result.join(y_train)
test_result = pd.DataFrame(test_pre_new)
test_result = test_result.join(y_test)

#train_result.to_csv('train_pre+key.csv', sep=',',header=True, index=False)
```

```python
#test_result.to_csv('test_pre+key.csv', sep=',',header=True, index=False)

import numpy as np
import matplotlib.pyplot as plt

train=pd.read_csv("train_pre+key.csv")
x_train=train.iloc[:, 0]
y_train=train.iloc[:, 1]

test=pd.read_csv("test_pre+key.csv")
x_test=test.iloc[:, 0]
y_test=test.iloc[:, 1]

# use bag of words model for text vectorization
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=5,binary=True) #set M=5 for svm
#vectorizer = CountVectorizer(min_df=5,binary=True) #set M=1 for lr and lsvm

train_corpus = x_train
print (train_corpus.head(5))

train_vectors = vectorizer.fit_transform(train_corpus) #transform each
document into a word frequency vector
print(type(train_vectors)) #type: sparse vector
print (train_vectors)

train_vectors = train_vectors.toarray()
print (train_vectors)

train_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
train_voca_dic = vectorizer.vocabulary_
print('vocabulary list of trainset:', train_voca_list)
print( 'vocabulary dic of trainset:', train_voca_dic)

print(type(train_vectors)) #type: np array
print(train_vectors.sum(axis=0)) #count each word' frequency in the corpus

X1 = train_vectors.shape
print (X1)
train_feature_num = X1[1] #vector length/number of features
print ('total number of features in trainset:', train_feature_num)


test_corpus = x_test
print (test_corpus.head(5))
```

```
test_vectors = vectorizer.transform(test_corpus) #transform each document into
a word frequency vector
print(type(test_vectors)) #type: sparse vector
print (test_vectors)

test_vectors = test_vectors.toarray()
print (test_vectors)

test_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
test_voca_dic = vectorizer.vocabulary_
print('vocabulary list of trainset:', test_voca_list)
print( 'vocabulary dic of trainset:', test_voca_dic)

print(type(test_vectors)) #type: np array
print(test_vectors.sum(axis=0)) #count each word' frequency in the corpus

X2 = test_vectors.shape
print (X2)
test_feature_num = X2[1] #vector length/number of features
print ('total number of features in devset:', test_feature_num)




# 1.f
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
#LR model
LR = LogisticRegression()
LR.fit(train_vectors,y_train)
predict_lr=LR.predict(test_vectors)
print(f1_score(y_test,predict_lr))
#coefficient
conf_mat = confusion_matrix(y_test, predict_lr)
print(conf_mat)
print(classification_report(y_test, predict_lr))
coef_lr=LR.coef_[0]
maxindex_lr = np.argmax(coef_lr )
minindex_lr = np.argmin(coef_lr )

for i in train_voca_dic.keys():
    if train_voca_dic[i]==maxindex_lr:
        print('is real',i)
    if train_voca_dic[i]==minindex_lr:
        print('not real',i)
#0.7521834061135372
#is real: hiroshima
```

```
#not real: bag

#1.g
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score

Cs=[0.01, 0.1, 1.0, 10.0, 100.0]
f1_scores_l=[]

for C in Cs:
    LSVM=LinearSVC(C=C,max_iter=100000)
    LSVM.fit(train_vectors,y_train)
    predict_lsvm=LSVM.predict(test_vectors)
    f1_scores_l.append(f1_score(y_test,predict_lsvm))
    #print(LSVM.score(test_vectors,y_test))
    print(f1_score(y_test,predict_lsvm))

## plot
fig1=plt.figure()
ax=fig1.add_subplot(1,1,1)
ax.plot(Cs,f1_scores_l)
ax.set_xlabel(r"C")
ax.set_ylabel(r"f1_scores")
ax.set_xscale('log')
ax.set_title("LinearSVC")
plt.show()
#Linear SVM model
lsvm_model=LinearSVC(C=0.1)
lsvm_model.fit(train_vectors,y_train)
predict_lsvm=lsvm_model.predict(test_vectors)
print(f1_score(y_test,predict_lsvm))

#coefficient
coef_lsvm=lsvm_model.coef_[0]
maxindex_lsvm = np.argmax(coef_lsvm )
minindex_lsvm = np.argmin(coef_lsvm )
for i in train_voca_dic.keys():
    if train_voca_dic[i]==maxindex_lr:
        print('is real',i)
    if train_voca_dic[i]==minindex_lr:
        print('not real',i)
#0.7494553376906319
#is real: hiroshima
#not real: bag

#1.h
from sklearn.svm import SVC
from sklearn.metrics import f1_score
```

```
G=[0.03,0.05,0.08,0.1,0.15]
for g in G:
    SVM=SVC(C=1,kernel='rbf',gamma=g)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    print(f1_score(y_test,predict_svm))

#best gamma=0.08
Cs_svm=[0.01, 0.1, 1.0, 10.0, 100.0]
f1_scores_nl=[]

for c in Cs_svm:
    SVM=SVC(C=c,kernel='rbf',gamma=0.08,max_iter=100000)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    f1_scores_nl.append(f1_score(y_test,predict_svm))
    print(f1_score(y_test,predict_svm))

#best C=1
## plot
fig2=plt.figure()
ax=fig2.add_subplot(1,1,1)
ax.plot(Cs_svm,f1_scores_nl)
ax.set_xlabel(r"C")
ax.set_ylabel(r"f1_scores")
ax.set_xscale('log')
ax.set_title("NoneLinearSVC")
plt.show()

#SVM model
SVM=SVC(C=1,kernel='rbf',gamma=0.08)
SVM.fit(train_vectors,y_train)
predict_svm=SVM.predict(test_vectors)
print(f1_score(y_test,predict_svm))
#0.7501442585112522
```

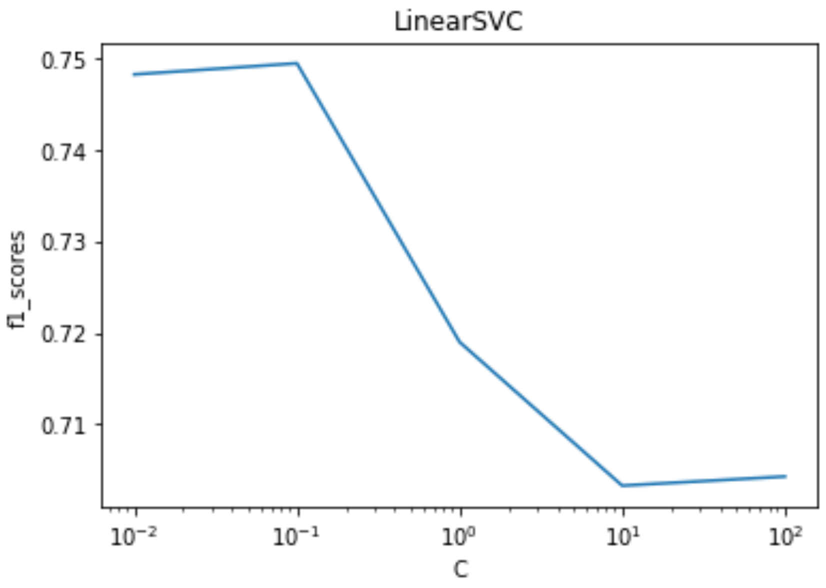
Results:

Logistic regression:

```
0.7521834061135372
[[1141  177]
 [ 277  689]]
              precision    recall  f1-score   support

           0       0.80      0.87      0.83      1318
           1       0.80      0.71      0.75       966

    accuracy                           0.80      2284
   macro avg       0.80      0.79      0.79      2284
weighted avg       0.80      0.80      0.80      2284

not real bag
is real hiroshima
```

Linear SVM:

```
0.7482435597189695
0.7494553376906319
0.7189819724284199
0.7032734952481521
0.7042698998418556
```



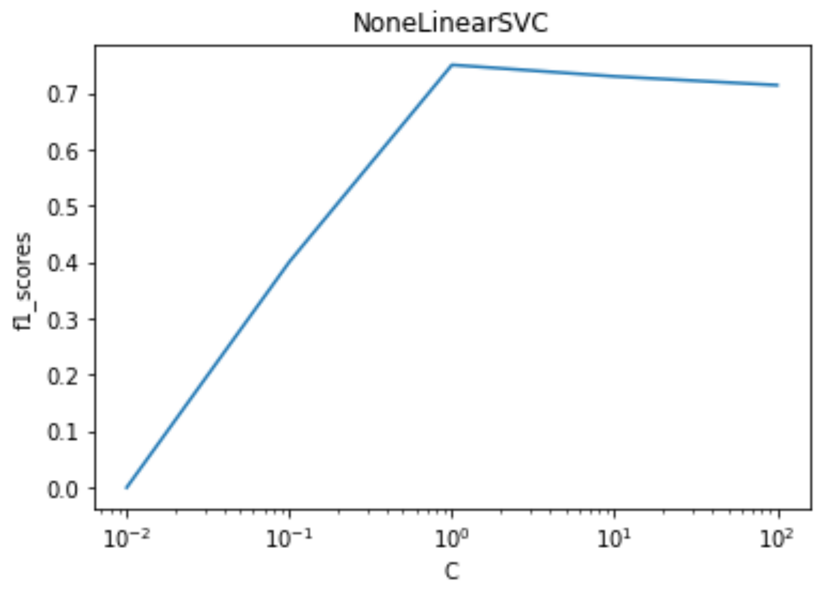
Best C=0.1

Non-linear SVM:

Gamma:

```
0.7245145631067961
0.7420494699646644
0.7501442585112522
0.7497116493656286
0.746987951807229
```

Best gamma=0.08



```
0.0
0.40032948929159795
0.7501442585112522
0.7297008547008548
0.7141372141372141
```

Best C=1

Most important words: bag, hiroshima

F1-scores are worse than those without the column "keywords". Because "keywords" is the theme of the text, but always has nothing to do with the disaster.

2. location+text

```
import pandas as pd
import re

from nltk.tokenize import TweetTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

train=pd.read_csv("split_train.csv")
x_train=train.drop('target',axis=1)#dataframe with index
x_train=x_train.where(x_train.notnull(), '0')
x_train=x_train.values.tolist()
for i in range(5329):
    x_train[i][3]=str(x_train[i][2])+' '+x_train[i][3]
x_train=pd.DataFrame(x_train,columns=['id', 'keyword', 'location','text'])
y_train=train.target

test=pd.read_csv("dev.csv")
x_test=test.drop('target',axis=1)#dataframe with index
x_test=x_test.where(x_test.notnull(), '0')
```

```
x_test=x_test.values.tolist()
for i in range(2284):
    x_test[i][3]=str(x_test[i][2])+' '+x_test[i][3]
x_test=pd.DataFrame(x_test,columns=['id', 'keyword', 'location','text'])
y_test=test.target

tweettoken = TweetTokenizer(strip_handles=True, reduce_len=True)
lemmatizer=WordNetLemmatizer()

train_pre_new=[]
test_pre_new=[]
data=[]
def preprocess(text,dataclass):
    url = re.compile(r'https?://\S+|www\.\S+')#delete url
    text_new=url.sub(r'',text)
    text_new=re.sub('[^a-zA-Z]'," ",text_new)#delete punctuation
    text_new=text_new.lower()#lowercase
    text_rest=tweettoken.tokenize(text_new)
    for i in text_rest:
        if i in stopwords.words('english'):#delete stopwords
            text_rest.remove(i)
    rest=[]
    for k in text_rest:
        rest.append(lemmatizer.lemmatize(k))#lemmatize
    ret=" ".join(rest)
    if dataclass==1:
        train_pre_new.append(ret)
    elif dataclass==0:
        test_pre_new.append(ret)

def splitclass(data,q,m):
         for j in range(q):
                preprocess(data["text"].iloc[j],m)

splitclass(x_train,5329,1)
splitclass(x_test,2284,0)


train_result = pd.DataFrame(train_pre_new)
train_result = train_result.join(y_train)
test_result = pd.DataFrame(test_pre_new)
test_result = test_result.join(y_test)

#train_result.to_csv('train_pre+loc.csv', sep=',',header=True, index=False)
#test_result.to_csv('test_pre+loc.csv', sep=',',header=True, index=False)

import numpy as np
import matplotlib.pyplot as plt
```

```python
train=pd.read_csv("train_pre+loc.csv")
x_train=train.iloc[:, 0]
y_train=train.iloc[:, 1]

test=pd.read_csv("test_pre+loc.csv")
x_test=test.iloc[:, 0]
y_test=test.iloc[:, 1]

# use bag of words model for text vectorization
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=5,binary=True) #set M=5 for svm
#vectorizer = CountVectorizer(min_df=5,binary=True) #set M=1 for lr and lsvm

train_corpus = x_train
print (train_corpus.head(5))

train_vectors = vectorizer.fit_transform(train_corpus) #transform each
document into a word frequency vector
print(type(train_vectors)) #type: sparse vector
print (train_vectors)

train_vectors = train_vectors.toarray()
print (train_vectors)

train_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
train_voca_dic = vectorizer.vocabulary_
print('vocabulary list of trainset:', train_voca_list)
print( 'vocabulary dic of trainset:', train_voca_dic)

print(type(train_vectors)) #type: np array
print(train_vectors.sum(axis=0)) #count each word' frequency in the corpus

X1 = train_vectors.shape
print (X1)
train_feature_num = X1[1] #vector length/number of features
print ('total number of features in trainset:', train_feature_num)

test_corpus = x_test
print (test_corpus.head(5))

test_vectors = vectorizer.transform(test_corpus) #transform each document into
a word frequency vector
print(type(test_vectors)) #type: sparse vector
print (test_vectors)

test_vectors = test_vectors.toarray()
print (test_vectors)
```

```
test_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
test_voca_dic = vectorizer.vocabulary_
print('vocabulary list of trainset:', test_voca_list)
print( 'vocabulary dic of trainset:', test_voca_dic)

print(type(test_vectors)) #type: np array
print(test_vectors.sum(axis=0)) #count each word' frequency in the corpus

X2 = test_vectors.shape
print (X2)
test_feature_num = X2[1] #vector length/number of features
print ('total number of features in devset:', test_feature_num)

# 1.f

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
#LR model
LR = LogisticRegression()
LR.fit(train_vectors,y_train)
predict_lr=LR.predict(test_vectors)
print(f1_score(y_test,predict_lr))
#coefficient
conf_mat = confusion_matrix(y_test, predict_lr)
print(conf_mat)
print(classification_report(y_test, predict_lr))
coef_lr=LR.coef_[0]
maxindex_lr = np.argmax(coef_lr )
minindex_lr = np.argmin(coef_lr )

for i in train_voca_dic.keys():
    if train_voca_dic[i]==maxindex_lr:
        print('is real',i)
    if train_voca_dic[i]==minindex_lr:
        print('not real',i)
#0.7483660130718954
#is real: hiroshima
#not real: full

#1.g
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score

Cs=[0.01, 0.1, 1.0, 10.0, 100.0]
f1_scores_l=[]
```

```
for C in Cs:
    LSVM=LinearSVC(C=C,max_iter=100000)
    LSVM.fit(train_vectors,y_train)
    predict_lsvm=LSVM.predict(test_vectors)
    f1_scores_l.append(f1_score(y_test,predict_lsvm))
    #print(LSVM.score(test_vectors,y_test))
    print(f1_score(y_test,predict_lsvm))

## plot
fig1=plt.figure()
ax=fig1.add_subplot(1,1,1)
ax.plot(Cs,f1_scores_l)
ax.set_xlabel(r"C")
ax.set_ylabel(r"f1_scores")
ax.set_xscale('log')
ax.set_title("LinearSVC")
plt.show()
#Linear SVM model
lsvm_model=LinearSVC(C=0.1)
lsvm_model.fit(train_vectors,y_train)
predict_lsvm=lsvm_model.predict(test_vectors)
print(f1_score(y_test,predict_lsvm))

#coefficient
coef_lsvm=lsvm_model.coef_[0]
maxindex_lsvm = np.argmax(coef_lsvm )
minindex_lsvm = np.argmin(coef_lsvm )
for i in train_voca_dic.keys():
    if train_voca_dic[i]==maxindex_lr:
        print('is real',i)
    if train_voca_dic[i]==minindex_lr:
        print('not real',i)
#0.7504078303425775
#is real: hiroshima
#not real: full

#1.h
from sklearn.svm import SVC
from sklearn.metrics import f1_score

G=[0.01,0.1,1]
for g in G:
    SVM=SVC(C=1,kernel='rbf',gamma=g)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    print(f1_score(y_test,predict_svm))

G=[0.05,0.08,0.12,0.15,0.17]
```

```
for g in G:
    SVM=SVC(C=1,kernel='rbf',gamma=g)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    print(f1_score(y_test,predict_svm))

#best gamma=0.1
#best C=1

Cs_svm=[0.01, 0.1, 1.0, 10.0, 100.0]
f1_scores_nl=[]

for c in Cs_svm:
    SVM=SVC(C=c,kernel='rbf',gamma=0.1,max_iter=100000)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    f1_scores_nl.append(f1_score(y_test,predict_svm))
    print(f1_score(y_test,predict_svm))

#best C=1
## plot
fig2=plt.figure()
ax=fig2.add_subplot(1,1,1)
ax.plot(Cs_svm,f1_scores_nl)
ax.set_xlabel(r"C")
ax.set_ylabel(r"f1_scores")
ax.set_xscale('log')
ax.set_title("NoneLinearSVC")
plt.show()

#SVM model
SVM=SVC(C=1,kernel='rbf',gamma=0.1)
SVM.fit(train_vectors,y_train)
predict_svm=SVM.predict(test_vectors)
print(f1_score(y_test,predict_svm))
#0.7501442585112522
```

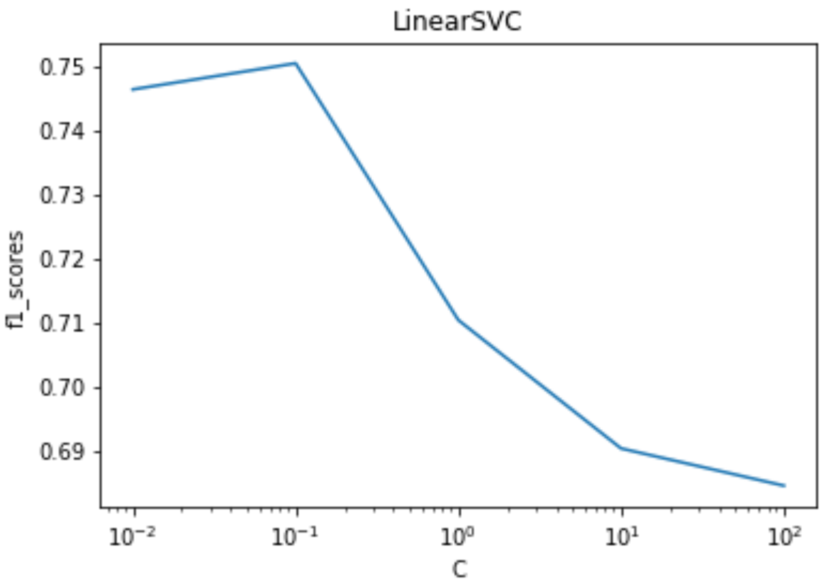
Results:

Logistic Regression:

```
0.7483660130718954
[[1135  183]
 [ 279  687]]
              precision    recall  f1-score   support

           0       0.80      0.86      0.83      1318
           1       0.79      0.71      0.75       966

    accuracy                           0.80      2284
   macro avg       0.80      0.79      0.79      2284
weighted avg       0.80      0.80      0.80      2284

not real full
is real hiroshima
```

Linear SVM:

```
0.7463386057410663
0.7504078303425775
0.7103594080338266
0.6903394255874673
0.6845360824742269
```



Best C=0.1

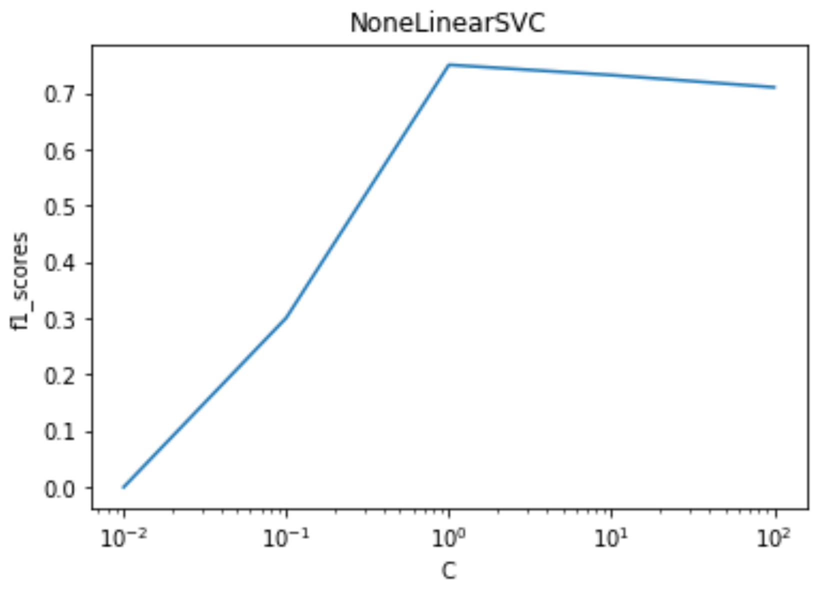
Non-linear SVM:

Gamma:

```
0.6443402545210984
0.7501442585112522
0.27272727272727276
```

```
0.7447058823529412
0.7484094852515905
0.7474048442906576
0.7463893703061814
0.745920745920746
```

Best gamma=0.1



```
0.0
0.30079155672823216
0.7501442585112522
0.7319148936170212
0.7102212855637513
```

Best C=1

Most important words: full, hiroshima

F1-scores are worse than those without the column "location". Because this column has a large variation and always has nothing to do with the disaster.

3. keyword+location+text

```python
import pandas as pd
import re

from nltk.tokenize import TweetTokenizer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

train=pd.read_csv("split_train.csv")
x_train=train.drop('target',axis=1)#dataframe with index
x_train=x_train.where(x_train.notnull(), '0')
x_train=x_train.values.tolist()
for i in range(5329):
    x_train[i][3]=str(x_train[i][1])+' '+str(x_train[i][2])+' '+x_train[i][3]
x_train=pd.DataFrame(x_train,columns=['id', 'keyword', 'location','text'])
y_train=train.target

test=pd.read_csv("dev.csv")
x_test=test.drop('target',axis=1)#dataframe with index
```

```
x_test=x_test.where(x_test.notnull(), '0')
x_test=x_test.values.tolist()
for i in range(2284):
    x_test[i][3]=str(x_test[i][1])+' '+str(x_test[i][2])+' '+x_test[i][3]
x_test=pd.DataFrame(x_test,columns=['id', 'keyword', 'location','text'])
y_test=test.target

tweettoken = TweetTokenizer(strip_handles=True, reduce_len=True)
lemmatizer=WordNetLemmatizer()

train_pre_new=[]
test_pre_new=[]
data=[]
def preprocess(text,dataclass):
    url = re.compile(r'https?://\S+|www\.\S+')#delete url
    text_new=url.sub(r'',text)
    text_new=re.sub('[^a-zA-Z]'," ",text_new)#delete punctuation
    text_new=text_new.lower()#lowercase
    text_rest=tweettoken.tokenize(text_new)
    for i in text_rest:
        if i in stopwords.words('english'):#delete stopwords
            text_rest.remove(i)
    rest=[]
    for k in text_rest:
        rest.append(lemmatizer.lemmatize(k))#lemmatize
    ret=" ".join(rest)
    if dataclass==1:
        train_pre_new.append(ret)
    elif dataclass==0:
        test_pre_new.append(ret)

def splitclass(data,q,m):
     for j in range(q):
            preprocess(data["text"].iloc[j],m)

splitclass(x_train,5329,1)
splitclass(x_test,2284,0)


train_result = pd.DataFrame(train_pre_new)
train_result = train_result.join(y_train)
test_result = pd.DataFrame(test_pre_new)
test_result = test_result.join(y_test)

train_result.to_csv('train_pre+key+loc.csv', sep=',',header=True, index=False)
test_result.to_csv('test_pre+key+loc.csv', sep=',',header=True, index=False)


import numpy as np
import matplotlib.pyplot as plt

train=pd.read_csv("train_pre+key+loc.csv")
x_train=train.iloc[:, 0]
y_train=train.iloc[:, 1]

train=pd.read_csv("test_pre+key+loc.csv")
x_test=test.iloc[:, 0]
y_test=test.iloc[:, 1]




# use bag of words model for text vectorization
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=5,binary=True) #set M=5 for svm
#vectorizer = CountVectorizer(min_df=5,binary=True) #set M=1 for lr and
lsvm,binary=True) #instantiate the CountVectorizer class, set M=2

train_corpus = x_train
print (train_corpus.head(5))

train_vectors = vectorizer.fit_transform(train_corpus) #transform each
document into a word frequency vector
print(type(train_vectors)) #type: sparse vector
print (train_vectors)

train_vectors = train_vectors.toarray()
print (train_vectors)

train_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
train_voca_dic = vectorizer.vocabulary_
print('vocabulary list of trainset:', train_voca_list)
print( 'vocabulary dic of trainset:', train_voca_dic)

print(type(train_vectors)) #type: np array
print(train_vectors.sum(axis=0)) #count each word' frequency in the corpus

X1 = train_vectors.shape
print (X1)
train_feature_num = X1[1] #vector length/number of features
print ('total number of features in trainset:', train_feature_num)




test_corpus = x_test
print (test_corpus.head(5))
```

```
test_vectors = vectorizer.transform(test_corpus) #transform each document into
a word frequency vector
print(type(test_vectors)) #type: sparse vector
print (test_vectors)

test_vectors = test_vectors.toarray()
print (test_vectors)

test_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
test_voca_dic = vectorizer.vocabulary_
print('vocabulary list of trainset:', test_voca_list)
print( 'vocabulary dic of trainset:', test_voca_dic)

print(type(test_vectors)) #type: np array
print(test_vectors.sum(axis=0)) #count each word' frequency in the corpus

X2 = test_vectors.shape
print (X2)
test_feature_num = X2[1] #vector length/number of features
print ('total number of features in devset:', test_feature_num)



# 1.f
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
#LR model
LR = LogisticRegression()
LR.fit(train_vectors,y_train)
predict_lr=LR.predict(test_vectors)
print(f1_score(y_test,predict_lr))
#coefficient
conf_mat = confusion_matrix(y_test, predict_lr)
print(conf_mat)
print(classification_report(y_test, predict_lr))
coef_lr=LR.coef_[0]
maxindex_lr = np.argmax(coef_lr )
minindex_lr = np.argmin(coef_lr )

for i in train_voca_dic.keys():
    if train_voca_dic[i]==maxindex_lr:
        print('is real',i)
    if train_voca_dic[i]==minindex_lr:
        print('not real',i)
#0.7464940668824164
#is real: hiroshima
```

```python
#not real: full

#1.g
from sklearn.svm import LinearSVC
from sklearn.metrics import f1_score

Cs=[0.01, 0.1, 1.0, 10.0, 100.0]
f1_scores_l=[]

for C in Cs:
    LSVM=LinearSVC(C=C,max_iter=100000)
    LSVM.fit(train_vectors,y_train)
    predict_lsvm=LSVM.predict(test_vectors)
    f1_scores_l.append(f1_score(y_test,predict_lsvm))
    #print(LSVM.score(test_vectors,y_test))
    print(f1_score(y_test,predict_lsvm))

## plot
fig1=plt.figure()
ax=fig1.add_subplot(1,1,1)
ax.plot(Cs,f1_scores_l)
ax.set_xlabel(r"C")
ax.set_ylabel(r"f1_scores")
ax.set_xscale('log')
ax.set_title("LinearSVC")
plt.show()
#Linear SVM model
lsvm_model=LinearSVC(C=0.01)
lsvm_model.fit(train_vectors,y_train)
predict_lsvm=lsvm_model.predict(test_vectors)
print(f1_score(y_test,predict_lsvm))

#coefficient
coef_lsvm=lsvm_model.coef_[0]
maxindex_lsvm = np.argmax(coef_lsvm )
minindex_lsvm = np.argmin(coef_lsvm )
for i in train_voca_dic.keys():
    if train_voca_dic[i]==maxindex_lr:
        print('is real',i)
    if train_voca_dic[i]==minindex_lr:
        print('not real',i)
#0.7447795823665895
#is real: hiroshima
#not real: full

#1.h
from sklearn.svm import SVC
from sklearn.metrics import f1_score
```

```
G=[0.01,0.1,1,5,10]
for g in G:
    SVM=SVC(C=1,kernel='rbf',gamma=g)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    print(f1_score(y_test,predict_svm))

G=[0.05,0.08,0.11,0.14,0.17]
for g in G:
    SVM=SVC(C=1,kernel='rbf',gamma=g)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    print(f1_score(y_test,predict_svm))

#best gamma=0.11
#best C=1

Cs_svm=[0.01, 0.1, 1.0, 10.0, 100.0]
f1_scores_nl=[]

for c in Cs_svm:
    SVM=SVC(C=c,kernel='rbf',gamma=0.11,max_iter=100000)
    SVM.fit(train_vectors,y_train)
    predict_svm=SVM.predict(test_vectors)
    f1_scores_nl.append(f1_score(y_test,predict_svm))
    print(f1_score(y_test,predict_svm))

## plot
fig2=plt.figure()
ax=fig2.add_subplot(1,1,1)
ax.plot(Cs_svm,f1_scores_nl)
ax.set_xlabel(r"C")
ax.set_ylabel(r"f1_scores")
ax.set_xscale('log')
ax.set_title("NoneLinearSVC")
plt.show()

#SVM model
SVM=SVC(C=1,kernel='rbf',gamma=0.11)
SVM.fit(train_vectors,y_train)
predict_svm=SVM.predict(test_vectors)
print(f1_score(y_test,predict_svm))

#0.7435897435897435
```

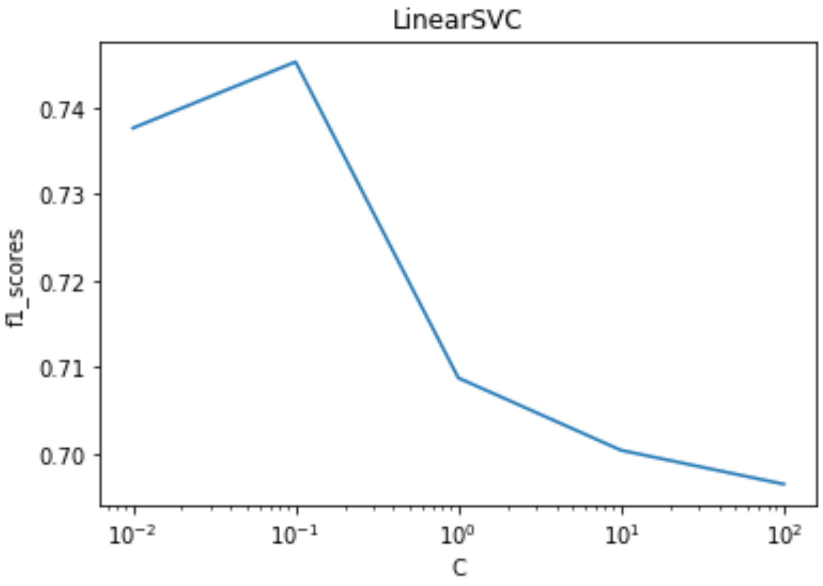
Result:

Logistic Regression:

```
0.7471201316511245
[[1142  176]
 [ 285  681]]
              precision    recall  f1-score   support

           0       0.80      0.87      0.83      1318
           1       0.79      0.70      0.75       966

    accuracy                           0.80      2284
   macro avg       0.80      0.79      0.79      2284
weighted avg       0.80      0.80      0.80      2284

not real full
is real hiroshima
```

Linear SVM:

```
0.7447795823665895
0.7430744160782183
0.7016806722689076
0.6911917098445596
0.6859205776173285
```



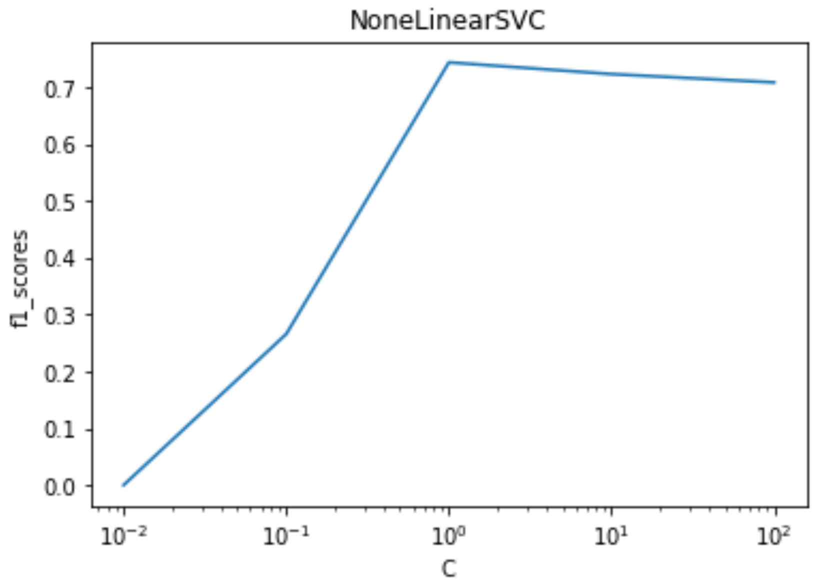
Best C=0.1

Non-linear SVM:

Gamma:

```
0.639300134589502
0.743455497382199
0.2714535901926445
0.20437956204379562
0.20437956204379562
0.7417061611374408
0.7430232558139535
0.7435897435897435
0.7425569176882663
0.7396937573616019
```

Best gamma=0.11



```
0.0
0.2657091561938959
0.7435897435897435
0.7229693383539537
0.7083333333333333
```

Best C=1

Most important words: full, hiroshima

F1-scores are the worst among all the combinations. Because the two columns have a large variation and do not have a strong relationship with the disaster, this combination has two bad result, which would be worse than the previous one.

(k) Finalizing your model

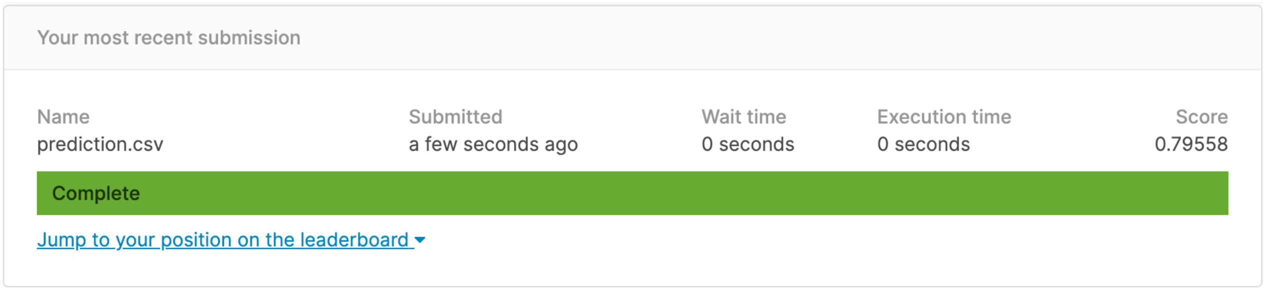
According to (d) and (i), Naive Bayes model has the best F1-score. So we choose this model to do the final prediction.

```
import pandas as pd
import re

from nltk.tokenize import TweetTokenizer
```

```python
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

train=pd.read_csv("train.csv")
x_train=train.drop('target',axis=1)#dataframe with index
y_train=train.target

x_test=pd.read_csv("test.csv")

tweettoken = TweetTokenizer(strip_handles=True, reduce_len=True)
lemmatizer=WordNetLemmatizer()

train_pre=[]
test_pre=[]
data=[]
def preprocess(text,dataclass):
    url = re.compile(r'https?://\S+|www\.\S+')#delete url
    text_new=url.sub(r'',text)
    text_new=re.sub('[^a-zA-Z]'," ",text_new)#delete punctuation
    text_new=text_new.lower()#lowercase
    text_rest=tweettoken.tokenize(text_new)
    for i in text_rest:
        if i in stopwords.words('english'):#delete stopwords
            text_rest.remove(i)
    rest=[]
    for k in text_rest:
        rest.append(lemmatizer.lemmatize(k))#lemmatize
    ret=" ".join(rest)
    if dataclass==1:
        train_pre.append(ret)
    elif dataclass==0:
        test_pre.append(ret)

def splitclass(data,q,m):
         for j in range(q):
                preprocess(data["text"].iloc[j],m)

splitclass(x_train,7613,1)
splitclass(x_test,3263,0)

train_result = pd.DataFrame(train_pre)
train_result = train_result.join(y_train)
test_result = pd.DataFrame(test_pre)

train_result.to_csv('train_pre_total.csv', sep=',',header=True, index=False)
test_result.to_csv('test_pre_total.csv', sep=',',header=True, index=False)

#train=pd.read_csv("train_pre_total.csv")
x_train=train_result.iloc[:, 0]
```

```
y_train=train_result.iloc[:, 1]
x_test=test_result.iloc[:,0]

# use bag of words model for text vectorization
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=3,binary=True) #instantiate the
CountVectorizer class, set M=3

train_corpus = x_train
print (train_corpus.head(5))

train_vectors = vectorizer.fit_transform(train_corpus) #transform each
document into a word frequency vector
print(type(train_vectors)) #type: sparse vector
print (train_vectors)

train_vectors = train_vectors.toarray()
print (train_vectors)

train_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
train_voca_dic = vectorizer.vocabulary_
print('vocabulary list of trainset:', train_voca_list)
print( 'vocabulary dic of trainset:', train_voca_dic)

print(type(train_vectors)) #type: np array
print(train_vectors.sum(axis=0)) #count each word' frequency in the corpus

X1 = train_vectors.shape
print (X1)
train_feature_num = X1[1] #vector length/number of features
print ('total number of features in trainset:', train_feature_num)

test_corpus = x_test
print (test_corpus.head(5))

test_vectors = vectorizer.transform(test_corpus) #transform each document into
a word frequency vector
print(type(test_vectors)) #type: sparse vector
print (test_vectors)

test_vectors = test_vectors.toarray()
print (test_vectors)

test_voca_list = vectorizer.get_feature_names() #generate corpus into a
vocabulary list
test_voca_dic = vectorizer.vocabulary_
print('vocabulary list of trainset:', test_voca_list)
print( 'vocabulary dic of trainset:', test_voca_dic)

print(type(test_vectors)) #type: np array
print(test_vectors.sum(axis=0)) #count each word' frequency in the corpus

X2 = test_vectors.shape
print (X2)
test_feature_num = X2[1] #vector length/number of features
print ('total number of features in devset:', test_feature_num)

# handwrite a BernoulliNB model
import numpy as np

class Bernoulli_NaiveBayes:

    def __init__(self):
        self.alpha = 1 # set smoothing factor=1(Laplace Smoothing), to avoid
zero probability problems

    def _cal_prior_prob_log(self, y, classes): # calculate the logarithm of
prior probability of each class, P(y=c_k)
        self.classes = np.unique(y)
        class_num = len(self.classes) #count the number of possible types of y
        sample_num = len(y)

        c_num = np.count_nonzero(y == classes[:, None], axis=1) #count sample
amount of each class
        prior_prob = (c_num + self.alpha) / (sample_num + class_num *
self.alpha) #calculate prior probabilities(add smoothing correction)
        prior_prob_log = np.log(prior_prob) #calculate logarithm

        return prior_prob_log

    def _cal_condi_prob_log(self, X, y, classes): #calculate the logarithm of
all conditional probabilities P(x^(j)|y=c_k)

        n = (X.shape)[1]
        K = len(classes)

        #create an empty multidimensional array
        #prob_log: logarithmic matrix of two conditional probabilities
        condi_prob_log = np.empty((2, K, n))

        for k, c in enumerate(classes):
            X_c = X[np.equal(y, c)] #acquire all samples of class c_k
            total_num = len(X_c)
            num_f1 = np.count_nonzero(X_c, axis=0) #count the number of
samples of which feature value is 1
```

```
            condi_prob_f1 = (num_f1 + self.alpha) / (total_num + self.alpha * 
2) #calculate conditional probability P(x^(j)=1|y=c_k)

            #calculate and store logarithm into matrix
            #prob_log[0]: store all values of log(P(x^(j)=0|y=c_k))
            #prob_log[1]: store all values of log(P(x^(j)=1|y=c_k))
            condi_prob_log[0, k] = np.log(1 - condi_prob_f1)
            condi_prob_log[1, k] = np.log(condi_prob_f1)

        return condi_prob_log

    def train(self, x_train, y_train): #train the model
        self.classes = np.unique(y_train) #acquire all classes
        self.prior_prob_log = self._cal_prior_prob_log(y_train, self.classes) 
#calculate and store the logarithm of all prior probabilities
        self.condi_prob_log = self._cal_condi_prob_log(x_train, y_train, 
self.classes) #calculate and store the logarithm of all conditional 
probabilities

    def _predict_single_sample(self, x): #predict the label of single sample

        K = len(self.classes)
        po_prob_log = np.empty(K) #create an empty multidimensional array

        index_f1 = x == 1 #acquire index of feature value=1
        index_f0 = ~index_f1 #acquire index of feature value=0

        for k in range(K): #iterate each class
            #calculate the logarithm of the numerator of the posterior 
probability
            po_prob_log[k] = self.prior_prob_log[k]
      + np.sum(self.condi_prob_log[0, k][index_f0])
      + np.sum(self.condi_prob_log[1, k][index_f1])

        label = np.argmax(po_prob_log) #get the class with the highest 
posterior probability
        return label

    def predict(self, X): #predict samples (include single sample)

        if X.ndim == 1: #if only predict single sample (the dimension of the 
array = 1), invoke _predict_single_sample()
            return self._predict_single_sample(X)
        else:
            #if predict multiple samples, loop call _predict_single_sample() 
and return a list of the predicted results
            labels = []
            for j in range(X.shape[0]):
                label = self._predict_single_sample(X[j])
                labels.append(label)
            return labels

#use Bernoulli_NaiveBayes to predict
x_train = train_vectors
y_train = np.array(y_train)
x_test = test_vectors

BernoulliNB = Bernoulli_NaiveBayes()
BernoulliNB.train(x_train,y_train)
y_pred = BernoulliNB.predict(x_test)
result=pd.DataFrame(y_pred,columns=['target'])
x_test=pd.read_csv("test.csv")
test_id=x_test.iloc[:,0]
test_id=test_id.to_frame()
result=test_id.join(result)
result.to_csv('prediction.csv', sep=',',header=True, index=False)
```



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
prediction.csv	a few seconds ago	0 seconds	0 seconds	0.79558

Complete

Jump to your position on the leaderboard

The result is 0.79558, which is higher than 0.76115 in (d). Because the total training dataset is bigger than what we use in (d), this improvement of performance is reasonable.

(l) Reflecting on interpretability

Description: Suppose that you were constructing a model for this task as part of a consulting job, where you not only cared about classification performance, but wanted an interpetable model, such that you could explain to your clients how it made its decisions, and they could trust the model.

l-1: Would you still choose the same approach, or a different one? Discuss why or why not, which approach you think would be best in this setting.

Answer:

Theoretically speaking, among the four models, Bernoulli Naive Bayes model has the best interpretability, Logistic Regression model has the second-best interpretability, Linear SVM model has the third-best interpretability, and Non-linear SVM model has the worst interpretability.

First, Naive Bayes and logistic regression are both linear models, in which feature weights correspond directly to the importance of features, so it is easy to understand what the model has learned. Both models are based on conditional probabilities, so they both have good interpretability for the final results of different classes.

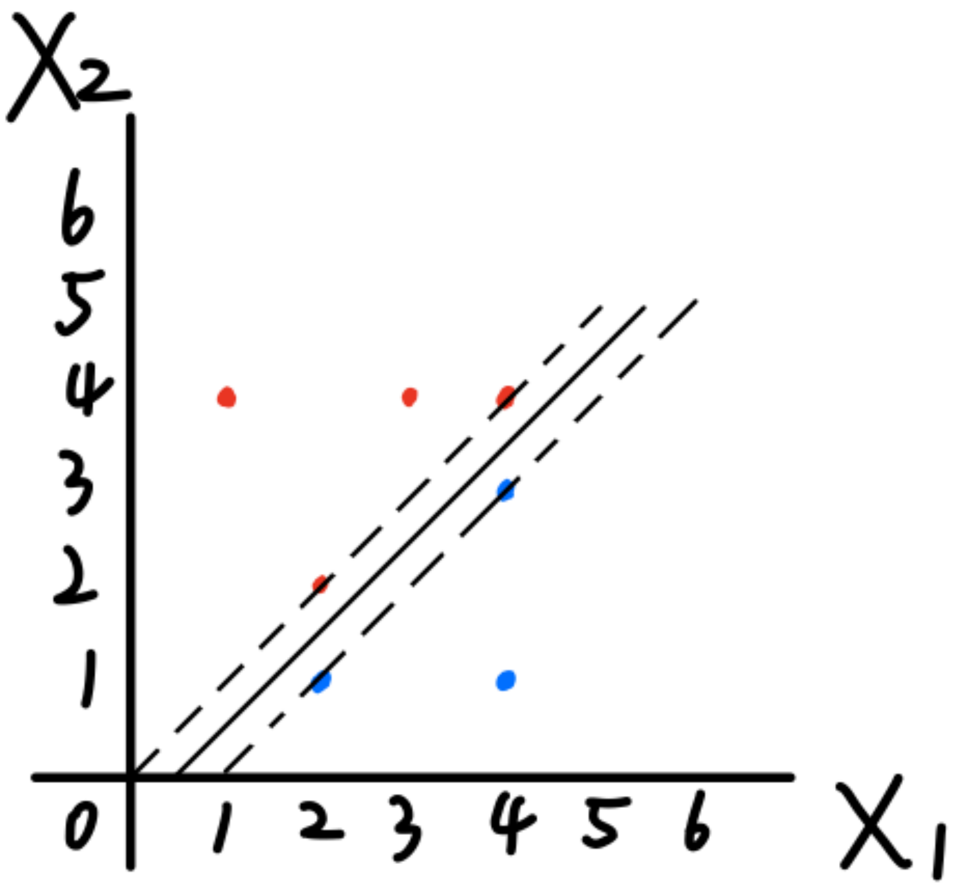
Second, Naive Bayes model is derived from the total probability formula, and each step has the probability deduction. It realizes classification by calculating the probability of features. Because constraints of Naive Bayes are stricter than logistic regression, the relevant parameters of Naive Bayes are more clear (i.e. have fixed forms). It directly counts the logical occurrence ratio of each feature as a weight. However, the parameters obtained by logistic regression do not have clear forms (because it does not have such strict constraints as Naive Bayes). It can obtain the weight of each feature through optimization methods such as gradient descent method to see which features are more important.

Third, SVM model has a theoretical derivation process, but its interpretability is not very strong. It's not as intuitive as Naive Bayes, logistic regression and decision tree. Relatively speaking, if using a linear kernel, its interpretability is slightly better, similar to linear regression; if using a nonlinear kernel, such as RBF, its interpretability is very poor.

In conclusion, interpretability is an essential characteristic of a reliable system and an important influence factor in clients' trust in the model/technique. **Given that the Bernoulli Naive Bayes model has not only the best performance but also the best interpretability among the four models used for this task, I would choose Bernoulli Naive Bayes model as our best approach.**

Written Problem 1

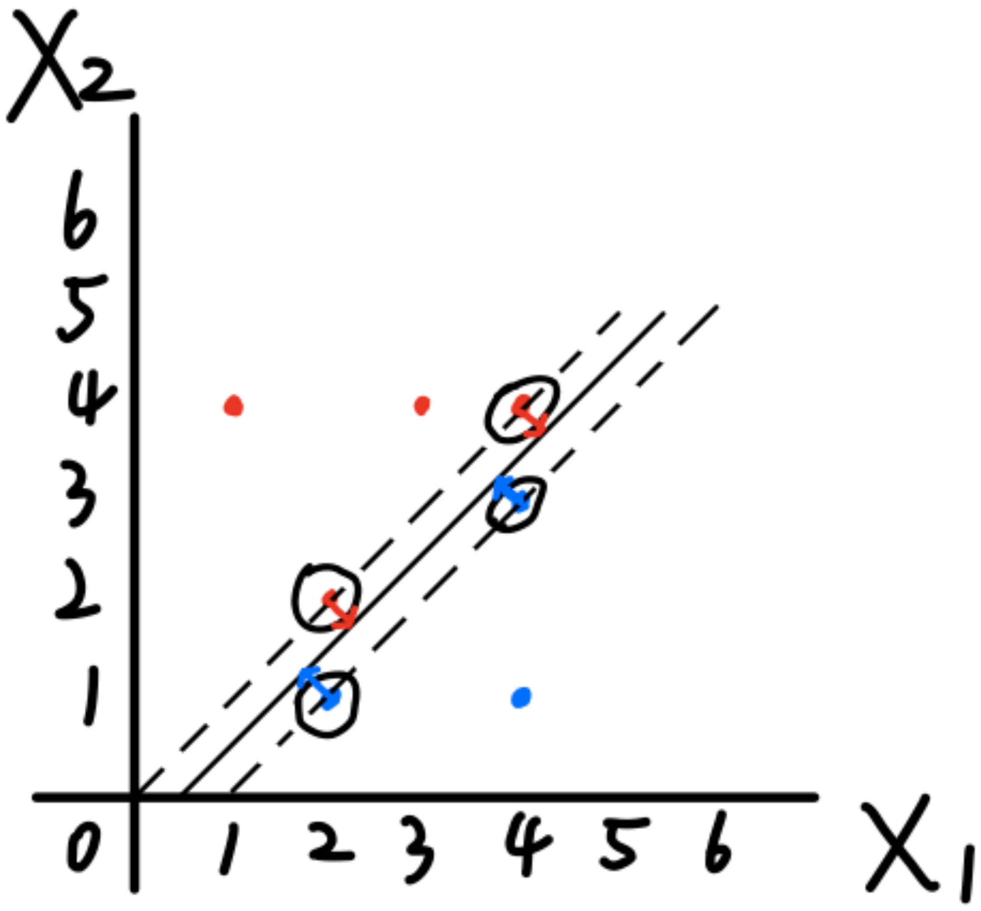
(a)



(b)

Classify to Red if $\beta_0 = -0.5, \beta_1 = 1, \beta_2 = -1$, and classify to Blue otherwise.

(c)

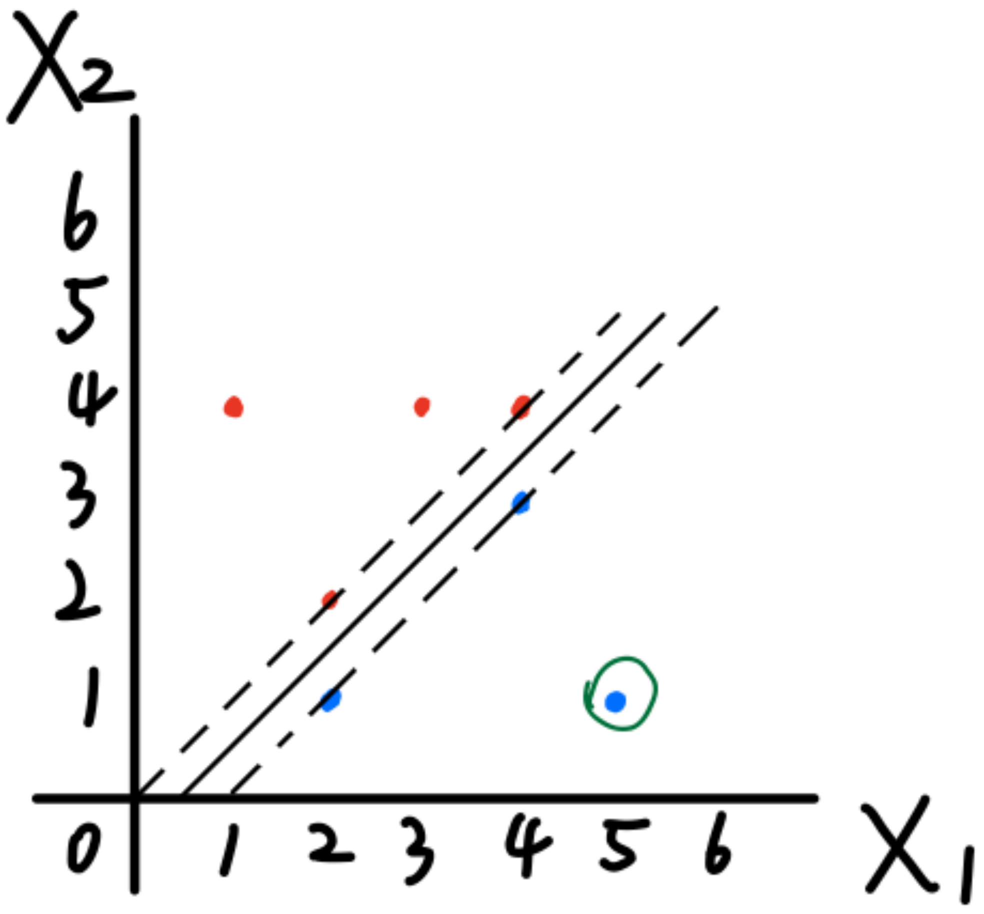


The margin are highlighted in black. And they are Red(2, 2), Red(4, 4), Blue(2, 1) and Blue(4, 3) (in the form of $Y(X_1, X_2)$)

(d)

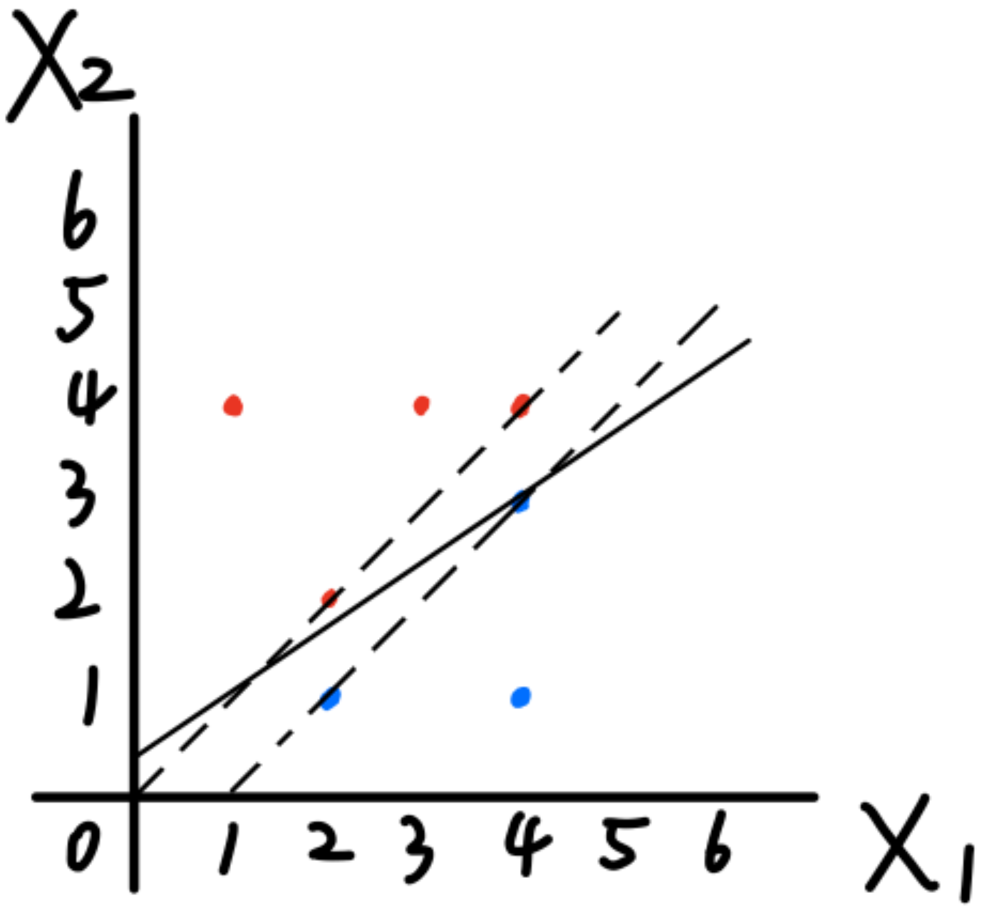
For Red observations, the support vectors are indicated as the red arrows (1, -1), for Blue observations, the support vectors are indicated as the blue arrows (-1, 1)

(e)



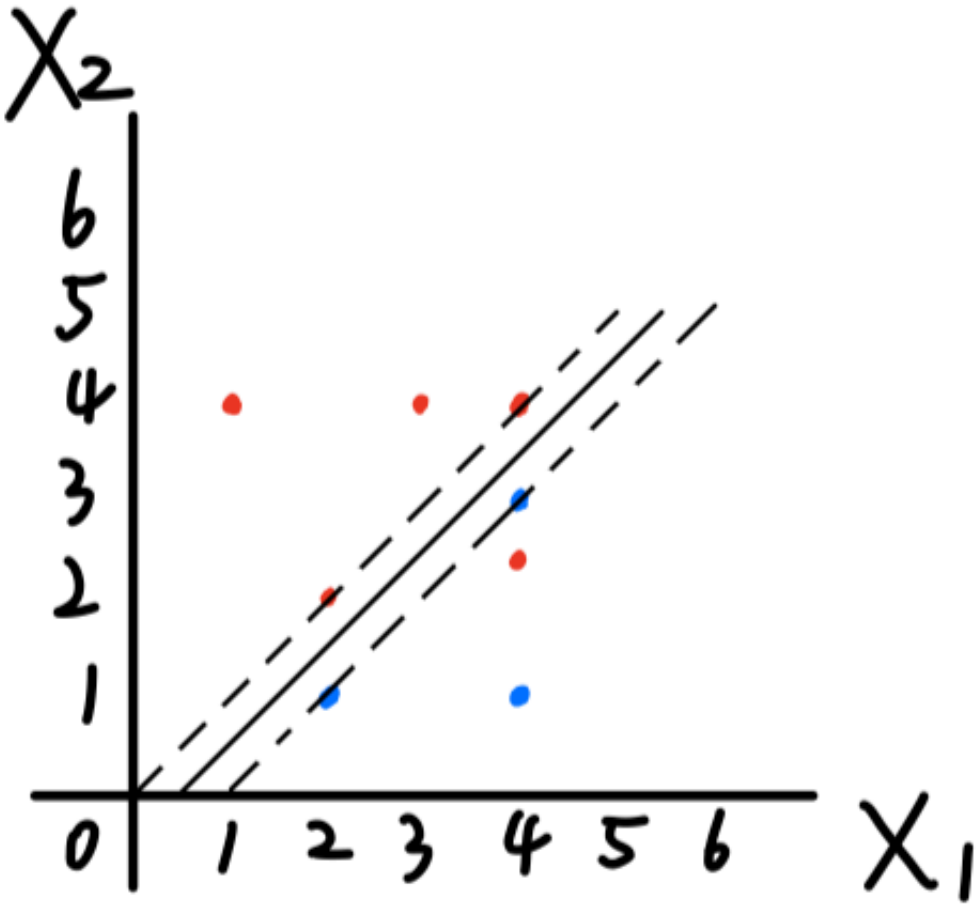
Move the seventh observation from (4, 1) to (5, 1) and the new observation is highlighted in green, the maximal margin hyperplane is the same as before, which is shown in the sketch.

(f)



The new hyperplane is shown in the sketch. The equation for this hyperplane is $0.5 + \frac{2}{3}X_1 - X_2$

(g)



The new observation is Red(4, 2.5) and the two classes are no longer separable by a hyperplane.

Written Problem 2

(a)

set events:

- $A = cough : 0/1$
- $B = sneeze : 0/1$
- $C = flu : 0/1$

from the text we know:

- $P(C = 0) = 0.8$
- $P(C = 1) = 0.2$
- $P(A = 1, B = 1|C = 1) = 0.75$
- $P(A = 0, B = 1|C = 1) = 0.05$
- $P(A = 1, B = 0|C = 1) = 0.05$
- $P(A = 0, B = 0|C = 1) = 0.15$
- $P(A = 1, B = 1|C = 0) = 0.04$

- $P(A=0, B=1|C=0) = 0.01$
- $P(A=1, B=0|C=0) = 0.01$
- $P(A=0, B=0|C=0) = 0.94$

the probability that the patient doesn't have the flu according to the full generative model:

$$P(C=0|A=1, B=1)$$

$$= \frac{P(A=1, B=1|C=0)}{P(A=1, B=1)}$$

$$= \frac{P(A=1, B=1|C=0) * P(C=0)}{P(A=1, B=1|C=1) * P(C=1) + P(A=1, B=1|C=0) * P(C=0)}$$

$$= \frac{(0.04 * 0.8)}{(0.75 * 0.2 + 0.04 * 0.8)} = 0.175824176$$

(b)

$P(A=1|C=0) = 0.05$

$P(B=1|C=0) = 0.05$

$P(C=0) = 0.8$

$P(A=1|C=1) = 0.8$

$P(B=1|C=1) = 0.8$

$P(C=1) = 0.2$

$P(A=1) = P(A=1|C=0) * P(C=0) + P(A=1|C=1) * P(C=1) = 0.2$

$P(B=1) = P(B=1|C=0) * P(C=0) + P(B=1|C=1) * P(C=1) = 0.2$

the probability that the above patient doesn't have the flu according to a naive Bayes model:

$$P(C=0|A=1, B=1)$$

$$= \frac{P(A=1, B=1|C=0) * P(C=0)}{P(A=1, B=1)}$$

$$= \frac{P(A=1|C=0) * P(B=1|C=0) * P(C=0)}{P(A=1) * P(B=1)}$$

$$= \frac{0.05 * 0.05 * 0.8}{0.2 * 0.2} = 0.05$$

(c)

The probability calculated in (a) is 0.175824176 and the probability calculated in (b) is 0.05. We can see a significant difference between these two results. Thus, we can say that the above approaches do give significantly different probabilities that the above patient doesn't have the flu.

The approach used in (a) gives a more reasonable estimate, and the approach in (b) gives a less reasonable estimate. Reason: The prerequisite of the Naive Bayes approach is that features should be independent of each other, but cough and sneeze are not independent of each other, so the result of the Naive Bayes approach is less reasonable.

Written Problem 3

(a)

$$\theta_t = \sum_{i=1}^{n} \beta_i^{(t)} \phi(x_i)$$

$$\theta_{t+1} = \theta_t - \alpha(\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta_t^T \phi(x_i))\phi(x_i) + \lambda\theta_t)$$

$$= \sum_{i=1}^{n} \beta_i^{(t)} \phi(x_i) - \frac{\alpha}{n}(\sum_{i=1}^{n} y_i \phi(x_i) - \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_j^{(t)} \phi^T(x_j)\phi^2(x_i) + \lambda n \sum_{i=1}^{n} \beta_i^{(t)} \phi(x_i))$$

$$= \sum_{i=1}^{n} \phi(x_i)[\beta_i^{(t)} - \frac{\alpha}{n} y_i + \sum_{j=1}^{n} \beta_j^{(t)} L_{ij} - \lambda\alpha\beta_i^{(t)}]$$

$$= \sum_{i=1}^{n} \phi(x_i)[\beta_i^{(t)}(1 - \lambda\alpha) - \frac{\alpha}{n} y_i + \sum_{j=1}^{n} \beta_j^{(t)} L_{ij}]$$

$$=> \beta^{(t+1)} = \beta_i^{(t)}(1 - \alpha\lambda) - \frac{\alpha}{n} y_i + \sum_{j=1}^{n} \beta_j^{(t)} L_{ij}$$

(b)

$$\hat{y} = \theta_t^T \phi(x_{test})$$

$$= \sum_{i=1}^{n} \beta_i^{(t)} \phi^T(x_i)\phi(x_{test})$$

$$= \sum_{i=1}^{n} \beta_i^{(t)} K(x_i, x_{test})$$

(c) Explanation/Reason:

In the equation $\hat{y} = \sum_{i=1}^{n} \beta_i^{(t)} K(x_i, x_{test})$ that we came out in (b),

1) there isn't ϕ or θ in it, so even though θ and φ(*x*) are infinite-dimensional, they does not prevent us from performing gradient descent and computing predicted values $\hat{y} = \theta_t^T \phi(x_{test})$;

2) $K(x, x^{'})$ is easy to compute;

3) the β is only related to the number of data points.

In conclusion, the components of this equation are all finite, so we can say that it can be calculated.