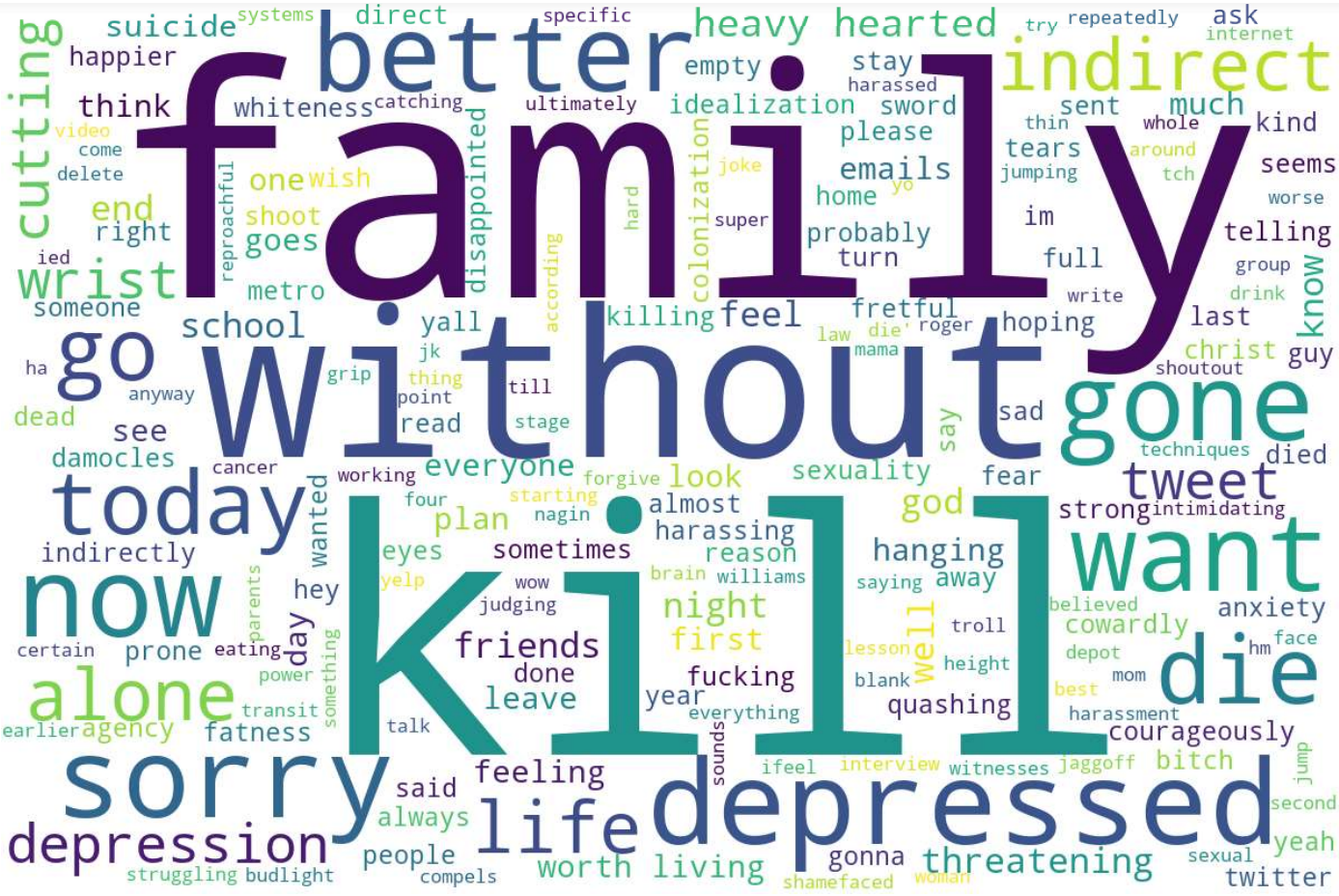
Israel Aminu Follow

Aug 17, 2020 · 4 min read ★ · Listen

Building a suicidal tweet classifier using NLP

Using Natural Language Processing to predict Suicide Ideation on Twitter.

Over the years, suicide has been one of the major causes of death worldwide, According to Wikipedia, Suicide resulted in 828,000 global deaths in 2015, an increase from 712,000 deaths in 1990. This makes suicide the 10th leading cause of death worldwide. There is also increasing evidence that the Internet and social media can influence suicide-related behaviour. Using Natural Language Processing, a field in Machine Learning, I built a very simple suicidal ideation classifier which predict whether a text is likely to be suicidal or not.



WordCloud of Tweet Analyzed

Data

I used a Twitter crawler which I found on Github, made some few changes to the code by removing hashtags, links, URL and symbols whenever it crawls data from Twitter, the data were crawled based on query parameters which contain words like:

> *Depressed, hopeless, Promise to take care of, I dont belong here, Nobody deserve me, I want to die etc.*

Although some of the text we're in no way related to suicide at all, I had to manually label the data which were about 8200 rows of tweets. I also sourced for more Twitter Data and I was able to concatenate with the one I previously had which was enough for me to train.

Building the Model

```
import pickle
import re
import numpy as np
import pandas as pd
from tqdm import tqdm
import nltk
nltk.download('stopwords')
```

I then wrote a function to clean the text data to remove any form of HTML markup, keep emoticon characters, remove non-word character and lastly convert to lowercase.

```
def preprocess_tweet(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text)
    lowercase_text = re.sub('[\W]+', ' ', text.lower())
    text = lowercase_text+' '.join(emoticons).replace('-', '')
    return text
```

After that, I applied the preprocess_tweet function to the tweet dataset to clean the data.

```
tqdm.pandas()

df = pd.read_csv('data.csv')
df['tweet'] = df['tweet'].progress_apply(preprocess_tweet)
```

Then I converted the text to tokens by using the .split() method and used word stemming to convert the text to their root form.

```
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()
def tokenizer_porter(text):
    return [porter.stem(word) for word in text.split()]
```


```
from nltk.corpus import stopwords
stop = stopwords.words('english')
```

Testing the function on a single text.

```
[w for w in tokenizer_porter('a runner likes running and runs a lot')
if w not in stop]
```

Output:

```
['runner', 'like', 'run', 'run', 'lot']
```

Vectorizer

For this project, I used the **Hashing Vectorizer** because it data-independent, which means that it is very low memory scalable to large datasets and it doesn't store vocabulary dictionary in memory. I then created a tokenizer function for the Hashing Vectorizer

```
def tokenizer(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\(|D|P)',text.lower())
    text = re.sub('[\W]+', ' ', text.lower())
    text += ' '.join(emoticons).replace('-', '')
    tokenized = [w for w in tokenizer_porter(text) if w not in stop]
    return tokenized
```

Then I created the Hashing Vectorizer object.

```
from sklearn.feature_extraction.text import HashingVectorizer
```


Model

For the Model, I used the stochastic gradient descent classifier algorithm.

```
from sklearn.linear_model import SGDClassifier
clf = SGDClassifier(loss='log', random_state=1)
```

Training and Validation

```
X = df["tweet"].to_list()
y = df['label']
```

For the model, I used 80% for training and 20% for testing.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,
                                                 y,
                                                 test_size=0.20,
                                                 random_state=0)
```

Then I transformed the text data to vectors with the Hashing Vectorizer we created earlier:

```
X_train = vect.transform(X_train)
X_test = vect.transform(X_test)
```

Finally, I then fit the data to the algorithm

```
classes = np.array([0, 1])
clf.partial_fit(X_train, y_train,classes=classes)
```


```
print('Accuracy: %.3f' % clf.score(X_test, y_test))
```

Output:

```
Accuracy: 0.912
```

I had an accuracy of 91% which is fair enough, after that, I then updated the model with the prediction

```
clf = clf.partial_fit(X_test, y_test)
```

Testing and Making Predictions

I added the text “I’ll kill myself am tired of living depressed and alone” to the model.

```
label = {0:'negative', 1:'positive'}
example = ["I'll kill myself am tired of living depressed and alone"]
X = vect.transform(example)
print('Prediction: %s\nProbability: %.2f%%'
      %(label[clf.predict(X)[0]],np.max(clf.predict_proba(X))*100))
```

And I got the output:

```
Prediction: positive
Probability: 93.76%
```

And when I used the following text “It’s such a hot day, I’d like to have ice cream and visit the park”, I got the following prediction:


The model was able to predict accurately for both cases. And that's how you build a simple suicidal tweet classifier.

You can find the notebook I used for this article here

Thanks for reading 😊