Table of Contents

Phase 4 Project

Business Objective

To help Acme Online, an online electronics store, analyze customer tweets from their Twitter page about Apple and Google products. The result of this analysis will be used to find out which company's product has more favourable reviews and the reasons behind it - this will help Acme Online adjust their inventory accordingly.

Methodology

1. Analyze tweets to check what customers are talking about.
2. Analyze tweets to identify the most popular product - pts 1&2 can be used to tweak Acme Online's inventory accordingly.
3. For each product, we will look to see what customers like/dislike to identify opportunities for improvement, if applicable.
4. Since human intervention was used identify products based on tweets, we will attempt to build a model using NLP to automate this. We will use the f1-score for model evaluations since minimizing False Positive and False Negatives is desirable .

Dataset

Dataset sourced from CrowdFlower via data.world: https://data.world/crowdflower/brands-and-product-emotions

Analysis

In [1]:

```
#import relevant libraries
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk.stem import PorterStemmer
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.metrics import classification_report, plot_confusion_matrix
from sklearn.metrics import f1_score,accuracy_score
from sklearn.pipeline import Pipeline
import numpy as np
from nltk import word_tokenize
from gensim.models import Word2Vec
from nltk.tokenize import RegexpTokenizer
from nltk import FreqDist
import warnings
warnings.filterwarnings('ignore')
```

Importing the dataset:

In [2]:

```
df= pd.read_csv('tweets.csv',encoding='unicode escape',)
```

```
df.head()
```

Out[2]:

	tweet_text	emotion_in_tweet_is_directed_at	is_there_an_emotion_directed_at_a_brand_or_product
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion

Pre-processing

To save ourselves from lot's of keystrokes, let's rename the columns:

In [3]:
```
#renaming the columns to make it less cumbersome
df.rename(columns={'emotion_in_tweet_is_directed_at':'product_service',
                   'is_there_an_emotion_directed_at_a_brand_or_product':'emotion'},inpl
df.head()
```

Out[3]:

	tweet_text	product_service	emotion
0	.@wesley83 I have a 3G iPhone. After 3 hrs twe...	iPhone	Negative emotion
1	@jessedee Know about @fludapp ? Awesome iPad/i...	iPad or iPhone App	Positive emotion
2	@swonderlin Can not wait for #iPad 2 also. The...	iPad	Positive emotion
3	@sxsw I hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion
4	@sxtxstate great stuff on Fri #SXSW: Marissa M...	Google	Positive emotion

In [4]:
```
#getting some info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9093 entries, 0 to 9092
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   tweet_text       9092 non-null   object
```

```
 1   product_service  3291 non-null   object
 2   emotion          9093 non-null   object
dtypes: object(3)
memory usage: 213.2+ KB
```

There are no numeric values in our df which is what we'd expect given that we're analyzing tweets.

In [5]:
```
#checking for null values
df.isna().sum()
```

Out[5]:
```
tweet_text            1
product_service    5802
emotion               0
dtype: int64
```

From the above,we can see that the `product_service` column has a large number of missing values; more than 50%. Let's leave it for now and remove the one empty row in `tweet_text`

In [6]:
```
#removing the null value in the tweet_text column
df = df[df['tweet_text'].notnull()]
```

Let's take a look at the `product_service` column to see the different kinds of products that are involved:

In [7]:
```
#examining the product_service column
df['product_service'].value_counts()
```

Out[7]:
```
iPad                               946
Apple                              661
iPad or iPhone App                 470
Google                             430
iPhone                             297
Other Google product or service    293
Android App                         81
Android                             78
Other Apple product or service      35
Name: product_service, dtype: int64
```

Let's group some of the categories to facilitate easier analysis:

In [8]:
```
#let's group product/services that resemble each other for both brands. This will make 

df['product_service'].replace('Other Google product or service','Google',inplace=True)
df['product_service'].replace('Other Apple product or service','Apple',inplace=True)
df['product_service'].replace('Android App','Android',inplace=True)
df['product_service'].fillna('Not Applicable',inplace=True)

#checking
df['product_service'].value_counts()
```

Out[8]:
```
Not Applicable        5801
iPad                   946
Google                 723
Apple                  696
iPad or iPhone App     470
iPhone                 297
Android                159
Name: product_service, dtype: int64
```

Let's apply some of the common pre-processing steps when it comes to working with text data:

1. Remove capitalization

2. Remove punctuations and special characters
3. Tokenizing

In [9]:
```
# Removing capiltalization
df['tweet_text'] = df['tweet_text'].str.lower()

#removing punctuations using the default pattern in sklearn and tokenizing
basic_token_pattern = r"(?u)\b\w\w+\b"
tokenizer = RegexpTokenizer(basic_token_pattern)

#applying the tokenizer to the df and creating a new column
df['text_token'] = df['tweet_text'].apply(tokenizer.tokenize)
df.head(10)
```

Out[9]:

	tweet_text	product_service	emotion	text_token
0	.@wesley83 i have a 3g iphone. after 3 hrs twe...	iPhone	Negative emotion	[wesley83, have, 3g, iphone, after, hrs, tweet...
1	@jessedee know about @fludapp ? awesome ipad/i...	iPad or iPhone App	Positive emotion	[jessedee, know, about, fludapp, awesome, ipad...
2	@swonderlin can not wait for #ipad 2 also. the...	iPad	Positive emotion	[swonderlin, can, not, wait, for, ipad, also, ...
3	@sxsw i hope this year's festival isn't as cra...	iPad or iPhone App	Negative emotion	[sxsw, hope, this, year, festival, isn, as, cr...
4	@sxtxstate great stuff on fri #sxsw: marissa m...	Google	Positive emotion	[sxtxstate, great, stuff, on, fri, sxsw, maris...
5	@teachntech00 new ipad apps for #speechtherapy...	Not Applicable	No emotion toward brand or product	[teachntech00, new, ipad, apps, for, speechthe...
7	#sxsw is just starting, #ctia is around the co...	Android	Positive emotion	[sxsw, is, just, starting, ctia, is, around, t...
8	beautifully smart and simple idea rt @madebyma...	iPad or iPhone App	Positive emotion	[beautifully, smart, and, simple, idea, rt, ma...
9	counting down the days to #sxsw plus strong ca...	Apple	Positive emotion	[counting, down, the, days, to, sxsw, plus, st...
10	excited to meet the @samsungmobileus at #sxsw ...	Android	Positive emotion	[excited, to, meet, the, samsungmobileus, at, ...

Trending topic on Twitter

By answering this question, we can understand what customers are tweeting about. We can visualize this using a feature called ***WordCloud.***

Wordcloud

In [10]:
```
#importing the stopwords list to pass onto the WC generator
stopwords_list = stopwords.words('english')
stopwords_list.append('mention')

#droppping null values
```

```python
df.dropna(inplace=True)

#instantiate
wc = WordCloud(background_color='white',
               stopwords=stopwords_list,
               height=1000,
               width=1000,
               )

#for the wordcloud, we have to join all the text data into a single string
text = " ".join(df['tweet_text'])

#generate the WC
wc.generate(text)

#plot the WC
plt.figure(figsize=(8,8))
plt.axis('off')
plt.imshow(wc)
plt.title('Trending topic on Twitter',fontdict={'fontsize':20})
plt.show()
```

Trending topic on Twitter

FreqDist

Let's use `nltk's FreqDist` class to get some numbers to gives us more clarity:

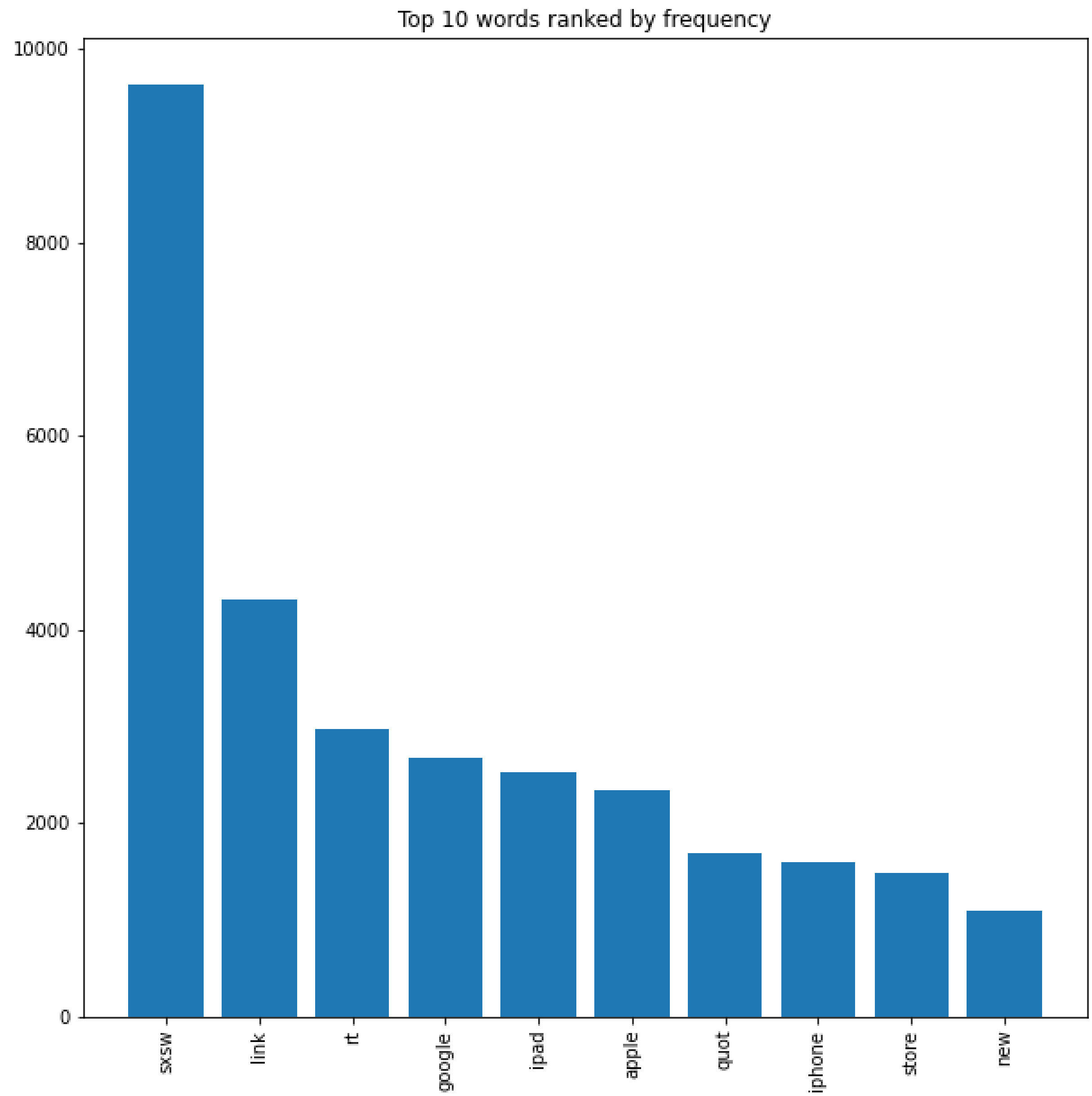
Removing *stopwords*

Words like `a,as,the,for,that etc.` are used to make the sentence grammatically correct but give very little information about the contents of the text itself. Since they are most likely to occur a lot more than nouns and adjectives, they also distort our analysis and are hence best removed from the corpus for analysis.

In [11]:
```
#defining a function to remove stopwords
def remove_stopwords(token_list):
    stopwords_removed = [token for token in token_list if token not in stopwords_list]
    return stopwords_removed

#applying the function to the text_token column
df['text_token'] = df['text_token'].apply(remove_stopwords)
```

In [12]:
```
#defining a function to plot the top_25 most occuring words

def plot_freq_dist(words):
    freq_dist = FreqDist(df[words].explode())
    # listing out the top 25 most occuring words and ther respective counts
    top_10 = list(zip(*freq_dist.most_common(10)))

    #creating a plot of the top_25 words
    fig,ax=plt.subplots(figsize=(10,10))
    ax.bar(top_10[0],top_10[1])
    ax.set_title('Top 10 words ranked by frequency')
    ax.tick_params(axis='x', rotation=90)
```

In [13]:
```
#plotting the freq_dist
plot_freq_dist('text_token')
```



We can see from the above that the words `SXSW,Google,iPad` are some of the most tweeted words. A google search of `SXSW` reveals it to be arts and music festival held in Austin,TX. Hence, we can reasonably conclude that tweets collected for the analysis was from the city of Austin,TX and also coincided when the festival was running. It is also quite possible that people were streaming it on their iPads with great success!

What is the most popular product?

This is to answer the first question : What are the emotional repsonses for each product_service category listed in the data? For eg: for the category 'Apple' how many positive,negative and neutral responses are there?

By comparing the responses for each category, we can gage customer sentiment

Pivot Tables can help better organize the data for the analysis

In [14]:
```
#creating a pivot table to organize the data
```

```python
df_pivot = df.pivot_table(index='product_service',aggfunc='count',columns='emotion')
df_pivot
```

Out[14]:

	text_token				tweet_text			
emotion	**I can't tell**	**Negative emotion**	**No emotion toward brand or product**	**Positive emotion**	**I can't tell**	**Negative emotion**	**No emotion toward brand or product**	**Positive emotion**
product_service								
Android	NaN	16.0	2.0	141.0	NaN	16.0	2.0	141.0
Apple	2.0	97.0	22.0	575.0	2.0	97.0	22.0	575.0
Google	2.0	115.0	24.0	582.0	2.0	115.0	24.0	582.0
Not Applicable	147.0	51.0	5297.0	306.0	147.0	51.0	5297.0	306.0
iPad	4.0	125.0	24.0	793.0	4.0	125.0	24.0	793.0
iPad or iPhone App	NaN	63.0	10.0	397.0	NaN	63.0	10.0	397.0
iPhone	1.0	103.0	9.0	184.0	1.0	103.0	9.0	184.0

In [15]:
```python
df_pivot.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, Android to iPhone
Data columns (total 8 columns):
 #   Column                                           Non-Null Count  Dtype  
---  ------                                           --------------  -----  
 0   (text_token, I can't tell)                       5 non-null      float64
 1   (text_token, Negative emotion)                   7 non-null      float64
 2   (text_token, No emotion toward brand or product) 7 non-null      float64
 3   (text_token, Positive emotion)                   7 non-null      float64
 4   (tweet_text, I can't tell)                       5 non-null      float64
 5   (tweet_text, Negative emotion)                   7 non-null      float64
 6   (tweet_text, No emotion toward brand or product) 7 non-null      float64
 7   (tweet_text, Positive emotion)                   7 non-null      float64
dtypes: float64(8)
memory usage: 504.0+ bytes
```

Since the `text_column` is not relevant right now, let's remove it:

In [16]:
```python
#dropping the columns
df_pivot.drop(df_pivot.columns[[0,1,2,3]],axis=1,inplace=True)
```

Renaming the columns for better understanding:

In [17]:
```python
#renaming the columns
df_pivot.columns = ["I can't tell",'Negative emotion','No emotion toward brand or produ
df_pivot
```

Out[17]:

	I can't tell	**Negative emotion**	**No emotion toward brand or product**	**Positive emotion**
product_service				
Android	NaN	16.0	2.0	141.0

product_service	I can't tell	Negative emotion	No emotion toward brand or product	Positive emotion
Apple	2.0	97.0	22.0	575.0
Google	2.0	115.0	24.0	582.0
Not Applicable	147.0	51.0	5297.0	306.0
iPad	4.0	125.0	24.0	793.0
iPad or iPhone App	NaN	63.0	10.0	397.0
iPhone	1.0	103.0	9.0	184.0

In [18]:

```
#dropping 'Not Applicable' since it is not relevant here
df_pivot.drop('Not Applicable',axis=0,inplace=True)

#rearranging the columns for better visualization
df_pivot=df_pivot[['Positive emotion','Negative emotion', 'No emotion toward brand or p

#sorting the values for better visuzalization
df_pivot.sort_values('Positive emotion',ascending=False,inplace=True)

df_pivot
```

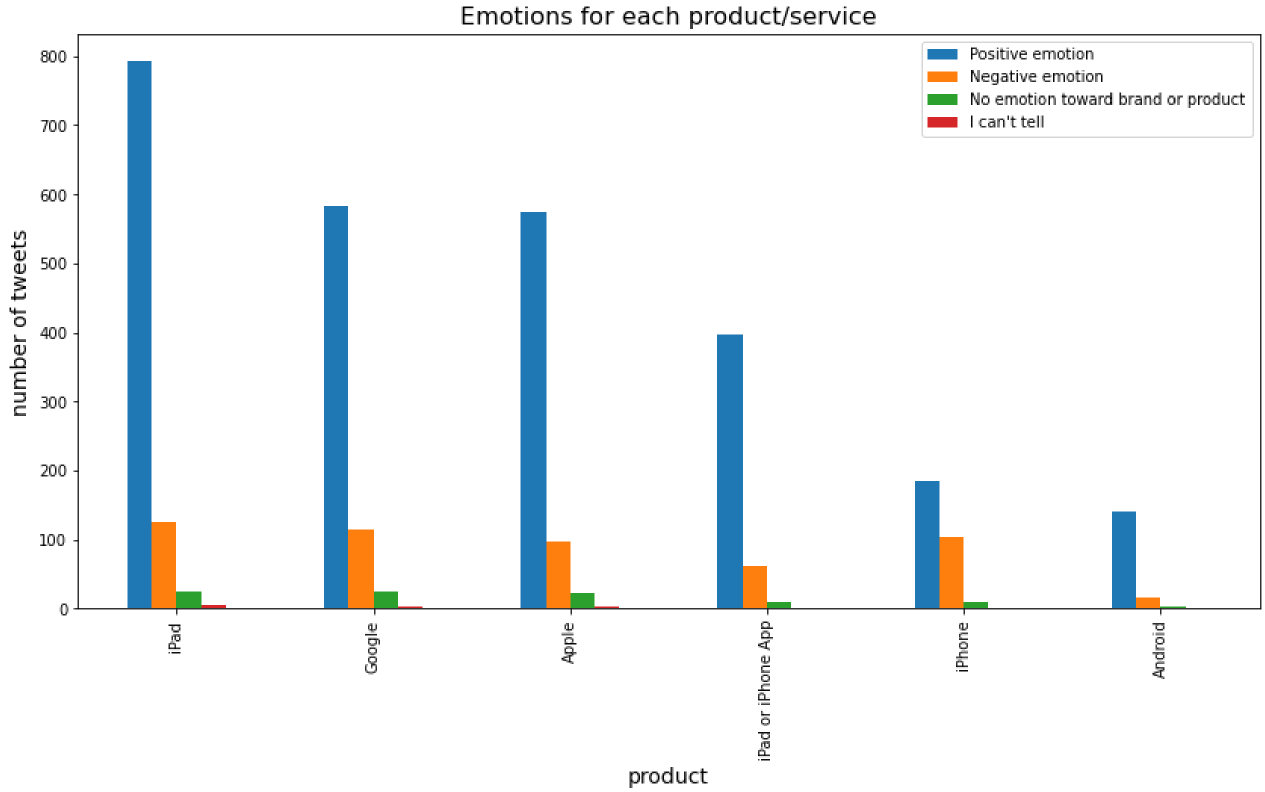
Out[18]:

product_service	Positive emotion	Negative emotion	No emotion toward brand or product	I can't tell
iPad	793.0	125.0	24.0	4.0
Google	582.0	115.0	24.0	2.0
Apple	575.0	97.0	22.0	2.0
iPad or iPhone App	397.0	63.0	10.0	NaN
iPhone	184.0	103.0	9.0	1.0
Android	141.0	16.0	2.0	NaN

Now that we've formatted the table to our liking, let's plot a bar chart see the different emotions for each product. This will give us an idea about how customers feel about each product

In [19]:

```
# bar chart listing emotion class for each product_service

df_pivot.plot(kind='bar',figsize=(14,7));
plt.title('Emotions for each product/service',fontdict={'fontsize':16});
plt.ylabel('number of tweets',fontsize=14);
plt.xlabel('product',fontsize=14);
```



We have a clear winner in ***iPad!*** i.e. the iPad is the most popular product among customers and Android the least. We can also see that the number of negative tweets seem to be somewhat level across all products except for Android.

What do customers like/dislike in a product?

Here, we are looking to answer the second question.We can do this by breaking down for each product,the different emotions to see if there are any key words that stand out. For eg: we can list out tweets by positive and negative emotions for iPad and analyze separately to gage sentiment.

In [20]:
```
#updating stopwords list to include SXSW since it appears nearly 10,000 times
stopwords_list.append('SXSW')
```

Let's define some functions to make things easier:

Function to get only ***positive tweets***

In [21]:
```
def get_positive(df,category,emotion):
    positive_df = df.loc[(df['product_service']==category) & (df['emotion']=='Positive
    return positive_df
```

Function to get only ***negative tweets***

In [22]:
```
def get_negative(df,category,emotion):
    negative_df=df.loc[(df['product_service']==category) & (df['emotion']=='Negative em
    return negative_df
```

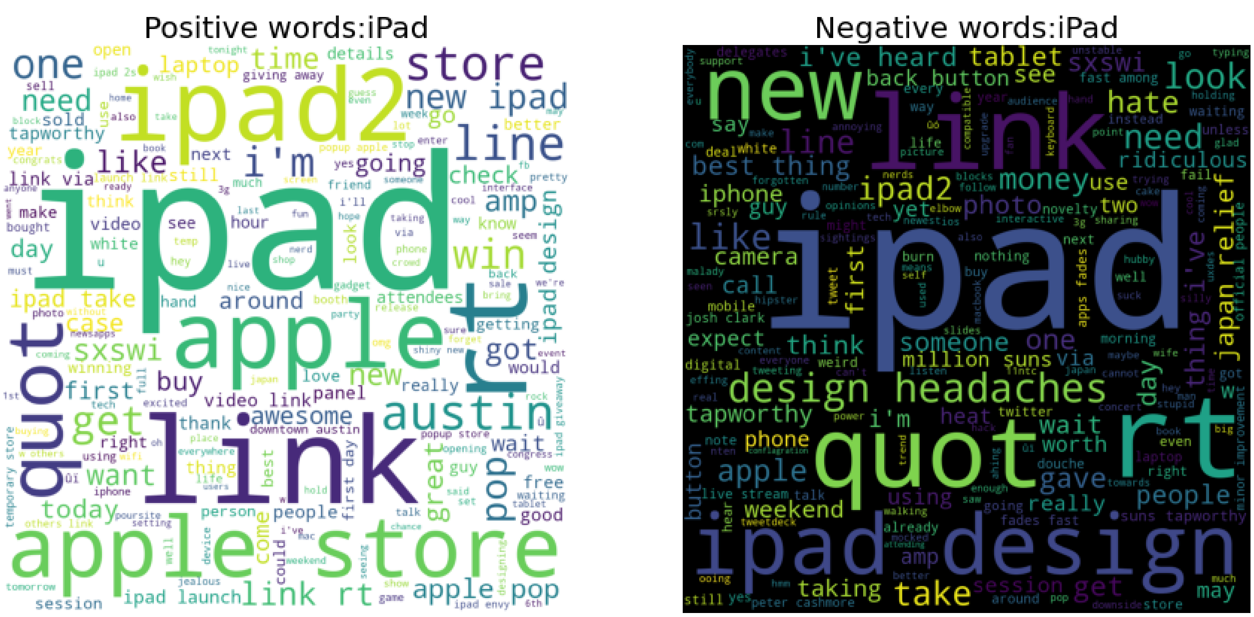
Function to generate ***Wordclouds for positive and negative tweets***

In [23]:
```
def wordcloud_gen(df,category):
```

```python
    pos=get_positive(df,category,'Positive emotion')
    neg=get_negative(df,category,'Negative emotion')
    product = category # for use in the heading of the wordcloud

    #instantiate wordclouds
    wc1 = WordCloud(background_color='white',   #positive wc
                 stopwords=stopwords_list,
                 height=1000,
                 width=1000,
                 )
    wc2 = WordCloud(background_color='black',   #negative wc
                 stopwords=stopwords_list,
                 height=1000,
                 width=1000,
                 )

#for the wordcloud, we have to join all the text data into a single string
    text1 = " ".join(pos['tweet_text']) #positive
    text2 = " ".join(neg['tweet_text']) #negative

#generate the WC
    wc1.generate(text1) #positive
    wc2.generate(text2) #negative

#plot the WC
    fig,(ax1,ax2) = plt.subplots(1,2,figsize=(15,10))
    font = {'fontsize':20}
    ax1.imshow(wc1) #positve
    ax1.axis('off')
    ax1.set_title(f'Positive words:{product}',fontdict=font);
    ax2.imshow(wc2) #negative
    ax2.axis('off')
    ax2.set_title(f'Negative words:{product}',fontdict=font);
```

Ipad

In [24]:
```python
wordcloud_gen(df,'iPad')
```



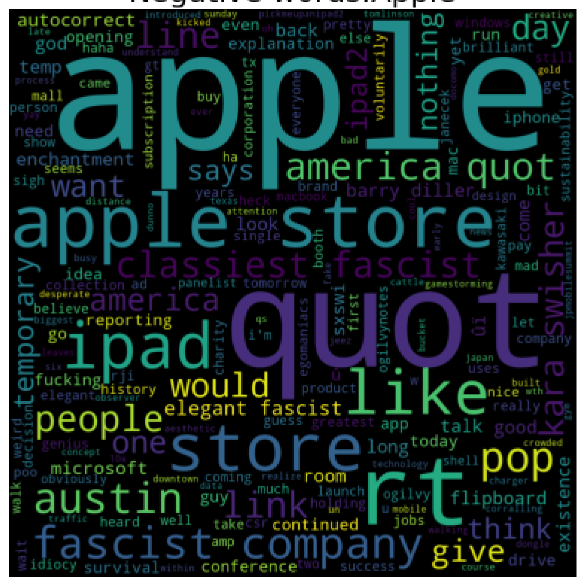
From the negative emotions, we can see `design headaches, iPad design,money,back` `button` are some of the words that feature prominently thus illustrating displeasure of the users regarding some of the features of the `iPad`. `iPad2` is also mentioned quite a lot.

Apple

In [25]:
```
wordcloud_gen(df,'Apple')
```

Positive words:Apple

Negative words:Apple



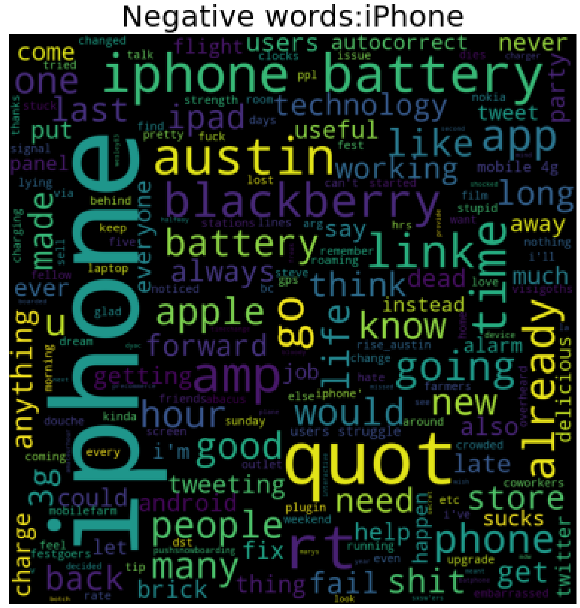
Sentiment against Apple seems to be quite severe given the high number of tweets featuring the word `fascist`!

iPhone

In [26]:
```
wordcloud_gen(df,'iPhone')
```

Positive words:iPhone

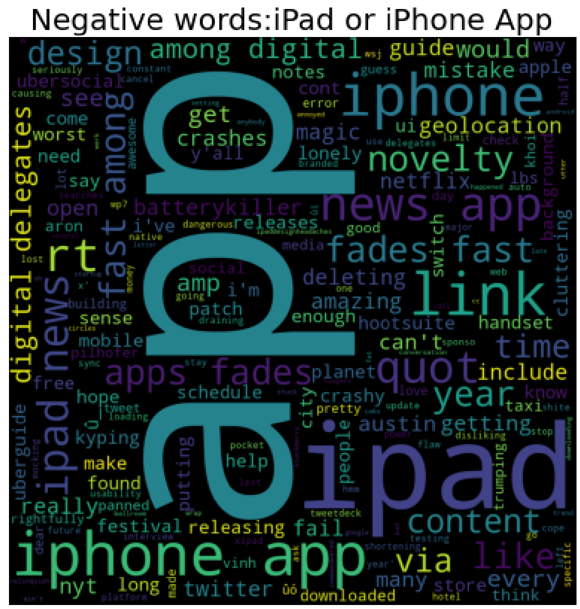
Negative words:iPhone



From positive emotions, `verizon` stands out suggesting their superiority from the other carriers. `iphone battery,battery` from the negative emotions illustrate unequivocally where the problem lies with the iPhone.

Ipad and iPhone apps

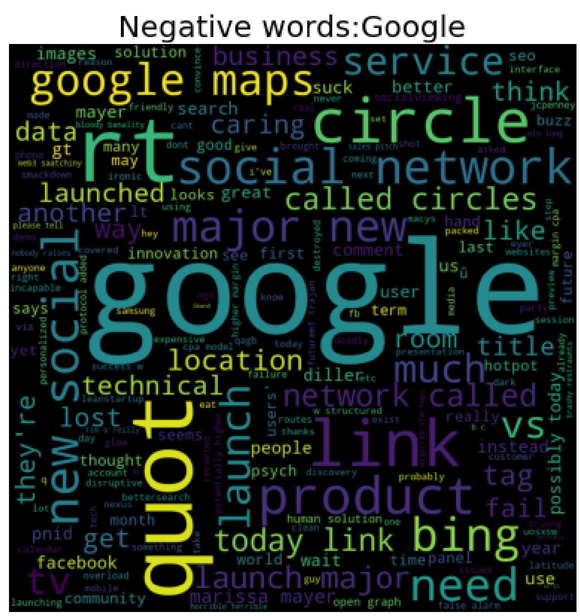
In [27]:
```
wordcloud_gen(df,'iPad or iPhone App')
```



Nothing really stands out here

Google

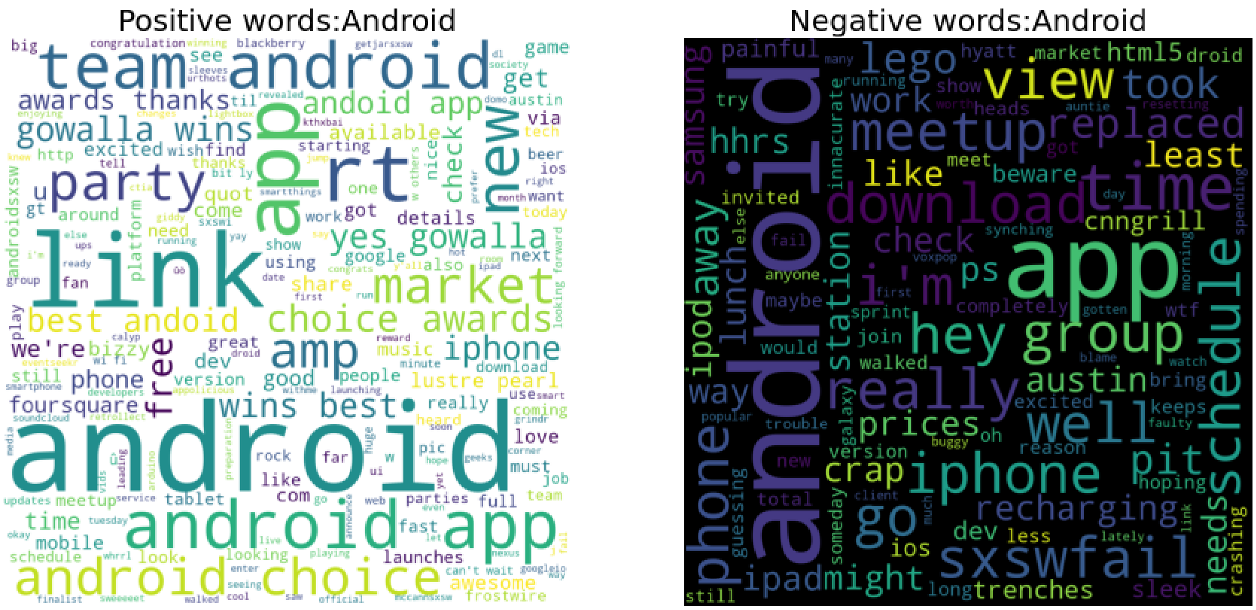
In [28]:
```
wordcloud_gen(df,'Google')
```



`google maps` seems to be equally represented in both positive and negative tweets. Some people like it and some don't. Same thing with `social network`. How that ties in with `google` needs some more exploration.

Android

In [29]:
```
wordcloud_gen(df,'Android')
```



The words `awards,wins best` from the positve tweets maybe point towards someone from the festival winning or an app on the Android platform winning some award! `Samsung` pops up in negative tweets suggesting issues with apps running on Samsung phones.

Model to predict company from tweets

Since we're only concerned about which company's is favoured by customers let's futher group the products as Apple and Google.

In [30]:
```
#let's group product/services that resemble each other for both brands. This will make 

df['product_service'].replace('Android','Google',inplace=True)
df['product_service'].replace(['iPhone','iPad','iPad or iPhone App'],'Apple',inplace=Tr

#checking
df['product_service'].value_counts()
```

Out[30]:
```
Not Applicable    5801
Apple             2409
Google             882
Name: product_service, dtype: int64
```

In [31]:
```
# creating target values for product_service
new_map = {'Not Applicable':0,
          'Apple':1,
          'Google':2}

df['target'] = df['product_service'].map(new_map)
df.head()
```

Out[31]:

	tweet_text	product_service	emotion	text_token	target

	tweet_text	product_service	emotion	text_token	target
0	.@wesley83 i have a 3g iphone. after 3 hrs twe...	Apple	Negative emotion	[wesley83, 3g, iphone, hrs, tweeting, rise_aus...	1
1	@jessedee know about @fludapp ? awesome ipad/i...	Apple	Positive emotion	[jessedee, know, fludapp, awesome, ipad, iphon...	1
2	@swonderlin can not wait for #ipad 2 also. the...	Apple	Positive emotion	[swonderlin, wait, ipad, also, sale, sxsw]	1
3	@sxsw i hope this year's festival isn't as cra...	Apple	Negative emotion	[sxsw, hope, year, festival, crashy, year, iph...	1
4	@sxtxstate great stuff on fri #sxsw: marissa m...	Google	Positive emotion	[sxtxstate, great, stuff, fri, sxsw, marissa, ...	2

Vectorizers

To be able to apply ML models to text data, we must first convert them into a numeric form.

This is accomplished by using *Vectorizers*. *Vectorizers* convert each word in the corpus into a feature and create vectors for each. There are different vectorizers and here, we will use the following three:

1. CountVectorizer
2. Tf-IDF Vectorizer
3. Word2vec Vectorizer

CountVectorizer

CountVectorizer builds on the *Bag Of Words* concept. All the words in the corpus are taken and their frequencies are calculated. The output of the CountVectorizer is a sparse matrix where each feauture is a word and the column is the vector of it's frequencies in each document.

In [32]:
```
#setting up X,y train and test sets
X= df['text_token']
y = df['target']

X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=123)
```

Since we have already pre-processed out text data, we have to circumvent *CountVectorizer's* `preprocessing and tokenizing` parameters. We do this by creating a dummy function:

In [33]:
```
def dummy(doc):
    return doc
```

Let's build baseline models using `LogisticRegression,Naive-Bayes,Random Forest and SVM`. We can build pipelines for each model and calculate the f1-score for each by creating a loop.

In [34]:
```
# building a pipeline of LogisticRegression, Naive-Bayes, SVM and RandomForest models

pipe_lr = Pipeline([('vectrorizer',CountVectorizer(stop_words=stopwords_list,preprocess
                    ('model',LogisticRegression(random_state=123,solver='liblinear'))
                   ])

pipe_nb = Pipeline([('vectrorizer',CountVectorizer(stop_words=stopwords_list,preprocess
```

```
                    ('model',MultinomialNB())
                    ])

pipe_rf = Pipeline([('vectrorizer',CountVectorizer(stop_words=stopwords_list,preprocess
                    ('model',RandomForestClassifier(random_state=123))
                    ])

pipe_svm = Pipeline([('vectrorizer',CountVectorizer(stop_words=stopwords_list,preproces
                    ('model',svm.SVC(random_state=123))
                    ])

#setting up names for the classification report
names_dict = dict(df['product_service'].value_counts())
names = [name for name in names_dict]

#build a list of tuples to build a df
models=['LogReg', 'MultiNB', 'RForest', 'SVM']
f1 = []

#fitting the models on the train sets
pipelines = [pipe_lr,pipe_nb,pipe_rf,pipe_svm]

for pipe in pipelines:
    pipe.fit(X_train,y_train)
    predictions = pipe.predict((X_test))
#     print(pipe)
#     print(classification_report(y_test,predictions,target_names=names))
    f1.append(f1_score(y_test,predictions,average='macro'))

#building a df of the f1_scores
scores = list(zip(models,f1))
scores_df = pd.DataFrame(data=scores,columns=['model','f1_score_cv'])
```

Tf-IDF Vectorizer

Tfidf Vectorizer takes into acccount the relative importance of the word to the corpus. It combines *term frequency* and _inverse_document*frequency*. It calculates how often a word occurs in a document (term frequency) and also how many documents contain the word(inverse document frequency). The output is again a sparse matrix like with CountVectorizer.

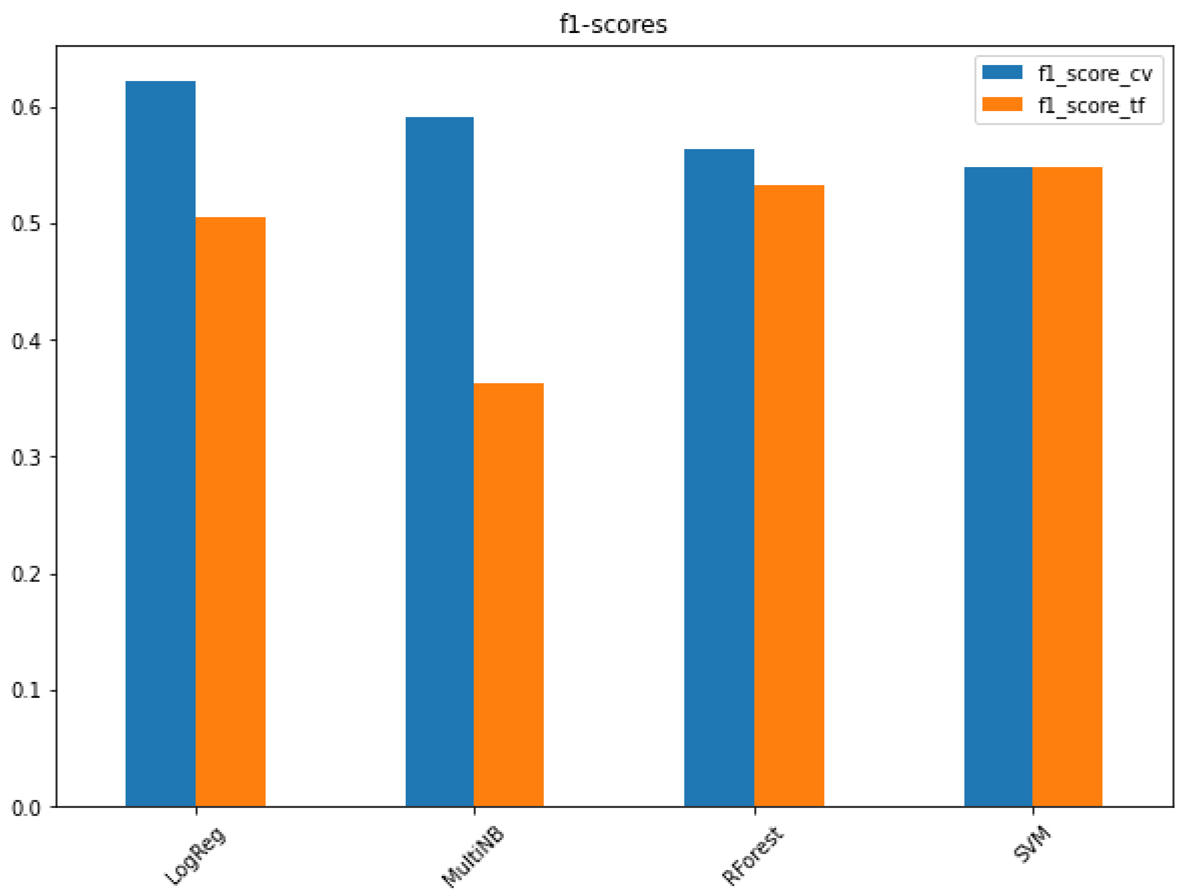
In [35]:

```
# repeating the same processes as above
pipe_lr = Pipeline([('vectrorizer',TfidfVectorizer(stop_words=stopwords_list,preprocess
                    ('model',LogisticRegression(random_state=123,solver='liblinear'))
                   ])

pipe_nb = Pipeline([('vectrorizer',TfidfVectorizer(stop_words=stopwords_list,preprocess
                    ('model',MultinomialNB())
                    ])

pipe_rf = Pipeline([('vectrorizer',TfidfVectorizer(stop_words=stopwords_list,preprocess
                    ('model',RandomForestClassifier(random_state=123))
                    ])

pipe_svm = Pipeline([('vectrorizer',TfidfVectorizer(stop_words=stopwords_list,preproces
                    ('model',svm.SVC(random_state=123))
                    ])

#setting up names for the classification report
```

```
names_dict = dict(df['product_service'].value_counts())
names = [name for name in names_dict]

#build a list of tuples to build a df
models=['LogReg', 'MultiNB', 'RForest', 'SVM']
f1 = []

#fitting the models on the train sets
pipelines = [pipe_lr,pipe_nb,pipe_rf,pipe_svm]

for pipe in pipelines:
    pipe.fit(X_train,y_train)
    predictions = pipe.predict((X_test))
#     print(pipe)
#     print(classification_report(y_test,predictions,target_names=names))
    f1.append(f1_score(y_test,predictions,average='macro'))

#adding the tf f1-scores to the scores_df
scores_df['f1_score_tf'] = f1
```

Now that we've run models with 2 CV and Tfidf vectorizers, let's visualize the f1-scores of each

In [36]:
```
#visualizing the f1-scores of all the models for the two vectorizers

fig,ax = plt.subplots(figsize=(10,7))
scores_df.plot(kind='bar',ax=ax);
ax.set_xticklabels(models,rotation=45);
ax.set_title('f1-scores');
```



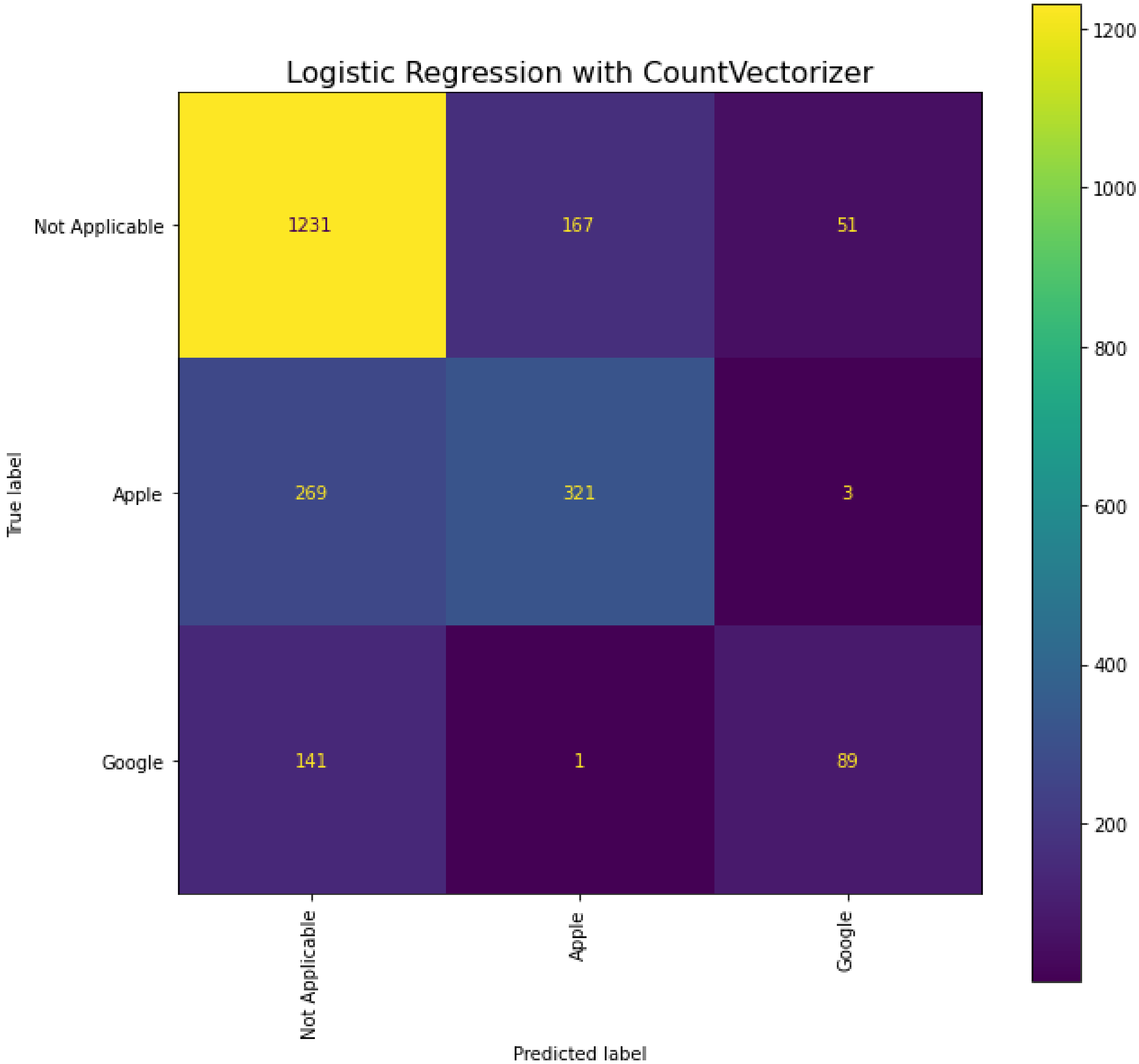
Since the LogisticRegression model with the CountVectorizer has the highest f1-score among all models, let's use that for optimizations

Tuning LogisticRegression with Countvectorizer

Let's get a closer look at the LR with CV model:

In [37]:
```
lr_pipe = Pipeline([('vectrorizer',CountVectorizer(stop_words=stopwords_list,preprocess
                    ('model',LogisticRegression(random_state=123,solver='liblinear'))
                   ])

lr_pipe.fit(X_train,y_train)
predictions=lr_pipe.predict(X_test)
print(classification_report(y_test,predictions,target_names=names))
fig,ax=plt.subplots(figsize=(10,10))
plot_confusion_matrix(lr_pipe,X_test,y_test,display_labels=names,ax=ax,xticks_rotation=
ax.set_title('Logistic Regression with CountVectorizer',fontdict={'fontsize':16});
```

```
                precision    recall  f1-score   support

Not Applicable       0.75      0.85      0.80      1449
         Apple       0.66      0.54      0.59       593
        Google       0.62      0.39      0.48       231

      accuracy                           0.72      2273
     macro avg       0.68      0.59      0.62      2273
  weighted avg       0.71      0.72      0.71      2273
```



From the above, we can see that the model does relatively well in the `Not Applicable` class compared to `Apple` and `Google`. This is unsurprising given the imbalance in the data.

min_df and max_df values

We can try tuning some of the hyperparameters of the model to improve performance. `min_df` and `max_df` are two that we can look. min_df removes words that appear rarely. Since they are rare, it possible that they will not provide a lot of information. max_df is the opposite of min_df. If the words are too frequent, chances are they too do not provide a lot of information. By creating a range for each parameter, we can run a loop to see if model performance improves

In [38]:

```
# setting a range for min_df and max_df
min_df_value = np.arange(1,5) # words than appear less than the min_df value in all doc
max_df_value = np.arange(1500,1505) # words than appear more than the max_df value in a

#initiating lists to use for plotting
f_score = []
min_value=[]
max_value=[]
```

```
#setting up the loop for min_df and max_df values
for i in min_df_value:
    for j in max_df_value:
        min_value.append(i)
        max_value.append(j)
        #instantiate pipeline
        new_pipe = Pipeline([('vectrorizer',CountVectorizer(stop_words=stopwords_list,m
                            ('model',LogisticRegression(random_state=123,solver=
                            ])
        new_pipe.fit(X_train,y_train)
        preds = new_pipe.predict(X_test)

        #getting f1 score
        score = round(f1_score(y_test,preds,average='macro'),3)
        f_score.append(score)
#         print(f'min_df = {i}, max_df = {j}, f1_score={score}')

#visualizing the accuracy score for the different combinations
d = list(zip(min_value,max_value))
fig,ax=plt.subplots(figsize=(15,5))
plt.tick_params(bottom=False)
ax.plot(f_score,marker='o',markerfacecolor='r',ls='--');
ax.set_xticklabels([],[]);
ax.set_title('F1 Score of the Logistic Regression Model with CountVectorizer');
```

Clearly, we've made the model worse. Our initial f1-score was 0.62 but here we're maxed out at 0.574

n-gram

The idea behind n-grams is that sometimes word pairings or short phrases are better. For eg: 'black sheep' is more informative than 'black' and 'sheep' seperately

In [39]:
```
lr_pipe = Pipeline([('vectrorizer',CountVectorizer(stop_words=stopwords_list,preprocess
                                                ngram_range=(1,2),min_df=1,max_df=200
                     ('model',LogisticRegression(random_state=123,solver='liblinear'))
                    ])

lr_pipe.fit(X_train,y_train)
predictions=lr_pipe.predict(X_test)
print(classification_report(y_test,predictions,target_names=names))
# fig,ax=plt.subplots(figsize=(10,10))
# plot_confusion_matrix(lr_pipe,X_test,y_test,display_labels=names,ax=ax,xticks_rotatio
```

```
                  precision    recall  f1-score   support

Not Applicable       0.75      0.86      0.80      1449
         Apple       0.67      0.54      0.60       593
        Google       0.63      0.35      0.45       231

      accuracy                           0.73      2273
     macro avg       0.68      0.59      0.62      2273
  weighted avg       0.72      0.73      0.71      2273
```

We can see that there is no discernible change in model performance

Stemming using PorterStemmer

With stemming, we use the use root of the word. For eg: ran,runs,running all stem from the word run. This way we reduce the number of features and can improve accuracy of the model.

In [40]:
```
#initializing the stemmer
ps=PorterStemmer()

#creating a function to tokenize and stem the tokens
def stem_and_tokenize(document):
    tokens = tokenizer.tokenize(document)
    return [ps.stem(token) for token in tokens]
```

In [41]:
```
df['stemmed_tokens'] = df['tweet_text'].apply(stem_and_tokenize)
df.head()
```

Out[41]:

	tweet_text	product_service	emotion	text_token	target	stemmed_tokens
0	.@wesley83 i have a 3g iphone. after 3 hrs twe...	Apple	Negative emotion	[wesley83, 3g, iphone, hrs, tweeting, rise_aus...	1	[wesley83, have, 3g, iphon, after, hr, tweet, ...
1	@jessedee know about @fludapp ? awesome ipad/i...	Apple	Positive emotion	[jessedee, know, fludapp, awesome, ipad, iphon...	1	[jessede, know, about, fludapp, awesom, ipad, ...
2	@swonderlin can not wait for #ipad 2 also. the...	Apple	Positive emotion	[swonderlin, wait, ipad, also, sale, sxsw]	1	[swonderlin, can, not, wait, for, ipad, also, ...
3	@sxsw i hope this year's festival isn't as cra...	Apple	Negative emotion	[sxsw, hope, year, festival, crashy, year, iph...	1	[sxsw, hope, thi, year, festiv, isn, as, crash...
4	@sxtxstate great stuff on fri #sxsw: marissa m...	Google	Positive emotion	[sxtxstate, great, stuff, fri, sxsw, marissa, ...	2	[sxtxstate, great, stuff, on, fri, sxsw, maris...

In [42]:
```
#running logistic regression on the stemmed tokens
#re-defining X and y
X = df['stemmed_tokens']
y = df['target']

X_train2,X_test2,y_train2,y_test2 = train_test_split(X,y,random_state=123)
```

In [43]:
```
lr_pipe = Pipeline([('vectrorizer',CountVectorizer(stop_words=stopwords_list,preprocess
```

```
                                        ngram_range=(1,2),min_df=1,max_df=200
                    ('model',LogisticRegression(random_state=123,solver='liblinear'))
                   ])

lr_pipe.fit(X_train2,y_train2)
predictions=lr_pipe.predict(X_test2)
print(classification_report(y_test2,predictions,target_names=names))
```

```
                precision    recall  f1-score   support

Not Applicable       0.75      0.85      0.80      1449
         Apple       0.66      0.55      0.60       593
        Google       0.61      0.37      0.46       231

      accuracy                           0.72      2273
     macro avg       0.67      0.59      0.62      2273
  weighted avg       0.71      0.72      0.71      2273
```

Word2Vec

Word2vec is another vectorization method and falls under the category called ***Word Embeddings***. It is essentially a neural network with an i/p layer, hidden layer and an o/p layer.The vectors are created in an ***embedding space*** and are used to capture the semantic relationships between words.

Here, we will import the Word2vec vector from the open source ***gensim*** library and use the ***skip gram*** architecture for modelling. The gensim library has vectors built in that we will use to base our model off of.

In [44]:
```
#instantiate the vect
model = Word2Vec(df['text_token'], vector_size=100, window=2, min_count=5,sg=1)

#train the model
model.train(df['text_token'],epochs=15,total_examples=model.corpus_count)
```

Out[44]:
```
(955088, 1574070)
```

Experimentation

The calculated vectors are stored in the `Word2VecKeyedVectors` instance stored in the `wv` attribute . Let's assign it to a different variable to save ourselves from lot's of keystrokes

In [45]:
```
wv=model.wv
```

Checking the vector for the word `battery` . This will display the weights that the model has calculated for the context that the word 'battery' will most likely used in

In [46]:
```
wv.most_similar('battery')
```

Out[46]:
```
[('double', 0.7678278684616089),
 ('backup', 0.7578180432319641),
 ('realized', 0.7568727135658264),
 ('brightness', 0.7554357647895813),
 ('charge', 0.7329474091529846),
 ('woke', 0.7273014783859253),
 ('extended', 0.7197436690330505),
 ('outlet', 0.7176432013511658),
```

```
 ('fully', 0.7125301361083984),
 ('half', 0.7111120820045471)]
```

In [47]:
```python
#the vector assocaited with the word 'battery'
wv['battery']
```

Out[47]:
```
array([-0.3852477 ,  0.1473897 ,  0.5075781 , -0.25849283,  0.36183384,
       -0.21421807,  0.1885854 ,  0.3600012 ,  0.00693324, -0.08847217,
       -0.43788448, -0.57707196, -0.13088648, -0.42850077,  0.31702712,
       -0.4826914 , -0.18213949, -0.47767192,  0.01602536, -0.5687668 ,
        0.10026983,  0.25245446,  0.48191145,  0.21089724, -0.13071814,
        0.35413998, -0.15259014, -0.20684358, -0.27916506,  0.06934378,
       -0.48606586,  0.25709513, -0.1984691 , -0.06171742,  0.24166107,
        0.01368475, -0.07399097, -0.41492984, -0.30766377, -0.28126   ,
       -0.14304595, -0.10936823, -0.06913579, -0.05304497,  0.47227603,
       -0.37170777, -0.21918008,  0.29798314, -0.36031303,  0.6944351 ,
       -0.02098559, -0.3272064 ,  0.16261105,  0.32140142, -0.5049191 ,
        0.09554312, -0.40757477,  0.03676416, -0.04204569, -0.09226754,
       -0.2619729 , -0.01748402,  0.04951853, -0.18803808, -0.36884603,
        0.1809068 , -0.3095469 ,  0.2794834 , -0.5456628 ,  0.3923531 ,
        0.2789598 ,  0.60674256,  0.5209727 , -0.27911094, -0.70896274,
        0.3250257 , -0.05716193, -0.81106555, -0.70417464, -0.2530832 ,
       -0.2658978 ,  0.02266886, -0.25652817,  0.5495951 , -0.03091053,
        0.42857042,  0.22563103,  0.06470291,  0.18618639, -0.08224505,
       -0.16818136, -0.24138579, -0.09538174,  0.0771694 ,  0.24346162,
        0.3925721 ,  0.04084785, -0.37567925,  0.24178706, -0.22596325],
      dtype=float32)
```

In [48]:
```python
#getting the list of words from the model
words = list(model.wv.index_to_key)
words[0:10] #looking at the first 10 words
```

Out[48]:
```
['sxsw',
 'link',
 'rt',
 'google',
 'ipad',
 'apple',
 'quot',
 'iphone',
 'store',
 'new']
```

In [49]:
```python
#getting the vectors assocated with each of those words
vector_list = [model.wv[word] for word in words]
vector_list[0] #examining the vector for the first word
```

Out[49]:
```
array([ 0.2939249 ,  0.3600829 ,  0.20681262, -0.0196951 ,  0.10259214,
       -0.35870022, -0.33109558,  0.20005769, -0.8791161 , -0.27900603,
        0.14658761, -0.13400513,  0.49446723,  0.16662355, -0.36503434,
       -0.07683668, -0.29224557,  0.20807473, -0.2085533 , -0.45254752,
        0.5781813 ,  0.58960783,  0.4815622 , -0.31043407,  0.13748093,
        0.21435413, -0.01618872,  0.05426585, -0.18348311, -0.11867385,
       -0.00741075, -0.25643295,  0.4731644 , -0.44306275,  0.21205485,
       -0.4835446 ,  0.5060575 , -0.07585768,  0.01181712, -0.22027975,
       -0.30940512,  0.16075183, -0.17455734,  0.03791101,  0.01207904,
       -0.05832497, -0.20840485,  0.54333454,  0.0704006 ,  0.45237488,
       -0.24760732,  0.10067454, -0.32775736,  0.09242272,  0.25574067,
        0.45016247,  0.07527873, -0.40016896, -0.24950868, -0.1579671 ,
       -0.3351678 , -0.22073431,  0.09135729, -0.3205818 ,  0.0790761 ,
        0.1704996 , -0.10673695,  0.29617006,  0.03980981,  0.01191649,
        0.5617382 ,  0.0826259 , -0.14490795, -0.3323023 ,  0.5682849 ,
        0.1029357 , -0.19387892,  0.12534112, -0.13610382, -0.19519174,
       -0.09571896,  0.37726015,  0.4392167 ,  0.20762801, -0.29639816,
```

```
 0.08691575,  0.23130304,  0.04967168, -0.31996852,  0.2656154 ,
-0.35869262, -0.26813707,  0.30323657,  0.13089976,  0.30091673,
-0.29393902,  0.21009995,  0.07233847, -0.11021888,  0.17519419],
dtype=float32)
```

In [50]:

```
#creating a df
word_vect_zip = dict(zip(words,vector_list))
word_vect_df = pd.DataFrame(word_vect_zip)
word_vect_df.head()
```

Out[50]:

	sxsw	link	rt	google	ipad	apple	quot	iphone	store	
0	0.293925	-0.030190	-0.072077	0.128399	-0.509867	-0.119943	0.307111	0.276807	-0.479090	-0.41
1	0.360083	0.510761	0.382146	0.898604	-0.233233	0.093957	0.402366	0.188934	0.462406	0.62
2	0.206813	0.359409	0.082197	-0.071371	0.490204	0.274466	0.241543	0.719241	0.003996	0.08
3	-0.019695	-0.038390	0.226925	-0.213785	0.039168	-0.195313	-0.237552	-0.025656	0.444969	-0.05
4	0.102592	0.106574	0.352603	0.510999	0.693884	-0.260912	0.190929	0.142170	0.672349	0.37

5 rows × 2383 columns

The df is used to illustrate the different words created by the model and their corresponding vectors.

Mean Embeddings

Now, we need to get a vector representation for each document to be able to apply a ML model. For this, we will take the mean of the vectors in each document for the words that are in the model vocabulary. Let's define a function to get the mean vector for each document:

In [51]:

```
#to get the vector representation of each document, you will define a fuction that will
#vector for each word in the document that is in the model's vocab,beacuse obviously yo
#vectors for words that are not there. Then you will take the average of the vectors an
#vector will the vector for that document

def doc_vector(token):
    vector_size=model.wv.vector_size #getting the size of the vectors created
    resultant_vector = np.zeros(vector_size) # initializing a vector of zeros of the sa
    ctr=1 #counter
    for w in token:
        if w in words:
            ctr += 1
            resultant_vector += wv[w]
    resultant_vector = resultant_vector/ctr
    return resultant_vector
```

In [52]:

```
#applying the function to the stemmed_tokens colums
df['vectors'] = df['stemmed_tokens'].apply(doc_vector)
df.head()
```

Out[52]:

	tweet_text	product_service	emotion	text_token	target	stemmed_tokens	vectors

	tweet_text	product_service	emotion	text_token	target	stemmed_tokens	vectors
0	.@wesley83 i have a 3g iphone. after 3 hrs twe...	Apple	Negative emotion	[wesley83, 3g, iphone, hrs, tweeting, rise_aus...	1	[wesley83, have, 3g, iphon, after, hr, tweet, ...	[-0.17503028895173753, 0.1573968646781785, 0.2...
1	@jessedee know about @fludapp ? awesome ipad/i...	Apple	Positive emotion	[jessedee, know, fludapp, awesome, ipad, iphon...	1	[jessede, know, about, fludapp, awesom, ipad, ...	[-0.3059240415692329, -0.0588498093187809, 0.4...
2	@swonderlin can not wait for #ipad 2 also. the...	Apple	Positive emotion	[swonderlin, wait, ipad, also, sale, sxsw]	1	[swonderlin, can, not, wait, for, ipad, also, ...	[-0.2785051167011261, 0.11736054221789043, 0.2...
3	@sxsw i hope this year's festival isn't as cra...	Apple	Negative emotion	[sxsw, hope, year, festival, crashy, year, iph...	1	[sxsw, hope, thi, year, festiv, isn, as, crash...	[-0.013888629951647349, 0.26941906980105806, 0...
4	@sxtxstate great stuff on fri #sxsw: marissa m...	Google	Positive emotion	[sxtxstate, great, stuff, fri, sxsw, marissa, ...	2	[sxtxstate, great, stuff, on, fri, sxsw, maris...	[-0.16928290682179586, 0.03712101437018386, -0...

Modelling

Now that we have a vecotrized representation of each document, we can apply different ML models and check performance

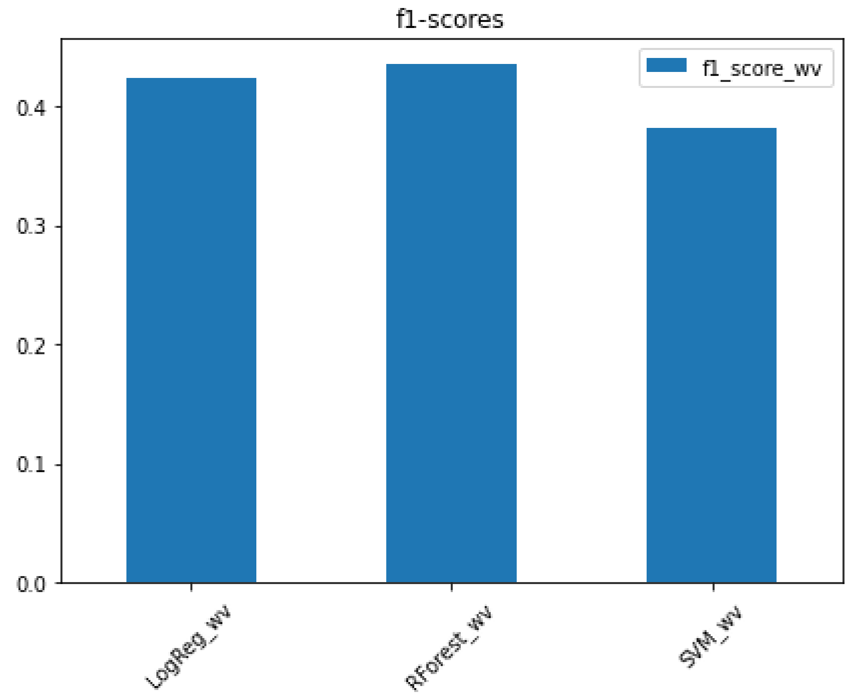
In [53]:
```
#redefining X&y
X= df['vectors'].to_list()
y = df['target'].to_list()

X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=123)
```

In [54]:
```
# repeating the same processes as above
pipe_lr_wv = Pipeline([('model',LogisticRegression(random_state=123,solver='liblinear')

# pipe_nb_wv = Pipeline([('model',MultinomialNB())])
#cannot use NB on negative values. Values will have to be normalized instead

pipe_rf_wv = Pipeline([('model',RandomForestClassifier(random_state=123))])

pipe_svm_wv = Pipeline([('model',svm.SVC(random_state=123))])

#build a list of tuples to build a df
models=['LogReg_wv', 'RForest_wv', 'SVM_wv']
f1 = []

#fitting the models on the train sets
```

```python
pipelines = [pipe_lr_wv,pipe_rf_wv,pipe_svm_wv]

for pipe in pipelines:
    pipe.fit(X_train,y_train)
    predictions = pipe.predict((X_test))
#     print(pipe)
#     print(classification_report(y_test,predictions,target_names=names))
    f1.append(f1_score(y_test,predictions,average='macro'))

#building a df of the f1_scores
scores_wv = list(zip(models,f1))
scores_df_wv = pd.DataFrame(data=scores_wv,columns=['model','f1_score_wv'])
```

In [55]:

```python
fig,ax = plt.subplots(figsize=(7,5))
scores_df_wv.plot(kind='bar',ax=ax);
ax.set_xticklabels(models,rotation=45);
ax.set_title('f1-scores');
```



As we can see, with Word2vec, the max f1-score that we can achieve is only 0.45, much less than our highest score of 0.62

Conclusions

Recommendations:

Since the iPad is the most popular product,Acme Online could look for opportunities to boost sales. Acme Online could also maybe expand their portfolio buy offering tablets from other manufacturers to see if they will bring in more revenue.

Modelling

1. Part-of-Speech tagging can be used to create more features.

2. Ensemble methods like XGBoost and Adaboost can also be trialed for modelling along with other word embedding techniques like fastText and Glove.

Limitations

More *varied* data is desirable. Current data is very imbalanced and localized to a time and place thus possibly skewing forecasts for the future. As next steps, that would be the starting point. We can then use it to improve model efficiency.