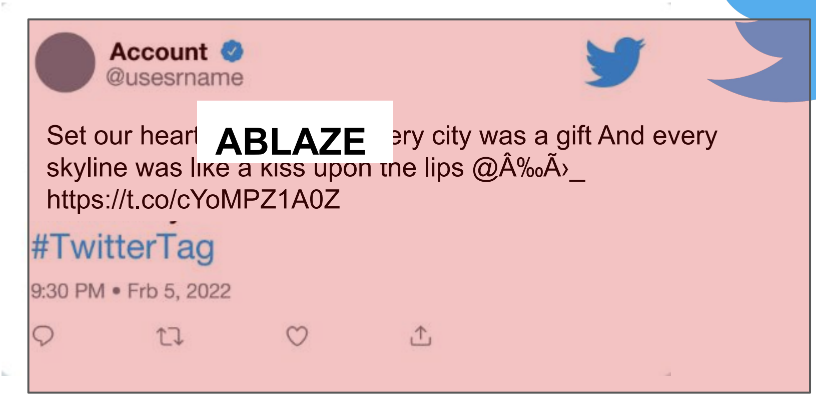
Analysis of Positional Encoding and Attention Mechanisms in Disaster-Related Tweet Classification

vs

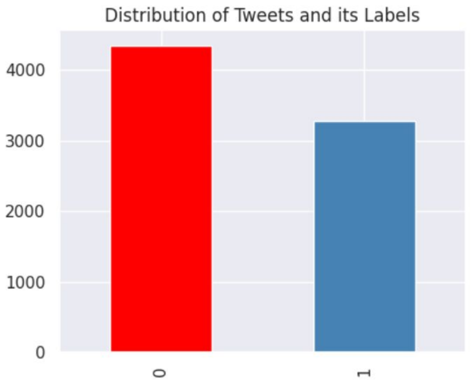
Javier Rodriguez & Edgar Leon

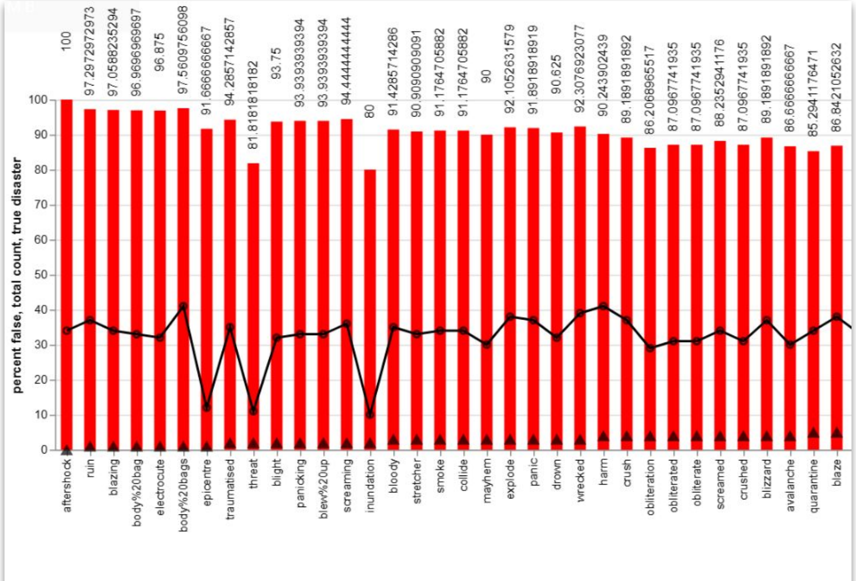
Disaster Tweets Input Data

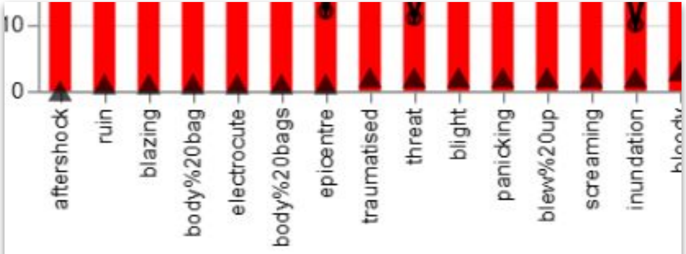


Disaster Tweets Input Data

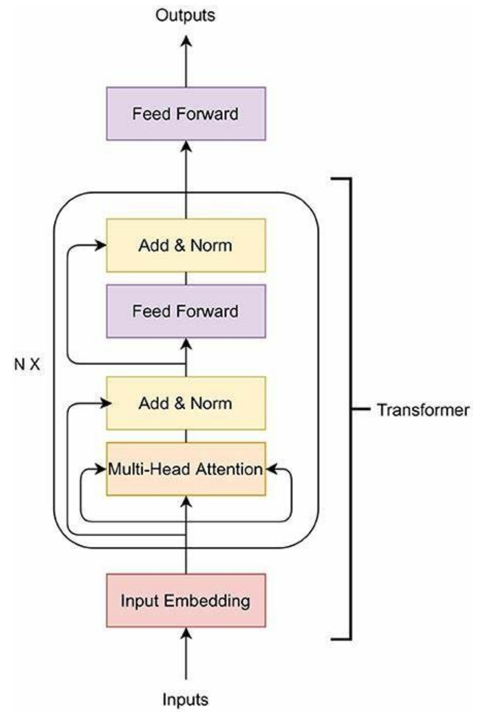
	id	keyword	location	text	target
100	144	accident	UK	.@NorwayMFA #Bahrain police had previously die...	1
101	145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya...	0
102	146	aftershock	Instagram - @heyimginog	@afterShock_DeLo scuf ps live and the game... cya	0
103	149	aftershock	304	'The man who can drive himself further once th...	0
104	151	aftershock	Switzerland	320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/yN...	0







NMT Classifier



Define the components

- The embedding and positional encoding layer
- Add and normalize
- The base attention layer
- The global self attention layer
- The feed forward network
- The encoder layer
- The encoder

The Transformer Classifier

The Classifier Transformer Architecture Along with its Components

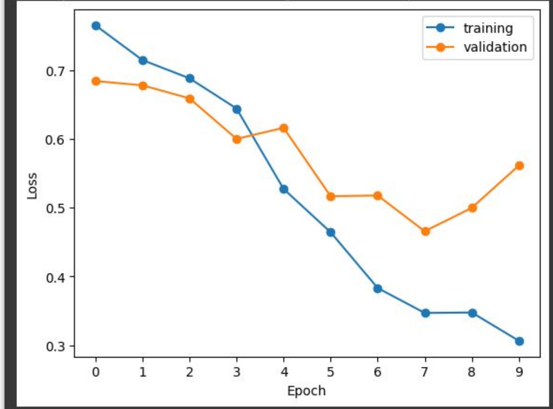
This classifier model is based on last hidden state

```
def positional_encoding(length, depth):
  depth = depth/2

  positions = np.arange(length)[:, np.newaxis]     # (seq, 1)
  depths = np.arange(depth)[np.newaxis, :]/depth   # (1, depth)

  angle_rates = 1 / (10000**depths)         # (1, depth)
  angle_rads = positions * angle_rates      # (pos, depth)

  pos_encoding = np.concatenate(
      [np.sin(angle_rads), np.cos(angle_rads)],
      axis=-1)

  return tf.cast(pos_encoding, dtype=tf.float32)

"""
The position encoding function is a stack of sines and cosines that vibrate at differe
along the depth of the embedding vector. They vibrate across the position axis.
"""
class PositionalEmbedding(tf.keras.layers.Layer):
  def __init__(self, vocab_size, d_model):
    super().__init__()
    self.d_model = d_model
    self.embedding = tf.keras.layers.Embedding(vocab_size, d_model, mask_zero=True)
    self.pos_encoding = positional_encoding(length=2048, depth=d_model)
    #the line below is new
    #self.pos_encoding = tf.cast(self.pos_encoding, tf.float64)

  def compute_mask(self, *args, **kwargs):
    return self.embedding.compute_mask(*args, **kwargs)

  def call(self, x):
    #x, _ = x #ignore attn weights
    length = tf.shape(x)[1]
    x = self.embedding(x)
    # This factor sets the relative scale of the embedding and positonal_encoding.
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    #x *= tf.math.sqrt(tf.cast(self.d_model, tf.float64))
    x = x + self.pos_encoding[tf.newaxis, :length, :]
    #print("positionalembedding: shape and type:")
    #the casting is new
    #x = tf.cast(x, tf.float64)
    #print(x.dtype)
    #print(tf.shape(x)[1])
    return x

"""The base attention layer"""
class BaseAttention(tf.keras.layers.Layer):
  def __init__(self, **kwargs):
    super().__init__()
    self.mha = tf.keras.layers.MultiHeadAttention(**kwargs)
    self.layernorm = tf.keras.layers.LayerNormalization()
```

First results on raw data

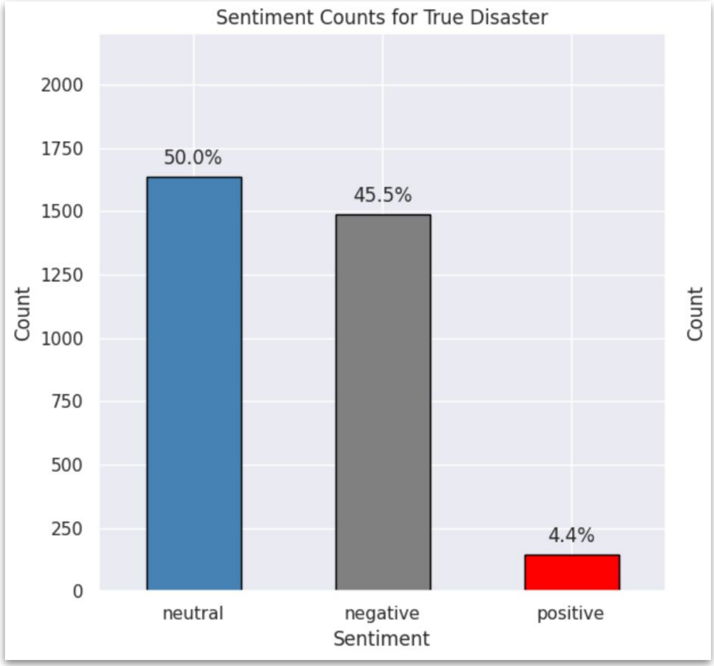
	id	keyword	location	text	target
100	144	accident	UK	.@NorwayMFA #Bahrain police had previously die...	1
101	145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya...	0
102	146	aftershock	Instagram - @heyimginog	@afterShock_DeLo scuf ps live and the game... cya	0
103	149	aftershock	304	'The man who can drive himself further once th...	0
104	151	aftershock	Switzerland	320 [IR] ICEMOON [AFTERSHOCK] | http://t.co/yN...	0

```
Epoch 10/10
84/84 [==============================] - 71s 851ms/step - loss: 0.3064 - accuracy: 0.8760 - val_loss: 0.5612 - val_accuracy: 0.7806
```



val accuracy: 0.78

Input Data EDA – content analysis & enhancement



Tweets enhanced by:

- Location
- Keyword
- Url
- Sentiment
- topic
- Irony
- emotion

```
location ablaze bbcmtd wholesale markets ablaze http://t.co/lhyxeohy6c neutral  news_&_social_concern irony ange
```

Data Wrangling Final Results

```
[ ] unknown_words, known_words = find_unknown_words(sentences, word_index)
```

```
# Call the function and print the unknown words
print(f"Words found by the tokenizer: {len(known_words)}")
print(f"Words not found by the tokenizer: {len(unknown_words)}")
```

```
Words found by the tokenizer: 17980
Words not found by the tokenizer: 9
```

```
[ ] for i in range(len(unknown_words)):
      print(list(unknown_words)[i])
```

```
`bbcnews
^oo^
^ag
^mp
\\\
3\30a
destruction\s
didn`t
^sj
```

Creation of Embeddings

-The “fasttext_english_twitter_100d.vec” file is a set of word vectors trained using the FastText library.

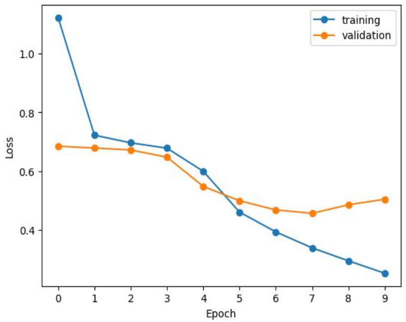
FastText is a library developed by Facebook’s AI Research (FAIR) lab. It’s dedicated to text classification and learning word representations. FastText models can be trained on more than a billion words on any multicore CPU in less than a few minutes.

Here’s a general process of how FastText trains word vectors:

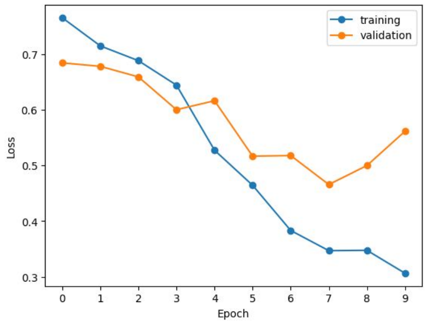
Corpus Selection: FastText trains word vectors on large text corpora. For example, FastText provides pre-trained word vectors trained, in this case, a collection of English tweets.

```
[ ] embedding_df.shape

(17984, 100)
```

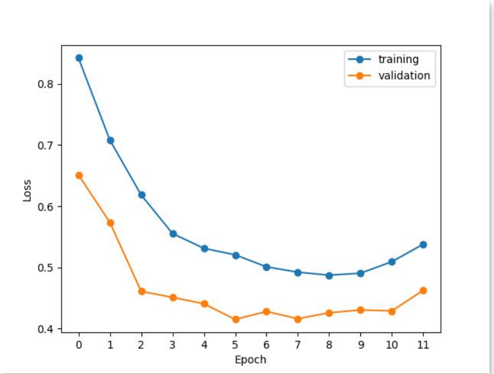
Clean Data and Creation of Embeddings



Clean and enhanced data
no embeddings: 0.79



Raw data no embeddings:
0.78



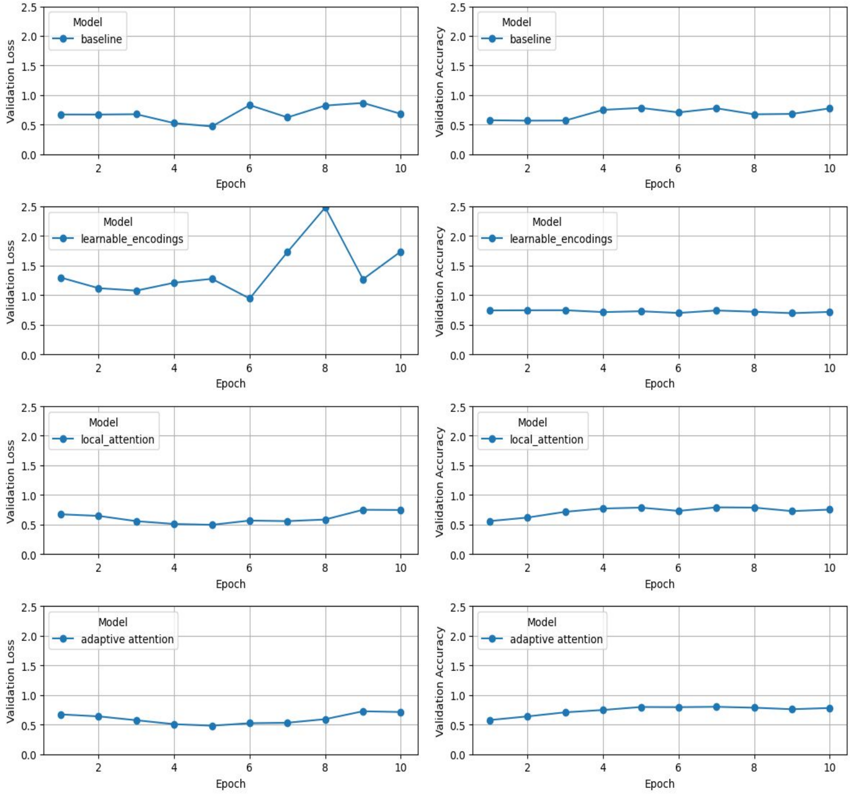
Clean and enhanced data
with embeddings: 0.84

Next explore other architecture options

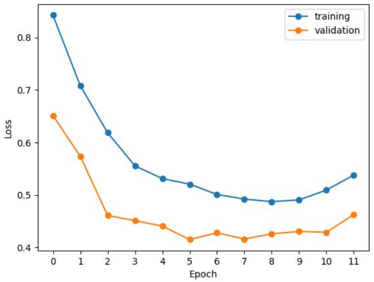
Model Comparison

Model 1: Baseline	**Validation Accuracy:** Peaked at 78.89% but fluctuated towards the end, settling at 77.76%.
Model 2: learnable encodings	**Training Accuracy:** Extremely high from the start, reaching 99.42%. **Validation Loss:** Started high and increased over epochs, indicating potential overfitting
Model 3: local attention	**Training Accuracy:** Improved consistently from 48.71% to 96.58%. **Validation Accuracy:** Increased to a peak of 80.04%, then fluctuated. Validation Loss: Started low, decreased to a point, then increased significantly in later epochs.
Model 4: adaptive attention	**Training Accuracy:** It shows robust improvement in training accuracy, the highest peak in validation accuracy, and while it does show an increase in validation loss indicating some overfitting.

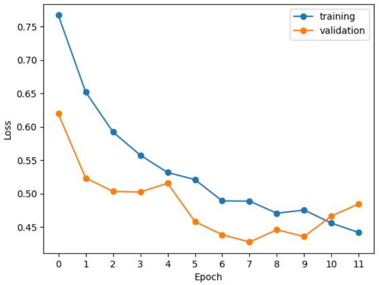
validation loss validation accuracy



Positional Encoding with Twitter Pre-Trained Embeddings



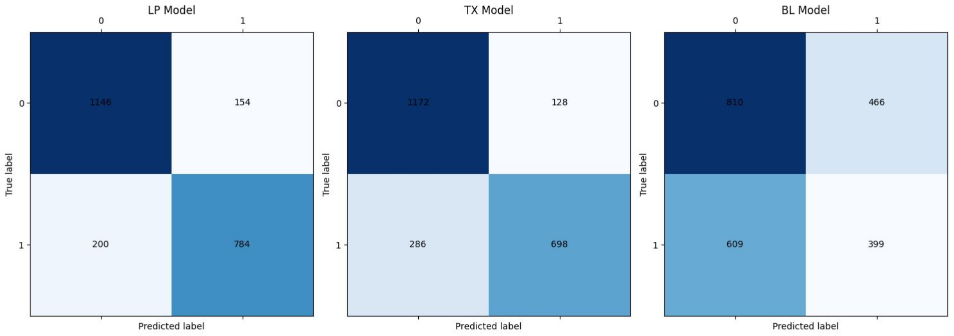
Baseline Transformer: 0.84



Learnable Positional Encoding: 0.82

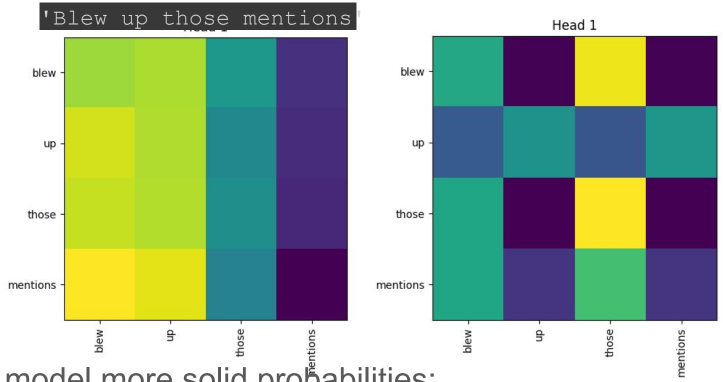
LPE vs Baseline Transformers Analysis

Lp_model more balanced:

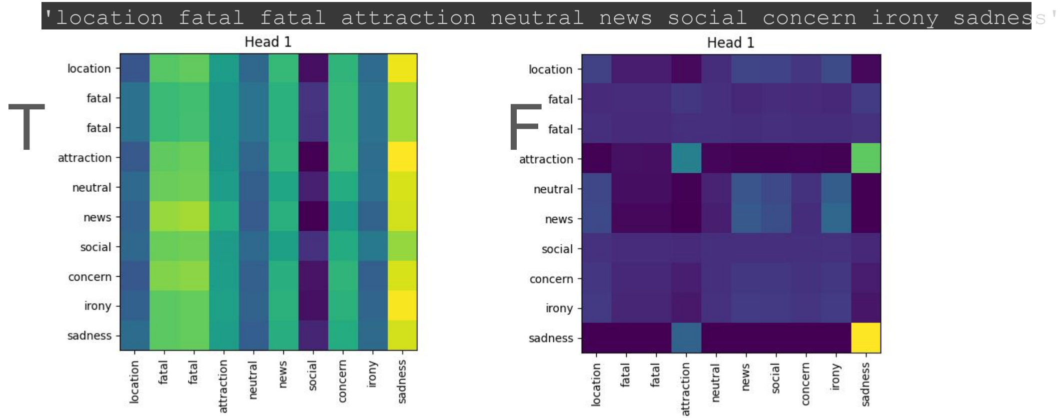


Lp_model more nuanced understanding:

F



Lp_model pays better attention:



Lp_model more solid probabilities:

0.63

0.98

Analysis Updates to Embeddings

Lp_model fine tuned each word more:

tx_model

lp_model

On average lp_model changed embeddings more (0.43 vs 0.22):

```
tx_disaster_words = top_changed_embeddings(loaded_embeddings, tx_embeddings_after_training, word_index)
tx_disaster_words_list = [item[0] for item in tx_disaster_words]
tx_disaster_words
```

```
[('suicide', 0.2542814085893426),
 ('hiroshima', 0.2366639098735998),
 ('derailment', 0.23234185947136063),
 ('positive', 0.2134087088705059),
 ('california', 0.21303300061344782),
 ('bombing', 0.21094468542664885),
 ('debris', 0.21006077369589732),
 ('wildfire', 0.20714493198224423),
 ('killed', 0.200944669891897),
 ('northern', 0.19962536227301594)]
```

```
lp_disaster_words = top_changed_embeddings(loaded_embeddings, lp_embeddings_after_training, word_index)
lp_disaster_words_list = [item[0] for item in lp_disaster_words]
lp_disaster_words
```

```
[('hiroshima', 0.496638707030925),
 ('spill', 0.4849229581973467),
 ('suicide', 0.448507128728351),
 ('bomb', 0.4369772545959775),
 ('derailment', 0.4233027778885434),
 ('northern', 0.4181329351044703),
 ('california', 0.4133995836162507),
 ('atomic', 0.40671896164405086),
 ('killed', 0.39854748212169594),
 ('wreckage', 0.3949224709598692)]
```

Challenge #3 Type of Disaster? Mislabeled as True

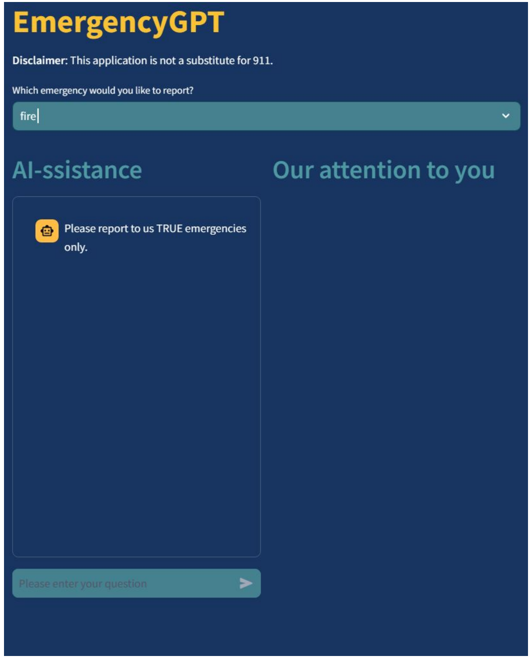
Lp_model did 82% while tx_model did 84%, we attribute it to mislabeled data:

	index	id	keyword	location	text	target	comma
108	229	328	annihilated	NaN	Ready to get annihilated for the BUCS game	1	1
281	753	1085	blew%20up	NaN	@BenKin97 @Mili_5499 remember when u were up l...	1	1
709	1831	2632	crashed	London	This guy bought my car on Tuesday police knock...	1	1
872	2310	3318	demolished	NaN	Got my first gamer troll I just demolished a k...	1	1
1154	2946	4237	drowned	NaN	I got drowned like 5 times in the damn game to...	1	1

Given better data, lp_model should do better:

	sentence	length
28	location cooool positive news social con...	61
24	location loooooool neutral news social con...	65
21	location london is cool positive news so...	69
17	location summer is lovely positive news s...	70
6705	location thunder thunder neutral news socia...	71
18	location | my car is so fast positive news ...	71
3749	location fire i see fire neutral news socia...	71
1882	location crushed crushed neutral news socia...	74
3608	location fatal fatal attraction neutral news ...	77
26	location was in nyc last week neutral news ...	77
2496	location desolate eggs desolate negative news...	78
4922	location mayhem mayhem is beautiful positive ...	78
3688	location fatality fatality https neutral new...	79
4184	location hazard get that hazard pay neutral n...	79
4092	location hail hail pic ‰ûó https positive ne...	80
3751	location fire im on fire weblink neutral ...	80
7448	location wounds 11 puncture wounds neutral ne...	80
6224	location smoke smoke eat sleep neutral news ...	80
388	location arson the sound of arson negative ne...	81
1886	location crushed crushed it https positive n...	83
823	location blizzard blizzard gamin ight neutral ...	83
3717	location fear my worst fear https negative ...	83

Future Work & Demo



Appendix

Challenge #4 Wide Range of Twitter Users

Age Difference

Cultural Difference

Model Comparison