Rui Maki
Sidarth Wadhwa

Homework 2

CS5785
October 6, 2021

Programming Exercises

Binary Classification on Text Data

1. Upon examination, it turns out that 42.966% of the tweets are about actual disasters. This implies that 57.034% of the tweets are not about actual disasters.

2. The dataset is split randomly using sklearn's train_test_split() function.

3. For preprocessing, we do the following steps:

   (a) Convert all entries to lowercase: This is done in order to make sure that the model does not begin to see the same in uppercase and lowercase as different from each other (due to the underlying fact that the ASCII code for them is different).

   (b) Dop URLs: They do not add any information to what we are trying to do given the fact that they are just addresses.

   (c) Drop mentions: These, again, do not add any information.

   (d) Remove punctuation: Irrelevant to detecting the target variable.

   (e) Lemmatize the words: To prevent confusion arising from word conjugation.

   (f) Remove stop words from the text to remove inconsequential words that are common, such as articles and the like.

   (g) Word Tokenization: Convert the words to token to remove any ordinal relations and to make them easier to process.

4. The decision was made to keep 'M' as 6 was taken based on the fact that, in Fig.1, the elbow seems to occur at that point, which should be a good compromise between maintaining sensitivity to keywords while reducing the feature space appreciably.

5. Logistic Regression

   (a) With no regularization, there is a stark difference between the training and development set F1 scores, which highlights the fact that the model is probably overfitting the training set.
   F1 score for Training set: 0.922
   F1 score for Development set: 0.693

   (b) With L1 regularization,
   F1 score for Training set: 0.831
   F1 score for Development set: 0.734

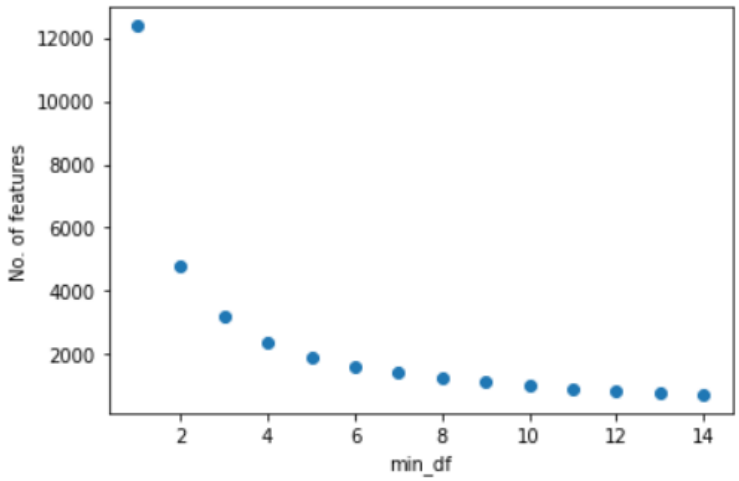


Figure 1: Graph of the number of features v/ min_df

(c) With L2 regularization,
F1 score for Training set: 0.853
F1 score for Development set: 0.737

(d) The L2 regularizer performs the best for our data, since its development set accuracy is the highest, as listed above (and it does converge the fastest, by far). As for overfitting, the unregularized model seemed to have a problem with that, but both the regularization paradigms seemed to work well to address this issue.

(e) The important words, based on the coefficients (theta) are as shown in Fig.2.

6. Bernoulli Naive Bayes': After training the model (with alpha=1 for Laplace Smoothing), the F1 score obtained by running the model on the development set comes out to be 0.510.

7. Model Comparison:
Logistic Regression performed much better in the classification task at hand. The generative classifier, i.e. Naive Bayes', performed poorly on both the training and development sets. Tweets are complex data points to work with, because they relate to how humans use language, which in and of itself is a complex combination of words that are strung together and have interdependencies that create meaning. Naive Bayes' assumes that the input variables are all distributed I.I.D., which means that it eschews the consideration of the relationships among different words in a sentence. This is to say, more or less, it reads each word in isolation, without the context of the sentence it is a part of, which makes it a bad fit for tasks that deal with data as complex as human language. Discriminative classifiers, on the other hand, do not [necessarily] make this assumption, which makes them well suited for tasks such as natural language

	Keys	Coeff
250	name	1.569240
1522	young	1.573206
356	obama	1.582352
355	cut	1.586177
1021	stay	1.588724
384	release	1.595699
274	automatic	1.615677
213	destroy	1.616816
955	equipment	1.618396
1437	rn	1.683167
1290	worry	1.733745
1326	link	1.763333
271	hand	1.840134
444	evacuation	1.906900
489	affected	1.923579
1367	2014	1.978250
400	18	2.027466
1569	rocky	2.034710
457	derailment	2.096169
703	nah	2.521804

Figure 2: Important words based on coefficient values

processing.

8. N-Gram modelling: The decision is made to set M=4, using the same reasoning as used for selecting the M value as 6 in subpart 4 above. The graph for N-Gram modelling is as shown in Fig.3. The elbos occurs around 4, hence that is chosen to be the value.
   Total number of 1-grams: 2382
   Total number of 2-grams: 908
   Ten 2-grams from the vocabulary are shown in Fig.4.

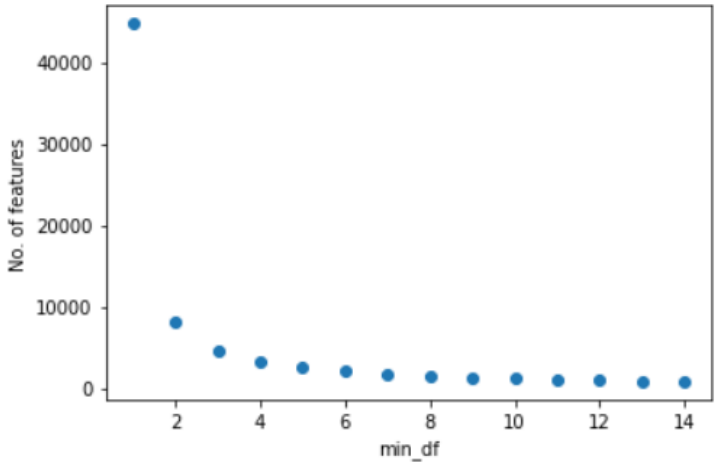


Figure 3: Graph of the number of features v/ min_df for N-Gram

```
potus strategicpatience
strategicpatience strategy
strategy genocide
genocide refugees
refugees idp
idp internally
internally displaced
displaced people
people horror
horror etc
```

Figure 4: 10 2-grams

Upon implementing Logistic Regression (L2):
F1 score for Training set: 0.886
F1 score for Development set: 0.743
Upon implementing Naive Bayes':
F1 score for Training set: 0.536
F1 score for Development set: 0.186

As for the difference from the bag-of-words model, there is no significant improvement for logistic regression (despite a marginal bump in performance), but there is a strong decline in the performance of Naive Bayes'. This implies that Naive Bayes' doesn't work well with 2-grams, and this happens because 2-grams try to figure in some form of relation for the words occurring in the vocabulary by linking them with the word that follows. This could, perhaps, be at odds with the assumption Naive Bayes' makes about the distribution of the data being fed to it. This technique also includes the possibility of each word being included twice.

9. After training the N-Gram model for Logistic Regression (L2), and submitting the results to Kaggle, The score we got is 0.78884. This score is more or less in line with the performance this model was showing on the development set. This increment could be chalked up to the fact that for this final round of training, the model was trained on the entire dataset, which gave it more data to augment the coefficient values with.

Written Exercises

Part I: Naive Bayes' with Binary Features

Considering the notation to be as follows:

- *Cov*: Patient has Covid
- *NCov*: Patient doesn't have covid
- *C*: Patient has a cough
- *NC*: Patient does not have a cough
- *F*: Patient is running a fever
- *NF*: Patient is not running a fever

Based on the probabilities given in the question, we have the following:

- *P(Cov)*: 0.1
- *P(NCov)*: 0.9
- For the patients who have COVID:
  - *P(F,C |Cov)*: 0.75
  - *P(F,NC |Cov)*: 0.05
  - *P(NF,C |Cov)*: 0.05
  - *P(NF,NC |Cov)*: 0.15
- For the patients who do not have COVID:
  - *P(F,C |NCov)*: 0.04
  - *P(F,NC |NCov)*: 0.01
  - *P(NF,C |NCov)*: 0.01
  - *P(NF,NC |NCov)*: 0.94

(a) Using Bayes' rule, we can calculate the probability of a person not having COVID given that they have a fever and a cough, i.e., *P(NCov |F,C)*:

$$
\begin{aligned}
P(NCov|F,C) &= \frac{P(F,C|NCov) * P(NCov)}{P(F,C|NCov) * P(NCov) + P(F,C|Cov) * P(Cov)} \\
&= \frac{0.04 * 0.9}{0.04 * 0.9 + 0.75 * 0.1} \\
&= \frac{0.036}{0.036 + 0.075} \\
&= 0.324
\end{aligned}
$$

(b) Using the Naive Bayes' approach and knowing that the dataset is sampled I.I.D., we can calculate *P(NCov |F,C)* [This all assumes that the symptoms are independent of each other.]:

$$
\begin{aligned}
P(F,NC|NCov) + P(F,C|NCov) = P(F|NCov) &= 0.01 + 0.04 = 0.05 \\
P(NF,C|NCov) + P(F,C|NCov) = P(C|NCov) &= 0.01 + 0.04 = 0.05 \\
P(F,NC|Cov) + P(F,C|Cov) = P(F|Cov) &= 0.05 + 0.75 = 0.80 \\
P(NF,C|Cov) + P(F,C|Cov) = P(C|Cov) &= 0.05 + 0.75 = 0.80
\end{aligned}
$$

$$
\begin{aligned}
P(NCov|C,F) &= \frac{P(C|NCov) * P(F|NCov) * P(NCov)}{P(evidence)} \\
P(evidence) &= P(C|NCov) * P(F|NCov) * P(NCov) \\
&\quad + P(C|Cov) * P(F|Cov) * P(Cov) \\
P(NCov|C,F) &= \frac{0.05 * 0.05 * 0.9}{0.05 * 0.05 * 0.90 + 0.80 * 0.80 * 0.10} \\
P(NCov|C,F) &= 0.033
\end{aligned}
$$

(c) As can be seen above, the two approaches supply us with probabilities that are rather different. This difference can be attributed to the fact that Naive Bayes' assumes that the symptoms supplied to it exist independent of one-another, whereas Bayes' rule does not make that assumption.
The model that produces a more relevant result is probably going to be Bays' rule, because it factors in the interlinking of symptoms that could contribute to the likelihood of the person in question having Covid.

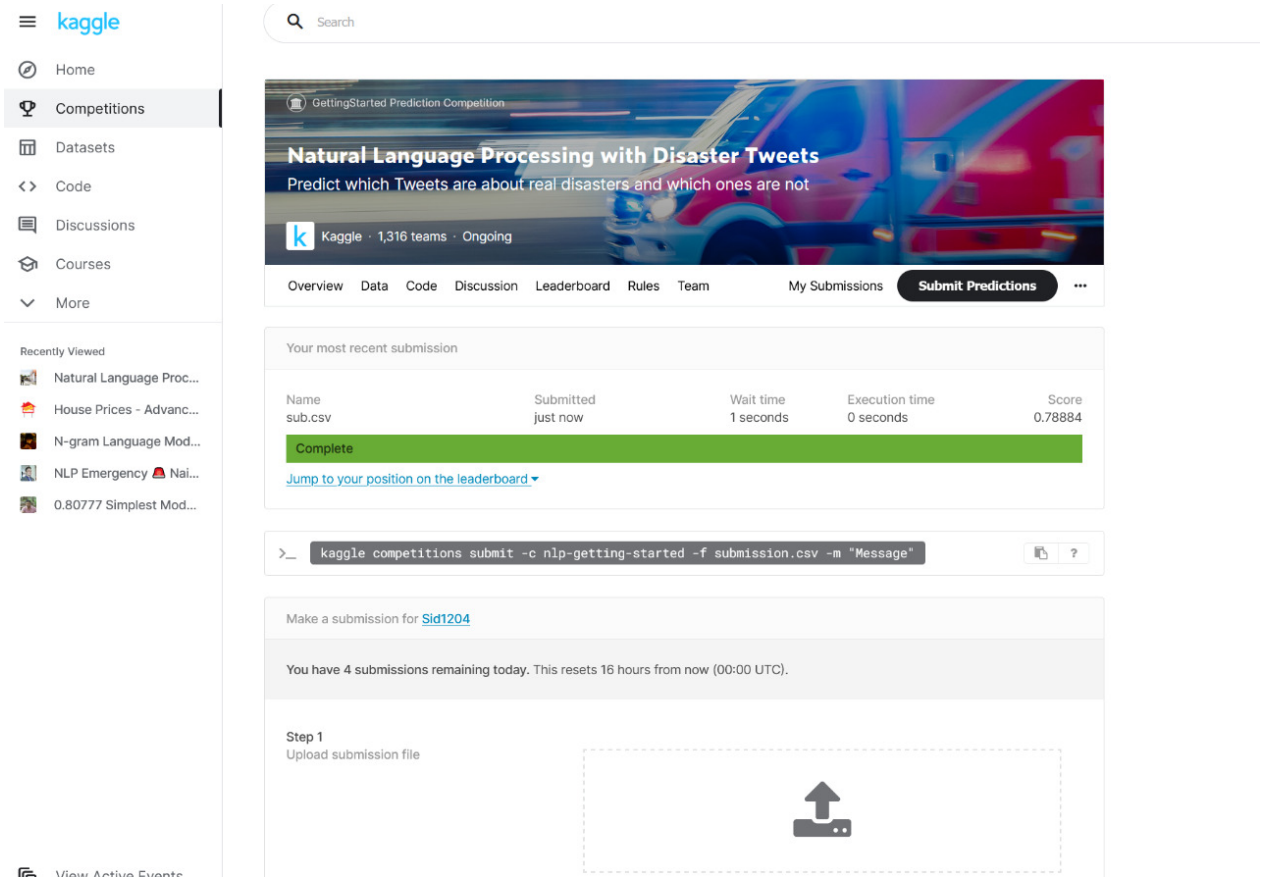


Figure 5: Submission to Kaggle

Part II: Categorical Naive Bayes'

1.

$$\overrightarrow{\phi} = (\phi_1, \phi_2, ..., \phi_K)$$
$$max_{\overrightarrow{\phi}} \sum_{i=1}^{n} logP_\theta(y = y^{(i)}; \phi)$$

Where $J(\overrightarrow{\phi})$ is the objective function,

$$J(\overrightarrow{\phi}) = \sum_{i=1}^{n} logP_\theta(y^{(i)}; \phi)$$
$$= \sum_{i=1}^{n} log\phi_{y^{(i)}} - n \cdot log \sum_{k=1}^{K} \phi_k$$
$$= \sum_{k=1}^{K} \sum_{i:y^{(i)}=k} log\phi_k - n \cdot log \sum_{k=1}^{K} \phi_k$$

Upon taking the derivative of the same and setting it to zero,

$$\frac{\phi_k}{\sum_l \phi_l} = \frac{n_k}{n}$$

For each k where $n_k = |\{i : y^{(i)} = k\}|$ is the number of targets with the class k.
This implies that the optimal ϕ_k is the ratio of the data points that belong to the class k to the size of the full dataset.

2. For each value of j,k:

$$\psi_{jk} = \psi_{jk1}, \psi_{jk2}, ..., \psi_{jkL}$$

These represent the parameters of the categorical distribution over L outcomes, with all the L possible values of the observation at j given that it belongs to the class k.
Taking the likelihood term as follows,

$$argmax_{\psi_{jk}} \sum_{i:y^{(i)}=k} logP(x_j^{(i)}|y^{(i)}; \psi_{j,k})$$

We can see that these terms exist for each value of k. Assuming the size of each term to be d, we can extrapolate that the total number of terms would be $K * d$, wherein each term can be indexed by j and k.
Going off of the proof in the previous part, we can see that this could intuitively be

written as the following.

$$\psi_{jkl} = \frac{n_{jkl}}{n_k}$$

where n_k the the number of terms that fall in the class k.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
import re
import string
import nltk

%matplotlib inline
```

In [2]:
```
train = pd.read_csv("C:\\Users\\sidar\\Downloads\\DisasterTweets\\train.csv")
test = pd.read_csv("C:\\Users\\sidar\\Downloads\\DisasterTweets\\test.csv")
```

In [3]:
```
X_train = train.drop(['target'], axis=1)
y_train = train['target']
y = y_train
```

Finding the number of train and test data points

In [4]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        7613 non-null   int64 
 1   keyword   7552 non-null   object
 2   location  5080 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64 
dtypes: int64(2), object(3)
memory usage: 297.5+ KB
```

In [5]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        3263 non-null   int64 
 1   keyword   3237 non-null   object
 2   location  2158 non-null   object
 3   text      3263 non-null   object
dtypes: int64(1), object(3)
memory usage: 102.1+ KB
```

Finding the number of tweets that are actually about real disasters.

In [6]:
```
percentage_real = (train['target'].sum() / train['target'].count()) * 100
percentage_real
```

Out[6]:
```
42.96597924602653
```

Which implies that the number of tweets that aren't about real disasters is 57.034.

Preprocessing

In [7]:
```
X_train.head()
```

Out[7]:

	id	keyword	location	text
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...

In [8]:
```
X = pd.DataFrame(X_train["id"])
X = pd.concat([X, X_train["keyword"].str.lower()], axis=1)
X = pd.concat([X, X_train["location"].str.lower()], axis=1)
X = pd.concat([X, X_train["text"].str.lower()], axis=1)

t = pd.DataFrame(test["id"])
t = pd.concat([t, test["keyword"].str.lower()], axis=1)
t = pd.concat([t, test["location"].str.lower()], axis=1)
t = pd.concat([t, test["text"].str.lower()], axis=1)
```

In [9]:
```
def drop_url(t):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'',t)

X['text'] = X['text'].apply(lambda x : drop_url(x))
t['text'] = t['text'].apply(lambda x : drop_url(x))
```

In [10]:
```
def drop_mentions(t):
    url = re.compile(r'@\S+')
    return url.sub(r'',t)

X['text'] = X['text'].apply(lambda x : drop_mentions(x))
t['text'] = t['text'].apply(lambda x : drop_mentions(x))
```

In [11]:
```
def punctuation(text):
    table=str.maketrans('','',string.punctuation)
    return text.translate(table)

X['text'] = X['text'].apply(lambda x : punctuation(x))
t['text'] = t['text'].apply(lambda x : punctuation(x))
```

In [12]:
```
#X_train["text"] = X_train["text"].str.split()
X.head(20)
```

Out[12]:

	id	keyword	location	text
0	1	NaN	NaN	our deeds are the reason of this earthquake ma...
1	4	NaN	NaN	forest fire near la ronge sask canada
2	5	NaN	NaN	all residents asked to shelter in place are be...
3	6	NaN	NaN	13000 people receive wildfires evacuation orde...
4	7	NaN	NaN	just got sent this photo from ruby alaska as s...
5	8	NaN	NaN	rockyfire update california hwy 20 closed in ...
6	10	NaN	NaN	flood disaster heavy rain causes flash floodin...
7	13	NaN	NaN	im on top of the hill and i can see a fire in ...
8	14	NaN	NaN	theres an emergency evacuation happening now i...
9	15	NaN	NaN	im afraid that the tornado is coming to our area
10	16	NaN	NaN	three people died from the heat wave so far
11	17	NaN	NaN	haha south tampa is getting flooded hah wait a...
12	18	NaN	NaN	raining flooding florida tampabay tampa 18 or ...
13	19	NaN	NaN	flood in bago myanmar we arrived bago
14	20	NaN	NaN	damage to school bus on 80 in multi car crash ...
15	23	NaN	NaN	whats up man
16	24	NaN	NaN	i love fruits
17	25	NaN	NaN	summer is lovely
18	26	NaN	NaN	my car is so fast
19	28	NaN	NaN	what a gooooooooaaaaaal

Next, we stem the words to increase the uniformity and make sure that tense and related conjugations don't confuse the algorithm.

In [13]:
```
lemmatizer = nltk.stem.PorterStemmer()
for x in X["text"]:
    x = lemmatizer.stem(x)
```

In [14]:
```
for x in t["text"]:
    x = lemmatizer.stem(x)
```

In [15]:
```
for a in X['keyword']:
    if a is None:
        a = ""
for a in t['keyword']:
    if a is None:
        a = ""
t.head()
```

Out[15]:

	id	keyword	location	text
0	0	NaN	NaN	just happened a terrible car crash
1	2	NaN	NaN	heard about earthquake is different cities sta...
2	3	NaN	NaN	there is a forest fire at spot pond geese are ...
3	9	NaN	NaN	apocalypse lighting spokane wildfires
4	11	NaN	NaN	typhoon soudelor kills 28 in china and taiwan

In [16]:
```
X['keyword'] = X['keyword'].apply(lambda b: str(b).replace('%20', ' '))
t['keyword'] = t['keyword'].apply(lambda b: str(b).replace('%20', ' '))

X.head(20)
```

Out[16]:

	id	keyword	location	text
0	1	nan	NaN	our deeds are the reason of this earthquake ma...
1	4	nan	NaN	forest fire near la ronge sask canada
2	5	nan	NaN	all residents asked to shelter in place are be...
3	6	nan	NaN	13000 people receive wildfires evacuation orde...
4	7	nan	NaN	just got sent this photo from ruby alaska as s...
5	8	nan	NaN	rockyfire update california hwy 20 closed in ...
6	10	nan	NaN	flood disaster heavy rain causes flash floodin...
7	13	nan	NaN	im on top of the hill and i can see a fire in ...
8	14	nan	NaN	theres an emergency evacuation happening now i...
9	15	nan	NaN	im afraid that the tornado is coming to our area
10	16	nan	NaN	three people died from the heat wave so far
11	17	nan	NaN	haha south tampa is getting flooded hah wait a...
12	18	nan	NaN	raining flooding florida tampabay tampa 18 or ...
13	19	nan	NaN	flood in bago myanmar we arrived bago
14	20	nan	NaN	damage to school bus on 80 in multi car crash ...

	id	keyword	location	text
15	23	nan	NaN	whats up man
16	24	nan	NaN	i love fruits
17	25	nan	NaN	summer is lovely
18	26	nan	NaN	my car is so fast

In [17]:
```
from nltk.corpus import stopwords

stop_words = stopwords.words('english')
X['text'] = X['text'].apply(lambda x: ' '.join([word for word in x.split() if
t['text'] = t['text'].apply(lambda x: ' '.join([word for word in x.split() if
```

In [18]:
```
t.head(20)
```

Out[18]:

	id	keyword	location	text
0	0	nan	NaN	happened terrible car crash
1	2	nan	NaN	heard earthquake different cities stay safe ev...
2	3	nan	NaN	forest fire spot pond geese fleeing across str...
3	9	nan	NaN	apocalypse lighting spokane wildfires
4	11	nan	NaN	typhoon soudelor kills 28 china taiwan
5	12	nan	NaN	shakingits earthquake
6	21	nan	NaN	theyd probably still show life arsenal yesterd...
7	22	nan	NaN	hey
8	27	nan	NaN	nice hat
9	29	nan	NaN	fuck
10	30	nan	NaN	dont like cold
11	35	nan	NaN	nooooooooo dont
12	42	nan	NaN	dont tell
13	43	nan	NaN	
14	45	nan	NaN	awesome
15	46	ablaze	london	birmingham wholesale market ablaze bbc news fi...
16	47	ablaze	niall's place | saf 12 squad |	wear shorts race ablaze
17	51	ablaze	nigeria	previouslyondoyintv toke makinwaû²s marriage ...
18	58	ablaze	live on webcam	check nsfw
19	60	ablaze	los angeles, califnordia	psa iû²m splitting personalities techies foll...

In [19]:
```
X['text'] = X.apply(lambda row: nltk.word_tokenize(row['text']), axis=1)
t['text'] = t.apply(lambda row: nltk.word_tokenize(row['text']), axis=1)
```

In [20]:
```
t.head()
```

Out[20]:

	id	keyword	location	text
0	0	nan	NaN	[happened, terrible, car, crash]
1	2	nan	NaN	[heard, earthquake, different, cities, stay, s...
2	3	nan	NaN	[forest, fire, spot, pond, geese, fleeing, acr...
3	9	nan	NaN	[apocalypse, lighting, spokane, wildfires]
4	11	nan	NaN	[typhoon, soudelor, kills, 28, china, taiwan]

Splitting Training and Dev sets

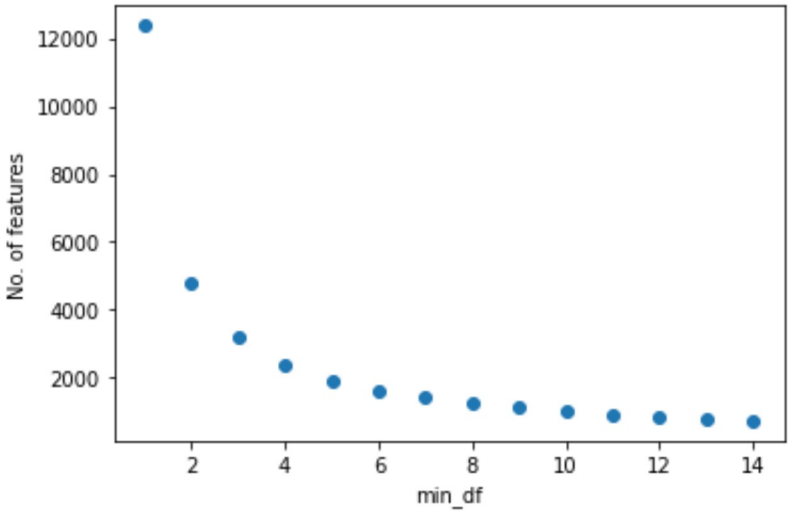
In [21]:
```
X_train, X_dev, y_train, y_dev = train_test_split(X, y_train, test_size=0.33,
```

Bag of Words Model

In [22]:
```
def dummy(doc):
    return doc

data = []
for i in range(1,15):
    cVect= CountVectorizer(min_df=i, binary=True, tokenizer=dummy, preprocess
    tFit = cVect.fit_transform(X_train['text'])
    data.append(tFit.shape[1])

plt.scatter(list(range(1,15)), data)
plt.xlabel('min_df')
plt.ylabel('No. of features')
plt.show()
```



Taking min_df to be 6 since the elbow seems to occur at that point, which should be a good compromise between maintaining sensitivity to keywords while reducing the feature space appreciably.

In [23]:
```
cv = CountVectorizer(min_df=6, binary=True, tokenizer=dummy, preprocessor=dumm
```

In [24]:
```
X_train_cv = cv.fit_transform(X_train['text'])
```

In [25]:
```
cv.vocabulary_
```

Out[25]:
```
{'photo': 1056,
 'around': 123,
 'rubble': 1191,
 'end': 475,
 'man': 879,
 'crush': 359,
 'everyday': 496,
 'yes': 1611,
 'feel': 542,
 'root': 1186,
 'shame': 1263,
 'found': 580,
 'trauma': 1452,
 'everyone': 497,
 'would': 1595,
 'join': 785,
 'isis': 768,
 'get': 612,
 'trying': 1470,
 'detonate': 408,
 'hot': 718,
 'dogs': 428,
 'fort': 579,
 'potus': 1088,
 'strategy': 1353,
 'genocide': 609,
 'refugees': 1144,
```

```
'internally': 758,
'displaced': 424,
'people': 1049,
'horror': 713,
'etc': 487,
'stream': 1354,
'hurricane': 728,
'reddits': 1143,
'new': 977,
'content': 331,
'policy': 1079,
'goes': 627,
'effect': 465,
'many': 882,
'horrible': 712,
'subreddits': 1365,
'banned': 159,
'quarantined': 1112,
'god': 625,
'right': 1171,
'love': 865,
'obliterated': 998,
'ball': 156,
'fuck': 590,
'box': 219,
'flames': 560,
'upon': 1495,
'bioterrorism': 186,
'2': 19,
'4': 33,
'ends': 477,
'mass': 887,
'murder': 955,
'loud': 863,
'ass': 133,
'thunder': 1421,
'wanted': 1533,
'bomb': 210,
'blood': 202,
'pressure': 1098,
'dont': 430,
'need': 973,
'shit': 1270,
'reddit': 1142,
'quarantine': 1111,
'offensive': 1001,
'ceo': 284,
'close': 303,
'making': 877,
'money': 937,
'50': 38,
'bloody': 203,
'hate': 672,
'weapon': 1547,
'force': 573,
'change': 286,
'says': 1217,
'fall': 518,
'like': 836,
```

```
'wild': 1568,
'fires': 556,
'wan': 1531,
'na': 962,
'mini': 925,
'heat': 683,
'wave': 1543,
'pick': 1059,
'ground': 641,
'drowned': 446,
'hit': 705,
'hard': 668,
'twice': 1479,
'time': 1425,
'different': 417,
'10': 2,
'twister': 1480,
'wont': 1583,
'running': 1196,
'post': 1085,
'pic': 1058,
'16yr': 13,
'old': 1012,
'pkk': 1063,
'suicide': 1367,
'bomber': 212,
'detonated': 409,
'turkey': 1472,
'army': 122,
'trench': 1457,
'released': 1149,
'much': 952,
'going': 628,
'head': 677,
'id': 733,
'rather': 1126,
'talk': 1390,
'latest': 814,
'homes': 710,
'razed': 1127,
'northern': 986,
'california': 251,
'wildfire': 1569,
'abc': 53,
'news': 978,
'guy': 653,
'bought': 218,
'car': 264,
'police': 1078,
'door': 431,
'crashed': 350,
'n': 961,
'amp': 99,
'text': 1404,
'send': 1246,
'book': 216,
'electrocute': 468,
'fatalities': 532,
'migrant': 916,
```

```
'boat': 207,
'capsizes': 262,
'med': 896,
'hundreds': 726,
'\x89û': 1622,
'fan': 524,
'sorry': 1316,
'world': 1588,
'blight': 196,
'lot': 861,
'soul': 1318,
'searching': 1228,
'officer': 1003,
'wounded': 1596,
'suspect': 1379,
'dead': 381,
'shots': 1274,
'\x89ûó': 1626,
'doesnt': 426,
'let': 826,
'go': 624,
'uses': 1501,
'seismic': 1243,
'hijack': 698,
'buses': 241,
'arrested': 124,
'enugu': 482,
'according': 59,
'nigerian': 981,
'via': 1511,
'russia': 1199,
'food': 571,
'crematoria': 354,
'provoke': 1107,
'outrage': 1022,
'country': 342,
'famine': 523,
'could': 341,
'anyone': 110,
'tell': 1396,
'taking': 1389,
'long': 853,
'please': 1074,
'better': 179,
'think': 1413,
'anything': 111,
'done': 429,
'hour': 719,
'class': 300,
'bombed': 211,
'pandemonium': 1031,
'aba': 51,
'woman': 1579,
'delivers': 390,
'baby': 150,
'without': 1577,
'face': 514,
'photos': 1057,
'wow': 1598,
```

```
'lies': 832,
'within': 1576,
'every': 495,
'storm': 1350,
'taken': 1387,
'today': 1428,
'rainstorm': 1123,
'entire': 481,
'community': 322,
'destroy': 403,
'industry': 745,
'drop': 442,
'31': 30,
'md': 893,
'ave': 145,
'silver': 1287,
'spring': 1330,
'low': 867,
'due': 451,
'dust': 452,
'east': 458,
'bridge': 225,
'following': 570,
'sunk': 1371,
'cost': 338,
'still': 1345,
'went': 1558,
'collapsed': 311,
'front': 589,
'broke': 228,
'city': 298,
'girl': 616,
'mountain': 946,
'snowstorm': 1305,
'sister': 1295,
'rt': 1189,
'video': 1514,
'coming': 319,
'us': 1497,
'earthquake': 457,
'tsunami': 1471,
'finally': 549,
'demolished': 394,
'2013': 22,
'property': 1106,
'since': 1288,
'name': 965,
'back': 151,
'derailed': 399,
'thinking': 1414,
'crazy': 351,
'im': 738,
'trapped': 1451,
'inside': 752,
'murderer': 956,
'meltdown': 902,
'ever': 494,
'seen': 1241,
'control': 332,
```

```
'tho': 1415,
'gtgt': 646,
'lightning': 835,
'5': 37,
'6': 41,
'phone': 1055,
'holding': 707,
'electrocuted': 469,
'hand': 660,
'bad': 152,
'ambulance': 95,
'automatic': 143,
'vehicle': 1509,
'choice': 296,
'14': 10,
'ebay': 461,
'drought': 444,
'everything': 498,
'comes': 318,
'ready': 1132,
'explode': 504,
'refugio': 1145,
'oil': 1009,
'spill': 1326,
'may': 890,
'costlier': 339,
'bigger': 182,
'projected': 1105,
'school': 1220,
'year': 1609,
'burning': 238,
'buildings': 236,
'feed': 541,
'seems': 1240,
'deluge': 391,
'week': 1553,
'never': 976,
'thought': 1417,
'wtf': 1604,
'moment': 936,
'yet': 1612,
'months': 939,
'90': 49,
'next': 979,
'flash': 561,
'floods': 567,
'b': 149,
'x': 1606,
'chance': 285,
'nothing': 988,
'seeks': 1239,
'70': 45,
'years': 1610,
'atomic': 135,
'bombs': 215,
'japan': 780,
'struggles': 1361,
'war': 1535,
'past': 1042,
```

```
'anniversary': 104,
'devastation': 412,
'wrought': 1603,
'3': 28,
'words': 1585,
'wrecked': 1601,
'stock': 1346,
'men': 905,
'escape': 486,
'engulfed': 479,
'parleys': 1036,
'canyon': 261,
'crews': 355,
'investigating': 762,
'cause': 276,
'island': 771,
'rd': 1128,
'really': 1134,
'blizzard': 197,
'leaving': 821,
'mudslide': 953,
'san': 1208,
'flight': 564,
'desolate': 401,
'state': 1342,
'real': 1133,
'twitter': 1481,
'prepare': 1095,
'9': 48,
'special': 1324,
'ebola': 462,
'rain': 1121,
'power': 1089,
'cut': 365,
'obama': 996,
'yeah': 1608,
'worth': 1594,
'bc': 167,
'already': 90,
'probably': 1102,
'blown': 204,
'second': 1231,
'night': 984,
'lucky': 869,
'block': 199,
'apocalypse': 113,
'20': 20,
'women': 1580,
'bags': 155,
'bag': 153,
'handbag': 661,
'cross': 358,
'body': 209,
'shoulder': 1275,
'ladies': 804,
'handbags': 662,
'high': 696,
'survived': 1377,
'ban': 157,
```

```
'game': 600,
'set': 1257,
'match': 889,
'release': 1148,
'hostages': 717,
'videos': 1515,
'needs': 974,
'someone': 1310,
'sinkhole': 1290,
'opens': 1017,
'brooklyn': 230,
'york': 1613,
'keep': 787,
'shape': 1264,
'amazon': 94,
'tree': 1455,
'stretcher': 1357,
'flooding': 566,
'florida': 568,
'18': 15,
'19': 16,
'days': 378,
'ive': 778,
'lost': 860,
'heres': 691,
'date': 376,
'totally': 1440,
'idea': 734,
'means': 895,
'look': 855,
'violent': 1519,
'crime': 356,
'rate': 1125,
'weapons': 1548,
'guns': 652,
'become': 171,
'australia': 142,
'\x89ûï': 1623,
'emergency': 472,
'action': 63,
'plan': 1065,
'trump': 1466,
'structural': 1360,
'government': 636,
'failure': 517,
'listen': 844,
'landslide': 809,
'oh': 1008,
'wonder': 1582,
'apply': 117,
'damn': 371,
'tried': 1460,
'see': 1236,
'blew': 195,
'thats': 1407,
'played': 1070,
'salt': 1207,
'wounds': 1597,
'dad': 367,
```

```
'first': 557,
'level': 828,
'hostage': 716,
'hours': 720,
'evacuation': 490,
'order': 1018,
'roosevelt': 1185,
'town': 1442,
'one': 1014,
'nuclear': 990,
'reactor': 1129,
'nearly': 972,
'double': 432,
'blaze': 192,
'madhya': 872,
'pradesh': 1092,
'train': 1446,
'derailment': 400,
'village': 1518,
'youth': 1617,
'saved': 1213,
'lives': 848,
'fire': 553,
'cake': 249,
'looks': 857,
'terrorism': 1400,
'story': 1351,
'maybe': 891,
'bombing': 213,
'theres': 1409,
'alarm': 83,
'working': 1587,
'minutes': 928,
'evacuate': 488,
'waiting': 1526,
'winds': 1573,
'siren': 1292,
'gone': 631,
'leave': 820,
'else': 470,
'wants': 1534,
'radiation': 1118,
'24': 25,
'giant': 615,
'brown': 232,
'families': 521,
'sue': 1366,
'legionnaires': 824,
'40': 34,
'affected': 67,
'fatal': 531,
'outbreak': 1021,
'disease': 422,
'black': 190,
'eye': 510,
'space': 1322,
'battle': 163,
'occurred': 1000,
'star': 1337,
```

```
'o784': 995,
'involving': 764,
'fleets': 563,
'totaling': 1439,
'ships': 1269,
'destroyed': 404,
'stop': 1347,
'responders': 1163,
'free': 584,
'details': 407,
'co': 306,
'happy': 667,
'got': 634,
'tracks': 1443,
'caused': 277,
'hailstorm': 657,
'day': 377,
'yyc': 1620,
'collision': 314,
'hwy': 729,
'8': 47,
'single': 1289,
'left': 823,
'little': 846,
'forest': 575,
'take': 1386,
'anymore': 109,
'emotional': 474,
'wreck': 1599,
'watching': 1541,
'imagine': 740,
'fighting': 547,
'heart': 682,
'cant': 260,
'watch': 1539,
'sad': 1202,
'collided': 313,
'another': 105,
'nowplaying': 989,
'strike': 1358,
'de': 380,
'snow': 1304,
'million': 920,
'severe': 1259,
'thunderstorm': 1422,
'warning': 1536,
'ar': 118,
'pm': 1076,
'cdt': 280,
'morning': 941,
'two': 1482,
'amazing': 93,
'memories': 904,
'tears': 1394,
'rescued': 1159,
'migrants': 917,
'arrive': 125,
'literally': 845,
'bed': 172,
```

```
'put': 1110,
'beach': 168,
'hijacking': 700,
'games': 601,
'stars': 1338,
'worlds': 1590,
'collide': 312,
'heavy': 684,
'sounds': 1320,
'song': 1314,
'hear': 680,
'movie': 948,
'walking': 1529,
'away': 147,
'cars': 268,
'national': 966,
'grows': 644,
'aint': 76,
'well': 1557,
'ahead': 74,
'best': 178,
'shot': 1273,
'crash': 349,
'spot': 1329,
'flood': 565,
'combo': 316,
'cree': 353,
'led': 822,
'work': 1586,
'light': 834,
'bar': 160,
'offroad': 1007,
'lamp': 806,
'full': 593,
're\x89û': 1170,
'\x89ûò': 1625,
'death': 383,
'businesses': 244,
'deluged': 392,
'make': 875,
'stand': 1335,
'likely': 838,
'rise': 1175,
'top': 1436,
'pay': 1046,
'funtenna': 597,
'computers': 326,
'data': 375,
'sound': 1319,
'waves': 1544,
'hat': 671,
'2015': 24,
'prebreak': 1094,
'military': 919,
'watched': 1540,
'part': 1038,
'died': 415,
'perfect': 1051,
'survivors': 1378,
```

```
'know': 799,
'want': 1532,
'cliff': 301,
'life': 833,
'keeps': 788,
'actions': 64,
'upheaval': 1494,
'gas': 602,
'100': 3,
'music': 957,
'debris': 386,
'mh370': 910,
'reunion': 1167,
'glass': 621,
'screen': 1225,
'film': 548,
'read': 1130,
'hold': 706,
'remains': 1151,
'2nd': 27,
'st': 1332,
'manchester': 880,
'big': 181,
'football': 572,
'media': 897,
'cyclone': 366,
'ancient': 100,
'tablet': 1384,
'king': 796,
'aftershock': 71,
'ship': 1268,
'desolation': 402,
'smaug': 1302,
'german': 610,
'causing': 279,
'sky': 1299,
'powerful': 1090,
'drunk': 448,
'driving': 441,
'search': 1227,
'attacked': 137,
'total': 1438,
'run': 1195,
'attack': 136,
'india': 743,
'fatality': 533,
'uk': 1487,
'family': 522,
'planned': 1067,
'toddler': 1430,
'climate': 302,
'america': 96,
'held': 685,
'11': 5,
'shouldnt': 1276,
'even': 491,
'complete': 324,
'zone': 1621,
'lava': 815,
```

```
'blast': 191,
'red': 1141,
'also': 91,
'sitting': 1298,
'event': 493,
'kills': 794,
'saudi': 1211,
'mosque': 943,
'started': 1340,
'terrorist': 1401,
'feeling': 543,
'quiz': 1116,
'service': 1255,
'destruction': 406,
'friends': 588,
'house': 721,
'true': 1465,
'bin': 184,
'laden': 803,
'afghanistan': 68,
'children': 292,
'panic': 1032,
'say': 1215,
'nice': 980,
'things': 1412,
'disaster': 419,
'jobs': 783,
'nah': 964,
'getting': 614,
'u': 1485,
'smoke': 1303,
'lmao': 850,
'scared': 1218,
'number': 991,
'makes': 876,
'sense': 1248,
'lol': 851,
'blog': 201,
'annihilated': 102,
'accident': 58,
'gon': 630,
'happen': 663,
'american': 97,
'fucking': 591,
'niggas': 983,
'w': 1524,
'screams': 1224,
'turkish': 1473,
'troops': 1462,
'killed': 791,
'ppl': 1091,
'agree': 73,
'israel': 773,
'kids': 789,
'natural': 967,
'everywhere': 499,
'f': 513,
'collapse': 310,
'todays': 1429,
```

```
'bioterror': 185,
'lab': 802,
'secret': 1233,
'remember': 1152,
'centre': 283,
'r': 1117,
'economic': 463,
'learn': 817,
'grow': 643,
'survive': 1376,
'must': 959,
'pass': 1040,
'lets': 827,
'something': 1311,
'beautiful': 170,
'hiroshima': 703,
'nagasaki': 963,
'using': 1503,
'twelve': 1478,
'feared': 537,
'pakistani': 1028,
'air': 77,
'helicopter': 686,
'worldnews': 1589,
'25': 26,
'express': 508,
'derail': 398,
'picking': 1060,
'bodies': 208,
'water': 1542,
'rescuers': 1160,
'mediterranean': 899,
'governor': 637,
'parole': 1037,
'bus': 240,
'hijacker': 699,
'heard': 681,
'africa': 69,
'hazard': 675,
'associated': 134,
'tonight': 1433,
'prevent': 1100,
'rioting': 1173,
'business': 243,
'check': 289,
'science': 1221,
'libya': 830,
'pray': 1093,
'coast': 308,
'absolutely': 55,
'devastated': 411,
'miss': 930,
'girls': 617,
'boy': 220,
'poor': 1082,
'ben': 177,
'airport': 81,
'wait': 1525,
'ruin': 1192,
```

```
'called': 253,
'fear': 536,
'future': 598,
'humanity': 725,
'general': 608,
'drown': 445,
'live': 847,
'lt3': 868,
'angry': 101,
'relief': 1150,
'rescue': 1158,
'liked': 837,
'usa': 1498,
'e': 453,
'others': 1020,
'drink': 437,
'linked': 842,
'causes': 278,
'injury': 750,
'mode': 933,
'whole': 1565,
'dog': 427,
'dude': 450,
'antioch': 108,
'demolition': 395,
'lady': 805,
'alone': 88,
'shall': 1262,
'obliterate': 997,
'sometimes': 1312,
'tragedy': 1445,
'stage': 1334,
'texas': 1403,
'comment': 320,
'rules': 1194,
'changes': 287,
'windstorm': 1574,
'insurer': 755,
'seven': 1258,
'summer': 1368,
'three': 1420,
'israeli': 774,
'soldiers': 1307,
'west': 1559,
'almost': 87,
'image': 739,
'chinas': 295,
'market': 885,
'gems': 607,
'5km': 40,
'volcano': 1522,
'hawaii': 674,
'download': 433,
'information': 747,
'near': 970,
'supposed': 1374,
'august': 141,
'daily': 368,
'result': 1165,
```

```
'sandstorm': 1209,
'swallowed': 1380,
'minute': 927,
'hope': 711,
'walk': 1528,
'outside': 1023,
'added': 66,
'playlist': 1073,
'disco': 420,
'official': 1005,
'burned': 237,
'fedex': 540,
'longer': 854,
'potential': 1087,
'pathogens': 1043,
'break': 223,
'thousands': 1418,
'ablaze': 54,
'destroys': 405,
'street': 1355,
'sea': 1226,
'piece': 1062,
'side': 1282,
'took': 1435,
'round': 1188,
'allows': 86,
'help': 689,
'youre': 1616,
'ok': 1010,
'way': 1546,
'bleeding': 194,
'catastrophic': 274,
'short': 1272,
'injuries': 749,
'sinking': 1291,
'home': 709,
'killer': 792,
'queen': 1113,
'students': 1362,
'stupid': 1364,
'center': 281,
'training': 1447,
'chemical': 290,
'response': 1164,
'lots': 862,
'road': 1180,
'direction': 418,
'album': 84,
'rly': 1178,
'mp': 950,
'recount': 1139,
'\x89ûiwhen': 1624,
'saw': 1214,
'coaches': 307,
'plunging': 1075,
'good': 632,
'play': 1069,
'along': 89,
'begin': 174,
```

```
'1': 1,
'eyes': 511,
'ashes': 131,
'test': 1402,
'currently': 364,
'interesting': 757,
'saying': 1216,
'fans': 525,
'problem': 1103,
'em': 471,
'hey': 694,
'whirlwind': 1563,
'nigga': 982,
'damage': 369,
'great': 639,
'shooting': 1271,
'aug': 140,
'bbc': 166,
'harm': 669,
'sunday': 1370,
'thank': 1405,
'eyewitness': 512,
'child': 291,
'plane': 1066,
'missing': 931,
'aircraft': 78,
'guys': 654,
'eat': 460,
'ago': 72,
'dropped': 443,
'articles': 130,
'share': 1265,
'act': 62,
'save': 1212,
'self': 1244,
'coffee': 309,
'episode': 484,
'gun': 650,
'ignition': 735,
'knock': 798,
'detonation': 410,
'sensor': 1249,
'gm': 623,
'equipment': 485,
'less': 825,
'wind': 1572,
'60': 42,
'mph': 951,
'hail': 656,
'river': 1177,
'till': 1424,
'issued': 776,
'05': 0,
'nws': 992,
'call': 252,
'times': 1426,
'middle': 914,
'epicentre': 483,
'serious': 1254,
```

```
 'funny': 596,
 'traumatised': 1453,
 'drive': 439,
 'september': 1252,
 'charged': 288,
 'looking': 856,
 'temple': 1397,
 'massacre': 888,
 'start': 1339,
 'ur': 1496,
 'fight': 546,
 'trouble': 1463,
 'crisis': 357,
 'mad': 870,
 'white': 1564,
 'boys': 221,
 'worst': 1593,
 'flattened': 562,
 'came': 254,
 'flag': 559,
 'find': 551,
 'theyre': 1410,
 'typhoon': 1483,
 'soudelor': 1317,
 'aim': 75,
 'taiwan': 1385,
 '15': 11,
 'security': 1235,
```

In [26]:
```
cv_dev = CountVectorizer(binary=True, vocabulary=cv.get_feature_names(), toker
X_dev_cv = cv_dev.fit_transform(X_dev['text'])
```

Logistic Regression

No regularization

Training set

In [27]:
```
logreg_N = LogisticRegression(penalty='none', max_iter=3600, solver='saga')
logreg_N.fit(X_train_cv, y_train)
```

Out[27]:
```
LogisticRegression(max_iter=3600, penalty='none', solver='saga')
```

In [28]:
```
preds_N_train = logreg_N.predict(X_train_cv)
f1_N_train = f1_score(y_train, preds_N_train)
print(f1_N_train)
```

```
0.9221556886227544
```

Dev set

In [29]:
```
preds_N_dev = logreg_N.predict(X_dev_cv)
f1_N_dev = f1_score(y_dev, preds_N_dev)
print(f1_N_dev)
```

```
0.6929347826086957
```

This is a textbook example of overfitting, which should be remedied to some extent by regularization.

L1 regularization

In [30]:
```
logreg_l1 = LogisticRegression(solver='saga', penalty='l1', max_iter=600)
logreg_l1.fit(X_train_cv, y_train)
```

Out[30]:
```
LogisticRegression(max_iter=600, penalty='l1', solver='saga')
```

Training set

In [31]:
```
preds_l1_train = logreg_l1.predict(X_train_cv)
f1_l1_train = f1_score(y_train, preds_l1_train)
print(f1_l1_train)
```

```
0.8313840155945419
```

Dev set

In [32]:
```
preds_l1_dev = logreg_l1.predict(X_dev_cv)
f1_l1_dev = f1_score(y_dev, preds_l1_dev)
print(f1_l1_dev)
```

```
0.7342026078234702
```

There is an improvement with L1 regularization, in that the model does not seem to be overfitting the training data as badly as it was, and the dev set accuracy is higher than the unregularized version.

L2 regularization

In [33]:
```
logreg_l2 = LogisticRegression(solver='saga', penalty='l2', max_iter=100)
logreg_l2.fit(X_train_cv, y_train)
```

Out[33]:
```
LogisticRegression(solver='saga')
```

Training set

In [34]:
```
preds_l2_train = logreg_l2.predict(X_train_cv)
f1_l2_train = f1_score(y_train, preds_l2_train)
print(f1_l2_train)
```

```
0.8534939759036144
```

Dev set

In [35]:
```
preds_l2_dev = logreg_l2.predict(X_dev_cv)
f1_l2_dev = f1_score(y_dev, preds_l2_dev)
print(f1_l2_dev)
```

```
0.7368946580129805
```

L2 regularization seems to be regularizing the model more effectively, given how it does not impact the training accuracy as strongly as L1 regularization does, and it boosts the dev set accuracy slightly more.

As a result, it is evident that L2 regularization regularizes the model the best. The training set accuracy is appreciably high and the dev set accuracy is the highest of the bunch. This means that the model's learning is not hindered during the training phase, and hence it is able to learn more than it does when L1 regularization is applied, given the fact that the dev set accuracy is higher for the L2 regularized model.

In [36]:
```
print("θ with no regularization: ")
print(logreg_N.coef_)
print("θ with L1 regularization: ")
print(logreg_l1.coef_)
print("θ with L2 regularization: ")
print(logreg_l2.coef_)
```

```
θ with no regularization:
[[ 7.89535241 -1.93509431  0.06442277 ...  0.3252434   5.87451079
   3.44808668]]
θ with L1 regularization:
[[0.05889373 0.         0.         ... 0.         0.66671072 0.        ]]
θ with L2 regularization:
[[ 0.52949306  0.15574657  0.16555223 ...  0.06305008  0.71779843
  -0.01728371]]
```

In [37]:
```
keys = cv.vocabulary_.keys()
df = pd.DataFrame()
df['Keys'] = keys

coeff = list(logreg_l2.coef_)

df['Coeff'] = list(list(coeff[0]))
df = df.sort_values(by = ['Coeff'])
```

In [38]:
```
df.tail(20)
```

Out[38]:

	Keys	Coeff
250	name	1.569606
1522	young	1.573785

	Keys	Coeff
356	obama	1.582343
355	cut	1.586302
1021	stay	1.588832
384	release	1.596194
274	automatic	1.616120
213	destroy	1.617317
955	equipment	1.618753
1437	rn	1.683547
1290	worry	1.734304
1326	link	1.763474
271	hand	1.840569
444	evacuation	1.907564
489	affected	1.924017
1367	2014	1.978534
400	18	2.027841
1569	rocky	2.035121
457	derailment	2.096568

Bernoulli Naive Bayes'

In [39]:
```
n = X_train_cv.shape[0]
d = X_train_cv.shape[1]
K = 2
alpha = 1

psis = np.zeros([K,d])
phis = np.zeros([K])

for k in range(K):
    X_k = X_train_cv[y_train == k]
    sum_rows = []
    for i in range(d):
        sum_rows.append((np.sum(X_k[i])+ alpha) / (X_k.shape[0]+2*alpha))
    psis[k] = sum_rows
    phis[k] = X_k.shape[0] / float(n)

print(psis)
```

```
[[0.00138026 0.00069013 0.00138026 ... 0.00207039 0.00207039 0.00276052]
 [0.00362647 0.00453309 0.00181324 ... 0.00407978 0.0049864  0.00362647]]
```

In [40]:
```
def nb_predictions(x, psis, phis):
    K=2
    n, d=x.shape
    x=np.reshape(x, (1, n, d))
    psis=np.reshape(psis, (K,1, d))

    psis=psis.clip(1e-14,1-1e-14)

    logpy=np.log(phis).reshape([K,1])
    logpxy=x*np.log(psis)+(1-x)*np.log(1-psis)
    logpyx=logpxy.sum(axis=2)+logpy

    return logpyx.argmax(axis=0).flatten(), logpyx.reshape([K,n])

train_idx, train_logpyx=nb_predictions(X_train_cv.toarray(), psis, phis)
print(train_idx[:10])
```

```
[0 0 1 1 1 1 0 1 1 0]
```

In [41]:
```
f1_bayes_train = f1_score(y_train, train_idx)
print(f1_bayes_train)
```

```
0.5359342915811088
```

In [42]:
```
dev_idx, dev_logpyx=nb_predictions(X_dev_cv.toarray(), psis, phis)
print(dev_idx[:10])
```

```
[0 0 0 1 0 0 0 1 1 0]
```

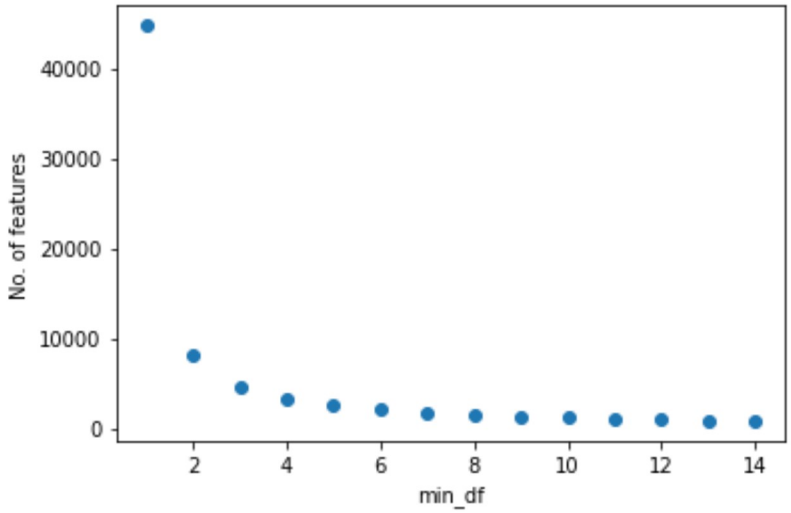
In [43]:
```
f1_bayes_dev = f1_score(y_dev, dev_idx)
print(f1_bayes_dev)
```

```
0.5101075268817205
```

N-Gram modelling

In [44]:
```
data = []
for i in range(1,15):
    cVect= CountVectorizer(min_df=i, binary=True, tokenizer=dummy, preprocess
    tFit = cVect.fit_transform(X_train['text'])
    data.append(tFit.shape[1])

plt.scatter(list(range(1,15)), data)
plt.xlabel('min_df')
plt.ylabel('No. of features')
plt.show()
```



We set min_df to 4 using the same logic as the one used for the bag-of-words model.

In [45]:
```
ngram_train = CountVectorizer(min_df=4, binary=True, tokenizer=dummy, preproce
X_train_ngram = ngram_train.fit_transform(X_train['text'])
```

In [67]:
```
count = 0
n = 0
for key in ngram_train.vocabulary_.keys():
    if ' ' in key:
        count+=1
        if n<10:
            print(key)
            n+=1
```

```
potus strategicpatience
strategicpatience strategy
strategy genocide
genocide refugees
refugees idp
idp internally
internally displaced
displaced people
people horror
horror etc
```

In [ ]:
```
print("Number of 2-gram words: "+str(count))
```

In [47]:
```
print("Number of 1-gram words: "+str(len(ngram_train.vocabulary_.keys()) - cou
```

```
Number of 1-gram words: 2382
```

In [48]:
```
ngram_dev = CountVectorizer(min_df=4, binary=True, vocabulary=ngram_train.get_
X_dev_ngram = ngram_dev.fit_transform(X_dev['text'])
```

Logistic Regression: N-Gram

In [49]:
```
logreg_l2_ngram = LogisticRegression(solver='saga', penalty='l2', max_iter=200
logreg_l2_ngram.fit(X_train_ngram, y_train)
```

Out[49]:
```
LogisticRegression(max_iter=200, solver='saga')
```

Training

In [50]:
```
preds_l2_train_ngram = logreg_l2_ngram.predict(X_train_ngram)
f1_l2_train_ngram = f1_score(y_train, preds_l2_train_ngram)
print(f1_l2_train_ngram)
```

```
0.8864942528735632
```

Dev

In [51]:
```
preds_l2_dev_ngram = logreg_l2_ngram.predict(X_dev_ngram)
f1_l2_dev_ngram = f1_score(y_dev, preds_l2_dev_ngram)
print(f1_l2_dev_ngram)
```

```
0.7430278884462151
```

Bernoulli Naive Bayes': N-Gram

In [52]:
```
n = X_train_ngram.shape[0]
d = X_train_ngram.shape[1]
K = 2
alpha = 1

psis = np.zeros([K,d])
phis = np.zeros([K])

for k in range(K):
    X_k = X_train_ngram[y_train == k]
    sum_rows = []
    for i in range(d):
        if i>=X_k.shape[0]:
            sum_rows.append(0)
        else:
            sum_rows.append((np.sum(X_k[i])+ alpha) / (X_k.shape[0]+2*alpha))
    psis[k] = sum_rows
    phis[k] = X_k.shape[0] / float(n)

print(psis)
```

```
[[0.00138026 0.00069013 0.00138026 ... 0.         0.         0.        ]
 [0.00362647 0.0099728  0.00226655 ... 0.         0.         0.        ]]
```

In [53]:
```
train_ngram_idx, train_ngram_logpyx=nb_predictions(X_train_ngram.toarray(), ps
print(train_ngram_idx[:10])
```

```
[0 0 1 0 0 0 0 0 0 0]
```

In [54]:
```
f1_bayes_train_ngram = f1_score(y_train, train_idx)
print(f1_bayes_train_ngram)
```

```
0.5359342915811088
```

In [55]:
```
dev_ngram_idx, dev_ngram_logpyx=nb_predictions(X_dev_ngram.toarray(), psis, pl
print(dev_ngram_idx[:10])
```

```
[0 0 0 0 0 0 1 0 0 0]
```

In [56]:
```
f1_bayes_dev_ngram = f1_score(y_dev, dev_ngram_idx)
print(f1_bayes_dev_ngram)
```

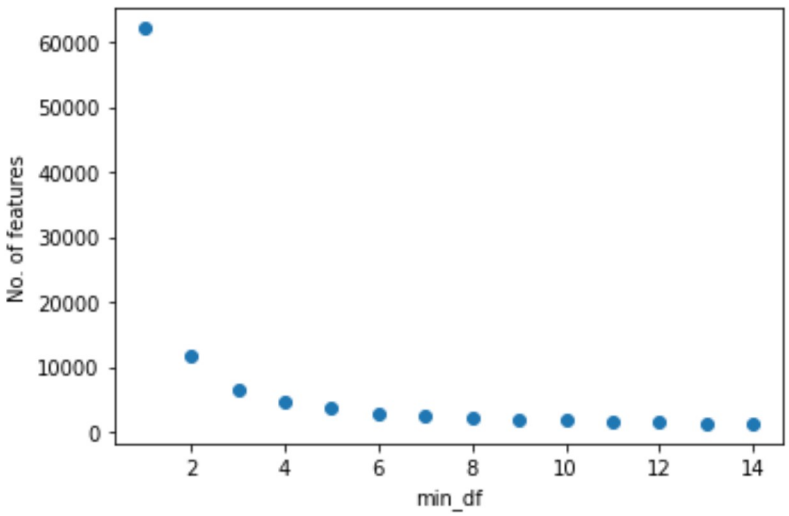
```
0.18631732168850074
```

Final run

The best performing model in our experiments has been the N-Gram model with L2 regularized Logistic Regression. Hence we decide to use this model, train it on the whole training set, and then run it on the test set to submit to Kaggle.

In [57]:
```
data = []
for i in range(1,15):
    cVect= CountVectorizer(min_df=i, binary=True, tokenizer=dummy, preprocess
    tFit = cVect.fit_transform(X['text'])
    data.append(tFit.shape[1])

plt.scatter(list(range(1,15)), data)
plt.xlabel('min_df')
plt.ylabel('No. of features')
plt.show()
```



In [58]:
```
ngram_X = CountVectorizer(min_df=4, binary=True, tokenizer=dummy, preprocesso
X_ngram_vect = ngram_X.fit_transform(X['text'])
```

In [59]:
```
ngram_test = CountVectorizer(min_df=4, binary=True, vocabulary=ngram_X.get_fe
test_ngram_vect = ngram_test.fit_transform(t['text'])
```

In [60]:
```
logreg_l2_ngram_X = LogisticRegression(solver='saga', penalty='l2', max_iter=
logreg_l2_ngram_X.fit(X_ngram_vect, y)
```

Out[60]:
```
LogisticRegression(max_iter=200, solver='saga')
```

In [61]:
```
preds_l2_X_ngram = logreg_l2_ngram_X.predict(X_ngram_vect)
f1_l2_X_ngram = f1_score(y, preds_l2_X_ngram)
print(f1_l2_X_ngram)
```

```
0.8825819007460266
```

In [62]:
```
preds_l2_test_ngram = logreg_l2_ngram_X.predict(test_ngram_vect)
```

In [63]:
```
final_preds = pd.DataFrame()
final_preds['id'] = t['id']
final_preds['target'] = preds_l2_test_ngram
```

In [64]:
```
#final_preds.to_csv('sub.csv', index=False)
```