Research Assistant Tool using Consumer Oriented Sentiment Analysis

Module 3: Final Report

Course Name	Capstone Term I
Course Code	AIDI 1003
Course Facilitator	Dr. Uzair Ahmad
Students (Group 2)	Tejas Vyas (#100809103), Riddhi Thakkar (#100799412), Manu Sihag (#100801028), Gbemisola Banjoko (#100766479)
Date	01/10/2020

DURHAM COLLEGE SCHOOL OF BUSINESS, IT AND MANAGEMENT

Contents

Executive Summary

The capstone project's topic is "Research Assistant Tool using Consumer Oriented Sentiment Analysis". This project's importance arises from the limited availability of tools for consumers to research general attitudes about products despite overwhelming amounts of user sentiment data in form of reviews and social media posts.

The purpose of this project is to develop an intuitive solution by creating a web-app where users can enter a product query and get a general outlook of other consumers towards that product using aspect-based sentiment analysis.

The primary goals of the project are:

- Create an AI model capable of performing Sentiment Analysis in real-time for a given query against test data, and a web application which can support the AI model endpoints while being accessible and scalable for users.
- Allow users to research general sentiment about products in real-time using social media reviews.
- Decrease background research involved in purchase decisions, saving consumer time and resources.

Introduction

Customer Experience (CX) is the key to business success. In fact, 81% of marketers interviewed by Gartner said they expected their companies to compete mostly on the basis of CX in two years' time, making CX the new marketing battlefront.

Now, with the huge amount of products spread across the globe, while it's easier for a new consumer to find products, it's order of magnitude harder for them to research properly.

We as a team have several instances where we wanted to buy a product, and spent hours researching, and despite of the effort we ended up confused, not sure exactly how the other customers thought the product was working for them.

This was our core premise behind going forward with this project. Make research easier! By using machine learning, we can delegate the task over to an AI which:

- Stays up-to-date with review changes
- Keeps track of how perception of the product changes
- Can dig into thousands of reviews in limited time, while giving an objective output, generally unbiased based on the reviewer username or image or current state of exhaustion

Rationale Statement

Sentiment analysis is widely used by businesses today to research product marketing strategies by reviewing trends towards user attitude towards topics as well as to analyze product performance and requirements for future products.

During our research we found similar systems for this research are either expensive, or limited and focused towards business stakeholders - https://blog.hubspot.com/service/sentiment-analysis-tools. There are similar systems currently in existence, but they are limited to certain markets. For example, Steam, a game marketplace allows users to view user sentiment towards the game.

Our project aims to extend this strategy towards consumers, providing them a unique accessible web application at minimal cost which may be used to perform preliminary market research and assist a consumer with buying decisions using a simpler positive or negative scale.

Problem Statement

Social networks today have allowed businesses and brands to connect to millions of people. This allows companies to analyze market trends and user attitudes towards products and topics using the huge datasets available.

While this in-theory should ease consumer burden allowing them to easily compare products and make purchase decisions, there is a significant lack of consumer focused solutions which allow a user to easily search for products and buy new items, which causes a lot of wasted time and resources where a buyer has to go through review manually and determine the public sentiment about the product.

Sentiment Analysis through machine learning can assist consumers in gaining quick insights about a product and gauge opinion of other consumers, and thus requires an accessible and buyer centric application which can utilize it and allow significant savings in consumer time and resources.

Data Requirements

This data requirement for this project includes a dataset which contains:

- Comment rows to be analyzed for sentiment
    - Constraint: Sentences must be opinions, to allow sentiment analysis to classify them
    - Assumption: Comments must cover broad range of users and topics
- Label column in every row providing a sentiment which may be used to train the model
- Source column in every row providing the source location of the comment
- Generated Text column with numbers and punctuation removed for n-gram analysis
- Data must be originated from social media accounts including review websites
- Data must be split equally between positive and negative comments to minimize bias
- Sentences column must be user-generated, to allow proper training for the model
- Label column must be a bit, clearly distinguishing between negative and positive
- Comment column must be linked to a product, movie, food, or event for aspect classifier model training
- Comments must be limited to a specific language to discourage underfitting model
- Dataset must not be older than a decade
- Comments in the dataset must be limited in technical word usage

Data

During our research, we found an existing dataset from the Conference Paper - *Group to Individual Labels using Deep Features, Kotzias et. al,. KDD 2015.*

It contains sentences labelled with a positive or negative sentiment extracted from Amazon, IMDB and Yelp. The data can be accessed at the following link: https://www.kaggle.com/marklvl/sentiment-labelled-sentences-data-set

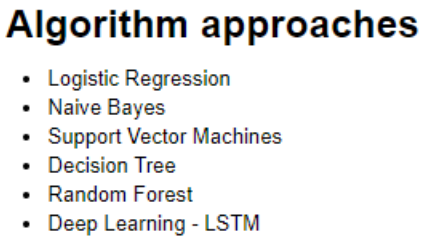
For each website, there exist 500 positive and 500 negative sentences. Those were selected randomly for larger datasets of reviews. We attempted to select sentences that have a clearly positive or negative connotation, the goal was for no neutral sentences to be selected.

Model/Architecture Approach

The project is divided into 2 components:

Machine Learning Model/Endpoint

For this component, we would be using the n-gram generated features to train an aspect-based sentiment classifier. Additionally,



- We would be using algorithms like Naïve Bayes, SVM, Logistic Regression, Decision Tree, Random Forest and LSTM and compare accuracy to use the best algorithm.
- The logic to preprocess data, machine learning model will be written in Python 3.0 using Pandas DataFrames. The model steps will be written in Jupyter notebook for easy review and analysis.
- The model endpoint will be hosted on a cloud system AWS/Azure.

Web Application

For this component, we would be creating a web application in HTML5 using MVC framework in C#.Net. Additionally,

- The web application will be linked with API endpoints for Social Media networks to gather a set of most recent reviews/posts which will then be input to the machine learning API endpoint and analyzed to provide the user a result and confidence percent for the search query.
    - API will be created by using the popular framework Flask
- This website will be hosted on a cloud platform like Azure.

Project Plan

This project tasks have been divided into 2 categories, classified by components detailed in the previous section. The task assignment between team members is using Agile methodologies, where a team member picks up tasks consecutively as per available hours. There would be a weekly meeting where the members would report completed tasks, as well as any roadblocks.

Each team member oversees part of the project assigning tasks if necessary and managing task progress between team members:

- Architecture, Deployment and General Project Management: Tejas Vyas
- Machine Learning Model Development: Manu Sihag
- Data Analysis and Reports: Gbemisola Banjoko
- Web Application Development and Integration: Riddhi Thakkar

This project assumes a team of 4, available to put in 10 hours a week towards the project with about 10 weeks of available time before final product deployment. This provides an approximate total budget of 400 work hours. Details of categories and tasks are as follows:

Machine Learning Model/Endpoint

Task	Effort (hrs)	Deadline
Collect Information about dataset	10	01/10/2020
Analyze Dataset	10	05/10/2020
Clean Dataset	10	07/10/2020
Create Data Preprocessing Pipeline	5	09/10/2020
Perform Feature Extraction using n-gram approach	10	14/10/2020
Prototype Model – Naïve-Bayes Algorithm	10	22/10/2020

Prototype Model – Logistic Regression	10	22/10/2020
Prototype Model – Support Vector Machines	10	22/10/2020
Prototype Model – Deep Learning (LSTM)	10	22/10/2020
Evaluate Prototypes	10	25/10/2020
Analyze best approach	10	27/10/2020
Prototype chosen Algorithm	10	29/10/2020
Develop Model	10	05/11/2020
Train Model	20	12/11/2020
Validate Model	5	14/11/2020
Test Model	10	17/11/2020
Refine Model/Architecture	5	19/11/2020
Develop Report	5	21/11/2020
Develop Dashboard for Model Review	5	24/11/2020
Develop API Endpoint for Result Aggregation	30	27/11/2020
Develop Pipeline for Model Retraining	10	29/11/2020
Test API Endpoint	10	01/12/2020
Deploy Model/API to Production	20	03/12/2020
Test API in Production	10	05/12/2020
Total	255	

Web Application

Task	Effort (hrs)	Deadline
Collect Information about web application	10	01/10/2020
Prepare Application Specifications	10	08/10/2020
Design Application Architecture	10	15/10/2020
Develop Web Application	35	19/11/2020
Integrate API Endpoints	35	30/11/2020
Test Web Application	10	03/12/2020
Deploy Web Application to Production	10	08/12/2020
Perform Integration Testing in Production	10	10/12/2020
Total	120	

Miscellaneous Tasks

Task	Effort (hrs)	Deadline
Prepare Demo Presentation	5	10/12/2020
Perform Project Management Tasks (Communication, Feedback, Meetings)	20	N/A
Total	25	

Exploratory Data Analysis

This section includes the exploratory analysis performed on the dataset, and includes NLP specific feature analysis including word-clouds and topic modelling with LDA.

Data Provisioning

We begin data provisioning by loading data into Python. The data is divided by individual sources, in 3 text files – amazon_cells_labelled.txt, yelp_labelled.txt, and imdb_labelled.txt. We import all of them and place them in 1 pandas DataFrame, while adding a column “source” to keep track of the source for reference.

Importing data

```
In [2]: #Amazon Data
input_file = "amazon_cells_labelled.txt"
amazon = pd.read_csv(input_file,delimiter='\t',header=None, names=['review', 'sentiment'])
amazon['source']='amazon'

#Yelp Data
input_file = "yelp_labelled.txt"
yelp = pd.read_csv(input_file,delimiter='\t',header=None, names=['review', 'sentiment'])
yelp['source']='yelp'

#Imdb Data
input_file = "imdb_labelled.txt"
imdb = pd.read_csv(input_file,delimiter='\t',header=None, names=['review', 'sentiment'])
imdb['source']='imdb'

#combine all data sets
data = pd.DataFrame()
data = pd.concat([amazon, yelp, imdb])
data['sentiment'] = data['sentiment'].astype(str)
print(data.head(5))
print(data.tail(5))
```

```
                                              review sentiment  source
0  So there is no way for me to plug it in here i...         0  amazon
1                        Good case, Excellent value.         1  amazon
2                             Great for the jawbone.         1  amazon
3  Tied to charger for conversations lasting more...         0  amazon
4                                   The mic is great.         1  amazon
                                              review sentiment source
743  I just got bored watching Jessice Lange take h...         0   imdb
744  Unfortunately, any virtue in this film's produ...         0   imdb
745                    In a word, it is embarrassing.         0   imdb
746                                  Exceptionally bad!         0   imdb
747  All in all its an insult to one's intelligence...         0   imdb
```

After using pandas import function, we have the data loaded, we can see we have 2748 rows containing 3 columns: **review** (string), **sentiment** (bit) and **source**.
We check the number of null rows, since this will be required to know for data cleanup.

Reviewing Data Distribution

```
In [3]: print(data.info())
```

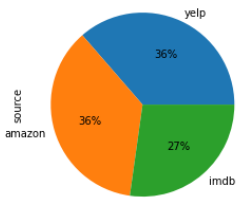
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2748 entries, 0 to 747
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     2748 non-null   object
 1   sentiment  2748 non-null   object
 2   source     2748 non-null   object
dtypes: object(3)
memory usage: 85.9+ KB
None
```

```
In [80]: print('Review NaN row counts:', data['review'].isnull().sum())
print('Seniment NaN row counts:', data['sentiment'].isnull().sum())
```

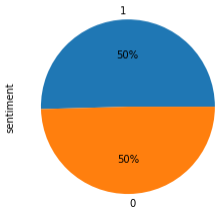
```
Review NaN row counts: 0
Seniment NaN row counts: 0
```

Since this dataset is from a research paper, limited preprocessing as already been done on it, which explains the lack of NaN rows. We also verify the data distribution to check for possible biases using pie charts.

```
In [81]: data['source'].value_counts().plot(kind='pie', autopct='%1.0f%%')
Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x217cd479ca0>
```




```
In [4]: data.sentiment.value_counts().plot(kind='pie', autopct='%1.0f%%')
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x217b37fd640>
```



The Data Distribution screenshot also shows us the memory usage of the DataFrame, and the pie chart show the Amazon and Yelp contain 36% rows each, with imdb having lower data due to possible null rows. Finally, another pie chart verifies the sentiment is divided into 50% positive and 50% negative, avoiding bias towards a specific sentiment in the model.

Exploratory Data Analysis

We begin analysis of the data and feature generation by reviewing Since our main data contains primarily of english words, we download the NLTK library, to get some insights on how the sentences are structured. We apply lambda function calls to DataFrame and add columns for **word_count, char_count, stopwords, num_count** and **upper_count,** to keep track of number of words, characters, stopwords, numbers, and upper digits in each sentence respectively.

```
In [5]: import nltk
        nltk.download('stopwords')

        data['word_count'] = data['review'].apply(lambda x : len(x.split()))
        data['char_count'] = data['review'].apply(lambda x : len(x.replace(" ","")))
        data['stopwords'] = data['review'].apply(lambda x: len([x for x in x.split() if x in stopwords.words('english')]))
        data['num_count'] = data['review'].apply(lambda x: len([x for x in x.split() if x.isdigit()]))
        data['upper_count'] = data['review'].apply(lambda x: len([x for x in x.split() if x.isupper()]))

        print(data[['word_count', 'char_count', 'stopwords', 'num_count', 'upper_count']].head(5))

        data.sum(axis = 0, numeric_only = True)
```

```
   word_count  char_count  stopwords  num_count  upper_count
0          21          62         13          0            2
1           4          24          0          0            0
2           4          19          2          0            0
3          11          69          4          1            1
4           4          14          1          0            0
```

```
Out[5]: word_count      35742
        char_count     162066
        stopwords       13865
        num_count         404
        upper_count      1260
        dtype: int64
```

We can see we have over 35000 words, out of which 13865 are stopwords, which can be excluded during data cleaning and visualizations, decreasing amount of features we need to process by 38%.

Word Cloud

Next part of analyzing the dataset would be creation of word clouds, which would assist in analysis of textual data points. Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance.

For generating word clouds in Python, we use the matplotlib and wordcloud modules, creating 3 pandas DataFrames respectively to show wordclouds for all words, as well as for negative and positive words. We exclude stopwords while generation of wordclouds and limit the max words in cloud to 100 for readability.

Word Clouds

```
In [6]: all_df1 = data[['review', 'sentiment']]
        all_df1['sentiment'] = all_df1['sentiment'].astype(int)
        df_1 = all_df1[all_df1['sentiment']==1]
        df_0 = all_df1[all_df1['sentiment']==0]
        rev_All = " ".join(review for review in all_df1.review)
        rev_1 = " ".join(review for review in df_1.review)
        rev_0 = " ".join(review for review in df_0.review)

        fig, ax = plt.subplots(3, 1, figsize  = (30,30))
        # Create and generate a word cloud image:
        wordcloud_ALL = WordCloud(width=1600, height=800, max_font_size=200,collocations=False, max_words=100, background_color="white").generate(rev_All)
        wordcloud_1 = WordCloud(width=1600, height=800, max_font_size=200,collocations=False, max_words=100, background_color="green").generate(rev_1)
        wordcloud_0 = WordCloud(width=1600, height=800, max_font_size=200,collocations=False, max_words=100, background_color="red").generate(rev_0)

        # Display the generated image:
        ax[0].imshow(wordcloud_ALL, interpolation='bilinear')
        ax[0].set_title('All Reviews', fontsize=30)
        ax[0].axis('off')
        ax[1].imshow(wordcloud_1, interpolation='bilinear')
        ax[1].set_title('Positive Reviews',fontsize=30)
        ax[1].axis('off')
        ax[2].imshow(wordcloud_0, interpolation='bilinear')
        ax[2].set_title('Negative Reviews',fontsize=30)
        ax[2].axis('off')
```

We generate a **white** word cloud for **all** reviews, **green** word cloud for **positive** reviews, and **red** word cloud for **negative** reviews. In the wordcloud, larger words have a higher frequency of use, which we can use to understand how the word great is most often used in positive reviews, and bad is most often used in negative reviews.

We now perform tokenization on the **sentences** column of the DataFrame and use the results to get further insights about frequency of words used.

(More details on tokenization and pre-processing available in the next section: Data Preprocessing Pipeline)

N-gram Analysis

N-gram(s) are a sequence of N words. In NLP, n-grams are used for a variety of things, such as auto completion of sentences, auto spell check, and grammar checks. We will be using n-grams to understand the feature set better and use the probability of bi-grams and tri-grams to generate topics and perform Topic Modeling using Latent Dirichlet allocation (LDA).

We begin by downloading *wordnet* module from NLTK library and create a function to perform lemmatization.

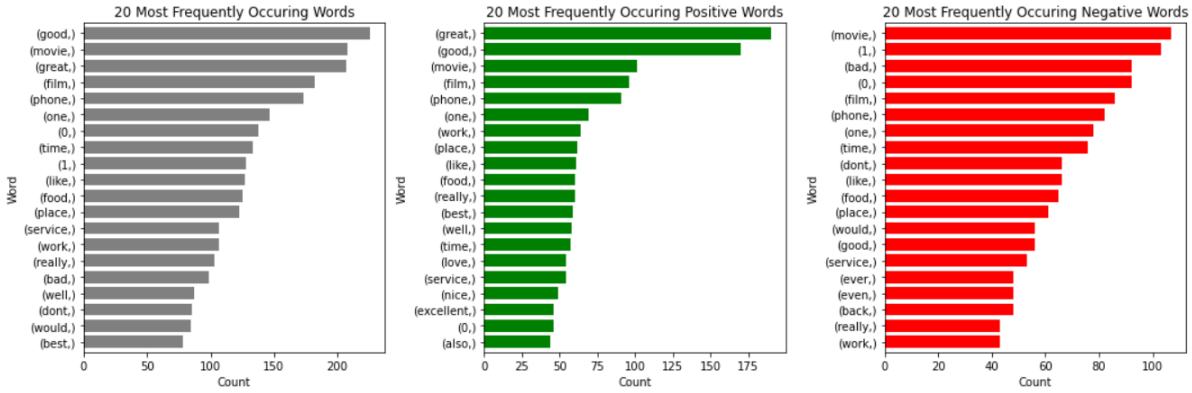
```
In [69]: nltk.download('wordnet')

         def basic_clean(text):
           wnl = nltk.stem.WordNetLemmatizer()
           stopwords = nltk.corpus.stopwords.words('english')
           words = re.sub(r'[^\w\s]', '', text).split()
           return [wnl.lemmatize(word) for word in words if word not in stopwords]
```

Now we start using this function to perform word frequency analysis for positive, negative and all words in our dataset:

```
In [46]: pos_words = basic_clean(''.join(str(data.where(data['sentiment']=="1").dropna()['body_text_nostop'].tolist())))
         neg_words = basic_clean(''.join(str(data.where(data['sentiment']=="0").dropna()['body_text_nostop'].tolist())))
         all_words = basic_clean(''.join(str(data.dropna()['body_text_nostop'].tolist())))
```

Now we get the top 20 highest frequency words, and positive and negative words and generate histograms for them:

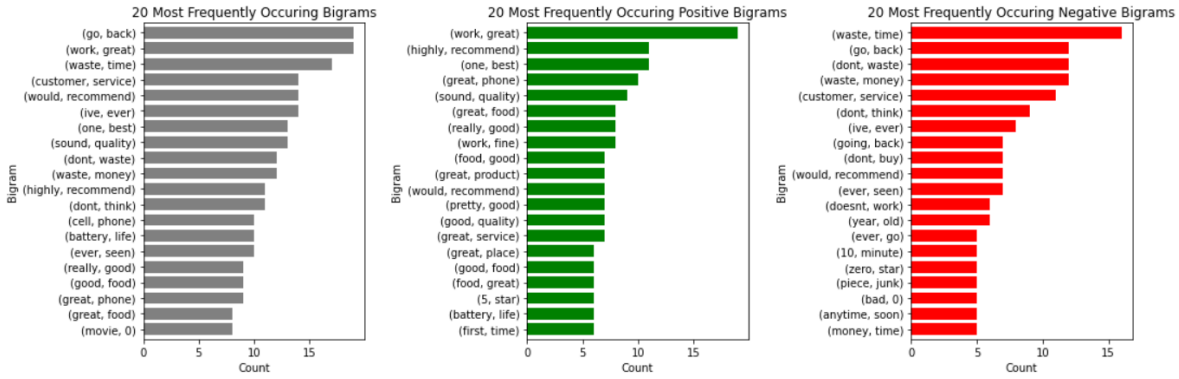
```
In [50]: unigrams_series = (pd.Series(nltk.ngrams(all_words,1)).value_counts())[:20]
         pos_unigrams_series = (pd.Series(nltk.ngrams(pos_words,1)).value_counts())[:20]
         neg_unigrams_series = (pd.Series(nltk.ngrams(neg_words,1)).value_counts())[:20]

         fig, axes = plt.subplots(1,3,figsize=(15,5))
         unigrams_series.sort_values().plot.barh(color='gray', ax=axes[0], width=0.75, title='20 Most Frequently Occuring Words')
         pos_unigrams_series.sort_values().plot.barh(color='green', ax=axes[1], width=0.75, title='20 Most Frequently Occuring Positive Words')
         neg_unigrams_series.sort_values().plot.barh(color='red', ax=axes[2], width=0.75, title='20 Most Frequently Occuring Negative Words')
         axes[0].set_xlabel("Count")
         axes[0].set_ylabel("Word")
         axes[1].set_xlabel("Count")
         axes[1].set_ylabel("Word")
         axes[2].set_xlabel("Count")
         axes[2].set_ylabel("Word")
         plt.tight_layout()
```



We can check against the word clouds which shows the frequency of the words are remarkably similar to the generated word clouds for respective case.

We now create bigrams and look at their frequencies:

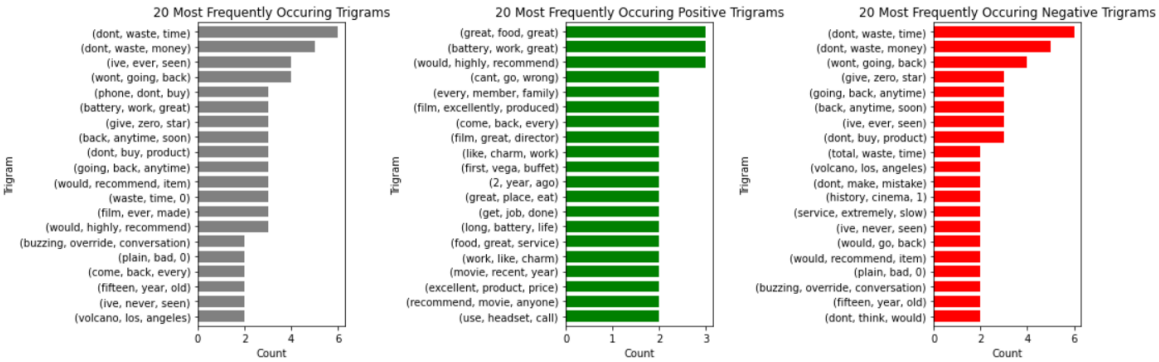
```
In [52]: bigrams_series = (pd.Series(nltk.ngrams(all_words,2)).value_counts())[:20]
pos_bigrams_series = (pd.Series(nltk.ngrams(pos_words,2)).value_counts())[:20]
neg_bigrams_series = (pd.Series(nltk.ngrams(neg_words,2)).value_counts())[:20]

fig, axes = plt.subplots(1,3,figsize=(15,5))
bigrams_series.sort_values().plot.barh(color='gray', ax=axes[0], width=0.75, title='20 Most Frequently Occuring Bigrams')
pos_bigrams_series.sort_values().plot.barh(color='green', ax=axes[1], width=0.75, title='20 Most Frequently Occuring Positive Bigrams')
neg_bigrams_series.sort_values().plot.barh(color='red', ax=axes[2], width=0.75, title='20 Most Frequently Occuring Negative Bigrams')
axes[0].set_xlabel("Count")
axes[0].set_ylabel("Bigram")
axes[1].set_xlabel("Count")
axes[1].set_ylabel("Bigram")
axes[2].set_xlabel("Count")
axes[2].set_ylabel("Bigram")
plt.tight_layout()
```



We can see highest occurring positive bigram is (work, great), negative bigram is (waste, time), and general bigram is (work, great)

Finally, we review trigrams and look at their frequencies:

```
In [54]: trigrams_series = (pd.Series(nltk.ngrams(all_words,3)).value_counts())[:20]
pos_trigram_series = (pd.Series(nltk.ngrams(pos_words,3)).value_counts())[:20]
neg_trigram_series = (pd.Series(nltk.ngrams(neg_words,3)).value_counts())[:20]

fig, axes = plt.subplots(1,3,figsize=(15,5))
trigrams_series.sort_values().plot.barh(color='gray', ax=axes[0], width=0.75, title='20 Most Frequently Occuring Trigrams')
pos_trigram_series.sort_values().plot.barh(color='green', ax=axes[1], width=0.75, title='20 Most Frequently Occuring Positive Trigrams')
neg_trigram_series.sort_values().plot.barh(color='red', ax=axes[2], width=0.75, title='20 Most Frequently Occuring Negative Trigrams')
axes[0].set_xlabel("Count")
axes[0].set_ylabel("Trigram")
axes[1].set_xlabel("Count")
axes[1].set_ylabel("Trigram")
axes[2].set_xlabel("Count")
axes[2].set_ylabel("Trigram")
plt.tight_layout()
```



We can notice highest occurring trigram and negative trigram is (dont, waste, money) and positive trigram is (great, food, great). With increasing n-grams, due to the nature of dataset, we can also notice the occurrence decreasing significantly, but with bigrams and trigrams the model would be able to analyze probability of next occurrence of a word and understand sentiment better.

Topic Modeling with LDA

Topic modeling is an unsupervised machine learning technique used to scan a set of documents, and assists in detecting word and phrase patterns within them, automatically clustering word groups and similar expressions (topics) that best characterize a set of documents.

We will be performing topic modeling using a technique called Latent Dirichlet allocation (LDA). Latent Dirichlet Allocation (LDA) is a "generative probabilistic model" of a collection of composites made up of parts. In terms of topic modelling, the composites are documents and the parts are words and/or phrases (phrases n words in length are referred to as n-grams).

To use this, we import *LatentDirichletAllocation* module from *sklearn* library, and create a general function to print topics for a given set of words. We limit topics to 5 and number of words to 10, to get analyzable data.

Setup

In [64]:
```
import warnings
warnings.simplefilter("ignore", DeprecationWarning)
# Load the LDA model from sk-learn
from sklearn.decomposition import LatentDirichletAllocation as LDA

# Helper function
def print_topics(model, count_vectorizer, n_top_words):
    words = count_vectorizer.get_feature_names()
    for topic_idx, topic in enumerate(model.components_):
        print("Topic #%d:" % topic_idx, " ".join([words[i]
                        for i in topic.argsort()[:-n_top_words - 1:-1]]).strip())

# Tweak the two parameters below
number_topics = 5
number_words = 10
```

Now we train this model for all words, positive words and negative words and get top 5 topics.

All Topics

In [63]:
```
# Create and fit the LDA model
lda = LDA(n_components=number_topics, n_jobs=-1)
lda.fit(x_count)
# Print the topics found by the LDA model
print("Topics found via LDA:\n")
print_topics(lda, count_vect, number_words)
```

```
Topics found via LDA:

Topic #0: would  disappointed recommend phone even place well item ive
Topic #1: 0 1 film movie one like time good great
Topic #2: phone great headset one ever worst service good sound
Topic #3: good food great really place service movie also like
Topic #4: movie go bad back good like film one worth
```

Positive topics

In [65]:
```
count_vect_pos=CountVectorizer(analyzer=remove_stopwords)
x_count_pos=count_vect_pos.fit_transform(data.where(data['sentiment']=="1")\
                                         .dropna()['body_text_nostop'])
# Create and fit the LDA model
lda_pos = LDA(n_components=number_topics, n_jobs=-1)
lda_pos.fit(x_count_pos)
# Print the topics found by the LDA model
print("Positive topics found via LDA:")
print_topics(lda_pos, count_vect_pos, number_words)
```

```
Positive topics found via LDA:
Topic #0: service food  nice place friendly delicious like staff pretty
Topic #1: good really  food phone movie excellent quality product definitely
Topic #2: headset best good works sound excellent great well comfortable
Topic #3: movie film 0 good well one also 1 great
Topic #4: great phone  place recommend one love works price product
```

Negative Topics

In [62]:
```
count_vect_neg=CountVectorizer(analyzer=remove_stopwords)
x_count_neg=count_vect_neg.fit_transform(data.where(data['sentiment']=="0")\
                                         .dropna()['body_text_nostop'])
# Create and fit the LDA model
lda_neg = LDA(n_components=number_topics, n_jobs=-1)
lda_neg.fit(x_count_neg)
# Print the topics found by the LDA model
print("Negative topics found via LDA:")
print_topics(lda_neg, count_vect_neg, number_words)
```

```
Negative topics found via LDA:
Topic #0: phone  bad quality minutes food never disappointing poor waited
Topic #1: 1 0  one movie film dont didnt work even
Topic #2: back time dont waste money would place go really
Topic #3: service  place food ever good worst terrible slow ive
Topic #4: bad like im movie film phone awful food would
```

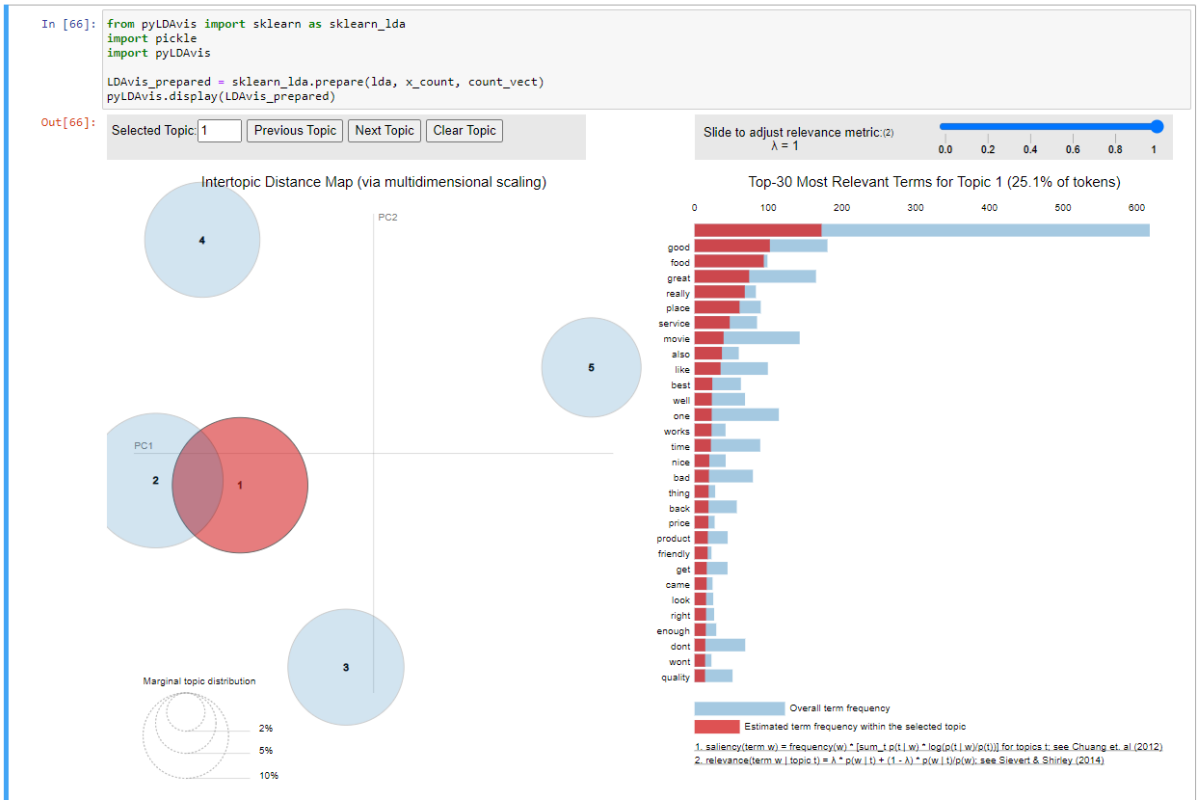
We can notice how the topics generally include highest frequency words from our n-gram analysis earlier, and these topics provide us an overview of how positive and negative words can be characterized.

We also create visualizations for these using *pyLDAvis* library, which allows us to view distributions and respective state of topics interactively. We do this for positive, negative and all words.

All words topic visualization

In [66]:
```
from pyLDAvis import sklearn as sklearn_lda
import pickle
import pyLDAvis

LDAvis_prepared = sklearn_lda.prepare(lda, x_count, count_vect)
pyLDAvis.display(LDAvis_prepared)
```

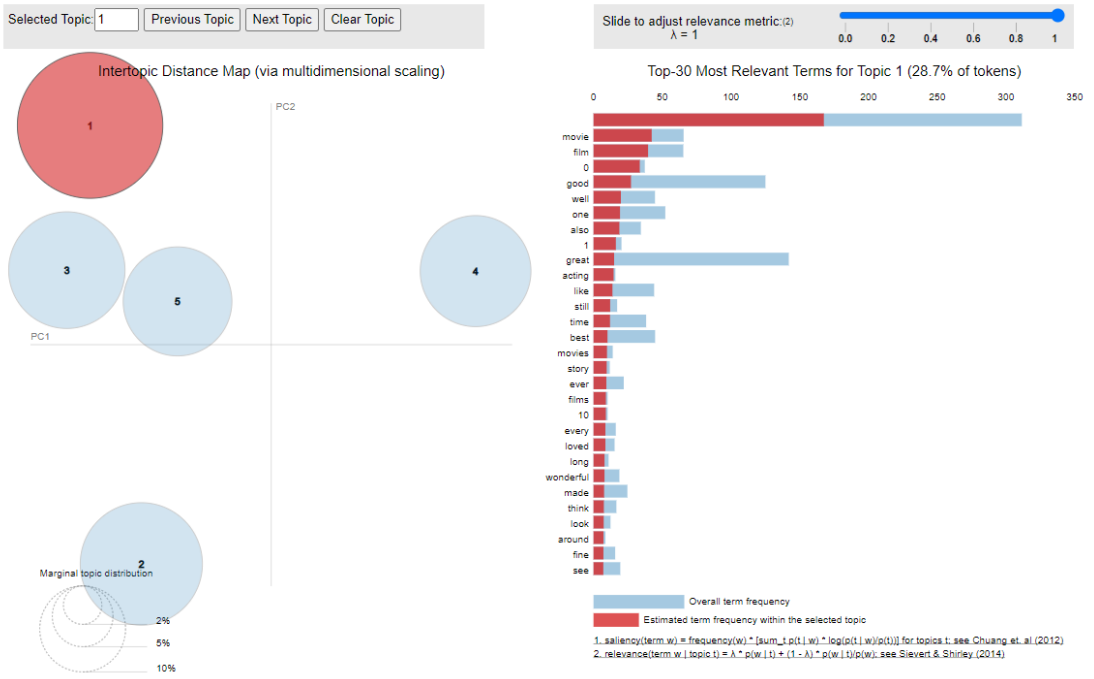
Out[66]:



Positive topics Visualization

```
In [67]: LDAvis_prepared_pos = sklearn_lda.prepare(lda_pos, x_count_pos, count_vect_pos)
         pyLDAvis.display(LDAvis_prepared_pos)
```

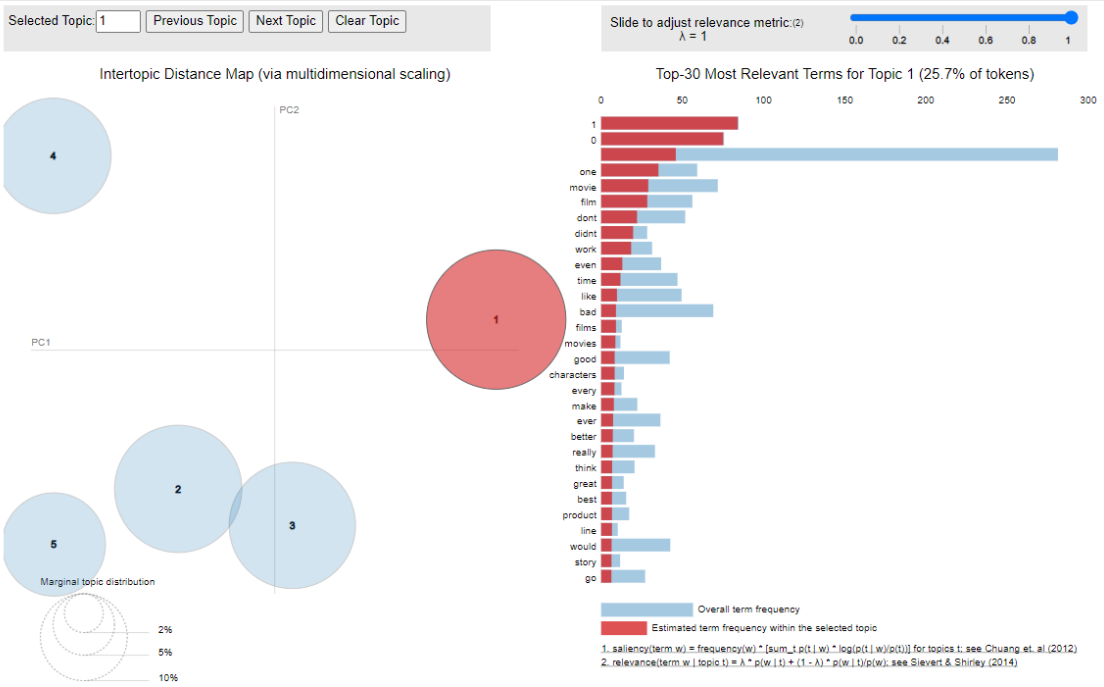
Out[67]:



Negative Topics Visualization

```
In [68]: LDAvis_prepared_neg = sklearn_lda.prepare(lda_neg, x_count_neg, count_vect_neg)
         pyLDAvis.display(LDAvis_prepared_neg)
```

Out[68]:



We can see the term frequencies match our earlier analysis and the topics are generally spread across the space, this is because the sources we're using are specific to food, movies and products, which have limited correlation in between them.

Data Preprocessing Pipeline

Data Cleaning and Preparation

We perform data cleanup by removing all punctuations from the data, followed by tokenization and finally remove stop words.

We do not need the counter columns we created earlier so we can drop them.

In [73]:
```
data = data.drop(columns=['word_count','char_count','num_count','upper_count','stopwords'])
```

Punctuation Removal

When analyzing sentiments, punctuations may be ignored and therefore we remove them and create a new column on the DataFrame named **body_text_clean**

Data Cleanup and tokenization

In [85]:
```
import string
def remove_punct(text):
  text_nopunct= "".join ([char for char in text if char not in string.punctuation])
  return text_nopunct
data['body_text_clean']=data['review'].apply(lambda x:remove_punct(x))
data.head()
```

Out[85]:

	review	sentiment	source	body_text_clean
0	So there is no way for me to plug it in here i...	0	amazon	So there is no way for me to plug it in here i...
1	Good case, Excellent value.	1	amazon	Good case Excellent value
2	Great for the jawbone.	1	amazon	Great for the jawbone
3	Tied to charger for conversations lasting more...	0	amazon	Tied to charger for conversations lasting more...
4	The mic is great.	1	amazon	The mic is great

Tokenization

Using sentences directly in the model would cause us to lose accuracy since the model needs to understand correlation between the words. We do this by tokenizing the data (and lowering case) and placing the data in a new column named **body_text_tokenized**

In [86]:
```
import re
def tokenize(text):
  tokens = re.split('\W+', text)
  return tokens
data['body_text_tokenized']=data['body_text_clean'].apply(lambda x: tokenize(x.lower()))
data.head()
```

Out[86]:

	review	sentiment	source	body_text_clean	body_text_tokenized
0	So there is no way for me to plug it in here i...	0	amazon	So there is no way for me to plug it in here i...	[so, there, is, no, way, for, me, to, plug, it...
1	Good case, Excellent value.	1	amazon	Good case Excellent value	[good, case, excellent, value]
2	Great for the jawbone.	1	amazon	Great for the jawbone	[great, for, the, jawbone]
3	Tied to charger for conversations lasting more...	0	amazon	Tied to charger for conversations lasting more...	[tied, to, charger, for, conversations, lastin...
4	The mic is great.	1	amazon	The mic is great	[the, mic, is, great]

Stop Word Removal

Stop words are words which are filtered out before or after processing of natural language data. Leaving these words in the dataset would result in additional processing to be performed and also skew our results since stop words do not assist as in determination of sentiment.

We perform this by using the NLTK corpus of stopwords and place the results in a new column **body_text_nostop**.

```
In [87]: stopword = nltk.corpus.stopwords.words('english')
         def remove_stopwords(tokenized_list):
           text = [word for word in tokenized_list if word not in stopword]
           return text

         data['body_text_nostop']=data['body_text_tokenized'].apply(lambda x: remove_stopwords(x))
         data.head()
```

Out[87]:

	review	sentiment	source	body_text_clean	body_text_tokenized	body_text_nostop
0	So there is no way for me to plug it in here i...	0	amazon	So there is no way for me to plug it in here i...	[so, there, is, no, way, for, me, to, plug, it...	[way, plug, us, unless, go, converter]
1	Good case, Excellent value.	1	amazon	Good case Excellent value	[good, case, excellent, value]	[good, case, excellent, value]
2	Great for the jawbone.	1	amazon	Great for the jawbone	[great, for, the, jawbone]	[great, jawbone]
3	Tied to charger for conversations lasting more...	0	amazon	Tied to charger for conversations lasting more...	[tied, to, charger, for, conversations, lastin...	[tied, charger, conversations, lasting, 45, mi...
4	The mic is great.	1	amazon	The mic is great	[the, mic, is, great]	[mic, great]

Vectorization

Machine learning models perform better with vectors so we use *CountVectorizer* module from *sklearn* library to extract features out of the text tokens generated above. We will also use this to perform n-gram analysis and Topic Modeling later, which are inherent to Natural Language Processing (NLP)

```
In [10]: from sklearn.feature_extraction.text import CountVectorizer
         count_vect=CountVectorizer(analyzer=remove_stopwords)
         x_count=count_vect.fit_transform(data['body_text_nostop'])
         print(x_count.shape)
         print(count_vect.get_feature_names())
```

```
(2748, 5277)
['believed', 'bell', 'bellagio', 'bellies', 'bells', 'bellucci', 'belly', 'belmondo', 'belowpar', 'belt', 'ben', 'bend', 'bennett', 'bergens', 'bertolucci',
't', 'bethe', 'better', 'betty', 'beware', 'beyond', 'bible', 'big', 'bigbudget', 'bigger', 'biggest', 'bill', 'bills', 'billy', 'binge', 'biographical',
d', 'biscuit', 'biscuits', 'bisque', 'bit', 'bitches', 'bitchy', 'bite', 'bites', 'bitpim', 'bits', 'black', 'blackberry', 'blacks', 'blacktop', 'blah',
land', 'blandest', 'blandly', 'blanket', 'blare', 'blatant', 'blew', 'blist', 'block', 'bloddy', 'blood', 'bloodiest', 'bloody', 'blow', 'blown', 'blows',
'bluegreenscreen', 'bluetoooth', 'bluetooth', 'bluetoothmotorola', 'bluetooths', 'blush', 'bmw', 'boasts', 'bob', 'boba', 'bodes', 'body', 'bohemian', 't
'bold', 'bombardments', 'bond', 'bone', 'bonus', 'bonuses', 'boobs', 'boogeyman', 'book', 'booking', 'booksomethats', 'boost', 'boot', 'bop', 'bordered',
ders', 'bore', 'bored', 'boring', 'boringpointless', 'borrowed', 'bose', 'boss', 'bother', 'bothersome', 'bottom', 'bottowm', 'bouchon', 'bought', 'bougt
'boxes', 'boy', 'boyfriend', 'boyle', 'boys', 'brain', 'brainsucking', 'brand', 'brat', 'bread', 'break', 'breakage', 'breakfast', 'breakfastlunch', 'bre
reeders', 'breeze', 'brevity', 'brian', 'brick', 'brief', 'brigand', 'bright', 'brilliance', 'brilliant', 'brilliantly', 'bring', 'brings', 'brisket', 't
ken', 'brokeni', 'brooding', 'brother', 'brought', 'brownish', 'browser', 'browsing', 'brunch', 'bruschetta', 'brushfire', 'brutal', 'bt', 'bt250v', 'bts
ks', 'buddy', 'budget', 'buds', 'buffalo', 'buffet', 'buffets', 'bug', 'build', 'builders', 'building', 'buildings', 'buildup', 'built', 'buldogis', 'bul
ly', 'bumpers', 'bunch', 'burger', 'burgers', 'burned', 'burrittos', 'burtons', 'bus', 'business', 'businesses', 'bussell', 'busy', 'butter', 'button',
yer', 'buyerbe', 'buyers', 'buying', 'buyit', 'buzzing', 'bye', 'ca42', 'caballeros', 'cable', 'cables', 'caesar', 'cafe', 'café', 'cailles', 'cakeohhh',
i', 'calendar', 'california', 'call', 'called', 'calligraphy', 'callings', 'calls', 'came', 'camelback', 'cameo', 'camera', 'camerawork', 'camp', 'campy'
'cancan', 'cancellation', 'cancelling', 'candace', 'candle', 'cando', 'cannoli', 'cannot', 'cant', 'capability', 'capacity', 'cape', 'capers', 'captain',
es', 'car', 'carbs', 'card', 'cardboard', 'cardellini', 'care', 'careful', 'caring', 'carlys', 'carol', 'carpaccio', 'carrell', 'carried', 'carriers', 'c
ars', 'cart', 'cartel', 'cartoon', 'cartoons', 'case', 'cases', 'cash', 'cashew', 'cashier', 'casing', 'casino', 'cassette', 'cast', 'casted', 'casting',
'catchy', 'caterpillar', 'caught', 'cause', 'caused', 'causing', 'cavier', 'cbr', 'cds', 'ceases', 'celebration', 'celebrity', 'cell', 'cellphone', 'cell
'celluloid', 'center', 'centers', 'central', 'cents', 'century', 'certain', 'certainly', 'ce', 'cei', 'chai', 'chains', 'chalkboard', 'challenges', 'char
```

```
In [11]: # In this step, we are turning our count vectorization output into a dataframe, and using the individual features of our data as the colucolumn heading so w
         x_count_df=pd.DataFrame(x_count.toarray())
         x_count_df.columns = count_vect.get_feature_names()
         x_count_df
```

Out[11]:

	0	010	1	10	100	1010	11	110	1199	...	yum	yummy	yun	z	z500a	zero	zillion	zombie	zombiestudents	zombiez
0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2743	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2744	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2745	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2746	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2747	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2748 rows × 5277 columns

This finally results in our cleaned data containing features with **shape (2748,5277).**

Test-Train Split

Before using machine learning algorithms for prototyping, we use the *train-test-split* module from *sklearn* and generate training and testing sets.

For the prototype, we limited training set (X_train) to 75% and testing set (X_test) to 25% of the original dataset's sentences. Additionally, we also split labels (sentiments) in the same ratio, to be used for training (y_train) and validation (y_test).

Setup train and test splits

In [90]:
```
from sklearn import linear_model, model_selection


clean_sentences = []
for l in data['body_text_nostop'].values:
    clean_sentences.append(' '.join(l))
#print(clean_sentences)
y = data['sentiment']

sentences_train, sentences_test, y_train, y_test = model_selection.train_test_split(clean_sentences, y, test_size=0.25, random_state=42)
#print(sentences_train)
vectorizer = CountVectorizer(ngram_range=(1, 3))
vectorizer.fit(sentences_train)
X = vectorizer.transform(clean_sentences)
X_train = vectorizer.transform(sentences_train)
X_test  = vectorizer.transform(sentences_test)

from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transform=TfidfTransformer()
X_tfidf = tfidf_transform.fit_transform(X)
X_train_tfidf=tfidf_transform.fit_transform(X_train)
X_test_tfidf=tfidf_transform.fit_transform(X_test)
```

We create 2 different training and testing sets using 2 vectorizers:

- *CountVectorizer*
- *TfidfTransformer*

to make sure the data is in a correct format to be used by machine learning classifiers, while comparing the effect of the different kinds of vectorizers.

Algorithm Evaluation

Now that we have our data vectorized, we can train our model prototypes using multiple algorithms. For each algorithm we picked during initial approach, we train them once using Count Vectorizer, and then perform another training instance for Tfidf Vectorizer.

We start by training – below screenshots show the libraries used and logic to train the model:

- Logistic Regression
    - Logistic Regression is a classification model that is used when we have a binary dependent variable. It uses the logistic function, also called as sigmoid function with formula: $1 / (1 + e^{-value})$.

In [91]:
```
#Logistic Regression
logRes = linear_model.LogisticRegression()
logRes.fit(X_train, y_train)
logResscore = logRes.score(X_test, y_test)
```

In [92]:
```
logRes_tfidf = linear_model.LogisticRegression()
logRes_tfidf.fit(X_train_tfidf, y_train)
logRes_tfidf_score = logRes_tfidf.score(X_test_tfidf, y_test)
```

- Naïve Bayes
    - Naïve Bayes is a supervised learning algorithm, based on Bayes theorem and used for solving classification problems.
    - It's a simple probabilistic classifier using conditional probabilities with formula:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In [93]:
```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train.toarray(), y_train)
nbscore = nb.score(X_test.toarray(), y_test)
```

In [94]:
```
nb_tfidf = GaussianNB()
nb_tfidf.fit(X_train_tfidf.toarray(), y_train)
nb_tfidf_score = nb_tfidf.score(X_test_tfidf.toarray(), y_test)
```

- Support Vector Machine
  - Support Vector Machine, abbreviated as SVM is another Supervised algorithm used for both regression and classification tasks.
  - The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — # of features) that distinctly classifies the data points.

In [95]:
```
#Support Vector Machines
from sklearn import svm

svc_classifier = svm.SVC()
svc_classifier.fit(X_train.toarray(), y_train)
svc_classifierscore = svc_classifier.score(X_test.toarray(), y_test)
```

In [96]:
```
svc_classifier_tfidf = svm.SVC()
svc_classifier_tfidf.fit(X_train_tfidf.toarray(), y_train)
svc_classifier_tfidf_score = svc_classifier_tfidf.score(X_test_tfidf.toarray(), y_test)
```

- Decision Trees
  - Decision Trees are one of the simplest algorithms to understand, used in both classification and regression. In decision analysis, a decision tree is used to visually and explicitly represent decisions, following a reverse tree like structure.

In [97]:
```
#Decision Trees
from sklearn import tree

dec_tree_tfidf = tree.DecisionTreeClassifier()
dec_tree_tfidf.fit(X_train_tfidf.toarray(), y_train)
dec_tree_tfidf_score = dec_tree_tfidf.score(X_test_tfidf.toarray(), y_test)
```

In [98]:
```
dec_tree = tree.DecisionTreeClassifier()
dec_tree.fit(X_train.toarray(), y_train)
dec_treescore = dec_tree.score(X_test.toarray(), y_test)
```

- Random Forests
  - Random Forest is another versatile supervised machine learning algorithm that combines multiple decision trees to form a “forest.”

In [99]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier

rf_tfidf = RandomForestClassifier()
rf_tfidf.fit(X_train_tfidf.toarray(), y_train)
rf_tfidf_score = rf_tfidf.score(X_test_tfidf.toarray(), y_test)
```

In [100]:
```
rf = RandomForestClassifier()
rf.fit(X_train.toarray(), y_train)
rfscore = rf.score(X_test.toarray(), y_test)
```

- Deep Learning - Long short-term memory
    - Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data point, but also entire sequences of data.
    - A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

In [102]:
```
#LSTM
#!pip install tensorflow_datasets
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout
import tensorflow_datasets as tfds
import numpy as np
max_length =100
trunc_type='post'
padding_type='post'
#pad all Sequence
vocab_size=1000

temp_sentences=data['review'].tolist()
label=data['sentiment'].astype(int).tolist()

tokenizer_tfds=tfds.deprecated.text.SubwordTextEncoder.build_from_corpus(temp_sentences,vocab_size,max_subword_length=5)
for i,sent in enumerate(temp_sentences):
    temp_sentences[i]=tokenizer_tfds.encode(sent)
sequence_added=pad_sequences(temp_sentences,maxlen=max_length,padding =padding_type,truncating=trunc_type)

training_size=int(len(temp_sentences)*0.75)
train_seq=sequence_added[:training_size]
train_labels=label[:training_size]

test_seq=sequence_added[training_size:]
test_labels=label[training_size:]

train_labels=np.array(train_labels)
test_labels=np.array(test_labels)

def createLSTM():
    embedding_dim=16
    model=Sequential([
        Embedding(1000,embedding_dim,input_length=100),
        Bidirectional(LSTM(embedding_dim,return_sequences=True)),
        Bidirectional(LSTM(embedding_dim)),
        Dense(6,activation='relu'),
        Dense(1,activation='sigmoid')
    ])
    return model
```

In [104]:
```
lstm_classifier = createLSTM()
lstm_classifier.compile(loss="binary_crossentropy",optimizer='adam',metrics=['accuracy'])
lstm_classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 16)           16000
_________________________________________________________________
bidirectional (Bidirectional (None, 100, 32)           4224
_________________________________________________________________
bidirectional_1 (Bidirection (None, 32)                6272
_________________________________________________________________
dense (Dense)                (None, 6)                 198
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 26,701
Trainable params: 26,701
Non-trainable params: 0
_________________________________________________________________
```

In [106]:
```
lstm_classifier.fit(train_seq,train_labels,epochs=12,validation_data=(test_seq,test_labels))
lstm_score = lstm_classifier.evaluate(test_seq,test_labels)[1]
```

Candidate Algorithm Selection and Rationale

Prototype Analysis and Top-3 Candidate Selection

After running all the algorithms our general accuracy results are:

Accuracies

In [107]:
```
print("Logistic Regression Accuracy")
print("Count Vectorizer", format(logResscore,'.2%'))
print("TFIDF Vectorizer:", format(logRes_tfidf_score,'.2%'))
print("-----------")
print("Naive Bayes Accuracy")
print("Count Vectorizer", format(nbscore,'.2%'))
print("TFIDF Vectorizer:", format(nb_tfidf_score,'.2%'))
print("-----------")
print("Support Vector Machine Accuracy")
print("Count Vectorizer", format(svc_classifierscore,'.2%'))
print("TFIDF Vectorizer:", format(svc_classifier_tfidf_score,'.2%'))
print("-----------")
print("Decision Tree Accuracy")
print("Count Vectorizer", format(dec_treescore,'.2%'))
print("TFIDF Vectorizer:", format(dec_tree_tfidf_score,'.2%'))
print("-----------")
print("Random Forest Accuracy")
print("Count Vectorizer", format(rfscore,'.2%'))
print("TFIDF Vectorizer:", format(rf_tfidf_score,'.2%'))
print("-----------")
print("Deep Learning LSTM Accuracy")
print("Subword Text Encoder Accuracy",format(lstm_score,'.2%'))
print("-----------")
```

```
Logistic Regression Accuracy
Count Vectorizer 82.53%
TFIDF Vectorizer: 82.24%
-----------
Naive Bayes Accuracy
Count Vectorizer 71.91%
TFIDF Vectorizer: 73.36%
-----------
Support Vector Machine Accuracy
Count Vectorizer 80.93%
TFIDF Vectorizer: 83.41%
-----------
Decision Tree Accuracy
Count Vectorizer 77.87%
TFIDF Vectorizer: 75.98%
-----------
Random Forest Accuracy
Count Vectorizer 78.89%
TFIDF Vectorizer: 79.33%
-----------
Deep Learning LSTM Accuracy
Subword Text Encoder Accuracy 66.08%
-----------
```

From a quick look, it appears our top 3 candidates would be:

- SVM with TFIDF Vectorizer
- Logistic Regression with Count Vectorizer
- Deep Learning (LSTM)
    - Rationale: In our use case and significant number of features, even with lowest accuracy we should continue with Deep Learning. This while taking a long time to train using multiple epochs and layer parameters, should yield most scalable model with high likelihood of success for the final result when using different Tokenizer and Encoders.

Accuracy doesn't tell us everything, so lets review the confusion matrixes and classification report for the algorithms:

- In order to do this, we use several modules such as plot_confusion_matrix and classification_report from sklearn.metrics library

Confusion Matrices

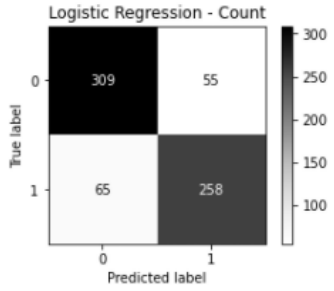
In [109]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

fig = plt.figure(figsize=(10,3))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
plot_confusion_matrix(logRes, X_test, y_test, cmap=plt.cm.Greys, ax=ax1)
plot_confusion_matrix(logRes_tfidf, X_test_tfidf, y_test, cmap=plt.cm.Greys, ax=ax2)
ax1.set_title("Logistic Regression - Count")
ax2.set_title("Logistic Regression - TFIDF")

fig = plt.figure(figsize=(10,3))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
plot_confusion_matrix(nb, X_test.toarray(), y_test, cmap=plt.cm.Purples, ax=ax1)
plot_confusion_matrix(nb_tfidf, X_test_tfidf.toarray(), y_test, cmap=plt.cm.Purples, ax=ax2)
ax1.set_title("Naive Bayes - Count")
ax2.set_title("Naive Bayes - TFIDF")

fig = plt.figure(figsize=(10,3))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
plot_confusion_matrix(svc_classifier, X_test.toarray(), y_test, cmap=plt.cm.Greens, ax=ax1)
plot_confusion_matrix(svc_classifier_tfidf, X_test_tfidf.toarray(), y_test, cmap=plt.cm.Greens, ax=ax2)
ax1.set_title("SVM - Count")
ax2.set_title("SVM - TFIDF")

fig = plt.figure(figsize=(10,3))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
plot_confusion_matrix(dec_tree, X_test.toarray(), y_test, cmap=plt.cm.Purples, ax=ax1)
plot_confusion_matrix(dec_tree_tfidf, X_test_tfidf.toarray(), y_test, cmap=plt.cm.Purples, ax=ax2)
ax1.set_title("Decision Tree Bayes - Count")
ax2.set_title("Decision Tree - TFIDF")

fig = plt.figure(figsize=(10,3))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
plot_confusion_matrix(rf, X_test, y_test, cmap=plt.cm.Oranges, ax=ax1)
plot_confusion_matrix(rf_tfidf, X_test_tfidf, y_test, cmap=plt.cm.Oranges, ax=ax2)
ax1.set_title("Random Forest - Count")
ax2.set_title("Random Forest - TFIDF")

print("Confusion Matrices -")
```

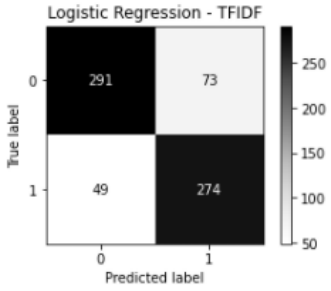
Other scores via Classification Report

In [113]:
```
from sklearn.metrics import classification_report

lr_pred = logRes.predict(X_test)
lr_pred_tfidf = logRes_tfidf.predict(X_test_tfidf)

nb_pred = nb.predict(X_test.toarray())
nb_pred_tfidf = nb_tfidf.predict(X_test_tfidf.toarray())

svc_pred = svc_classifier.predict(X_test.toarray())
svc_pred_tfidf = svc_classifier_tfidf.predict(X_test_tfidf.toarray())

dec_tree_pred = dec_tree.predict(X_test.toarray())
dec_tree_pred_tfidf = dec_tree_tfidf.predict(X_test_tfidf.toarray())

rf_pred = rf.predict(X_test.toarray())
rf_pred_tfidf = rf_tfidf.predict(X_test_tfidf.toarray())
```

In [114]:
```
print("Logistic Regression")
print("Count Vectorizer")
print(classification_report(y_test, lr_pred))
print("--")
print("TFIDF Vectorizer")
print(classification_report(y_test, lr_pred_tfidf))
print("----------------------")
print("Naive Bayes")
print("Count Vectorizer")
print(classification_report(y_test, nb_pred))
print("--")
print("TFIDF Vectorizer")
print(classification_report(y_test, nb_pred_tfidf))
print("----------------------")
print("SVM")
print("Count Vectorizer")
print(classification_report(y_test, svc_pred))
print("--")
print("TFIDF Vectorizer")
print(classification_report(y_test, svc_pred_tfidf))
print("----------------------")
print("Decision Tree")
print("Count Vectorizer")
print(classification_report(y_test, dec_tree_pred))
print("--")
print("TFIDF Vectorizer")
print(classification_report(y_test, dec_tree_pred_tfidf))
print("----------------------")
print("Random Forest")
print("Count Vectorizer")
print(classification_report(y_test, rf_pred))
print("--")
print("TFIDF Vectorizer")
print(classification_report(y_test, rf_pred_tfidf))
print("----------------------")
```

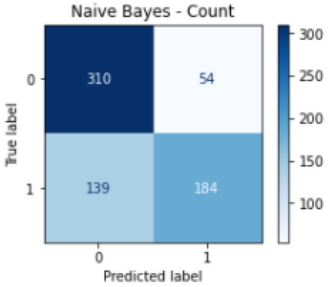
- **Logistic Regression:**

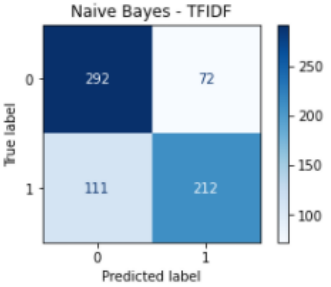





```
Logistic Regression
Count Vectorizer
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       364
           1       0.82      0.80      0.81       323

    accuracy                           0.83       687
   macro avg       0.83      0.82      0.82       687
weighted avg       0.83      0.83      0.83       687

--
TFIDF Vectorizer
              precision    recall  f1-score   support

           0       0.86      0.80      0.83       364
           1       0.79      0.85      0.82       323

    accuracy                           0.82       687
   macro avg       0.82      0.82      0.82       687
weighted avg       0.82      0.82      0.82       687

----------------------
```

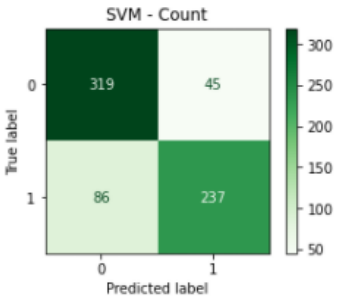
- **Naïve Bayes:**

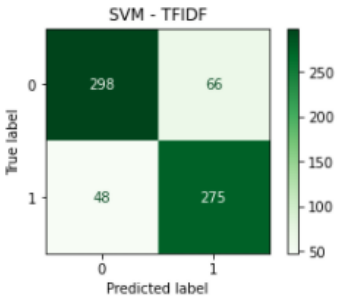





```
Naive Bayes
Count Vectorizer
              precision    recall  f1-score   support

           0       0.69      0.85      0.76       364
           1       0.77      0.57      0.66       323

    accuracy                           0.72       687
   macro avg       0.73      0.71      0.71       687
weighted avg       0.73      0.72      0.71       687

--
TFIDF Vectorizer
              precision    recall  f1-score   support

           0       0.72      0.80      0.76       364
           1       0.75      0.66      0.70       323

    accuracy                           0.73       687
   macro avg       0.74      0.73      0.73       687
weighted avg       0.73      0.73      0.73       687

----------------------
```

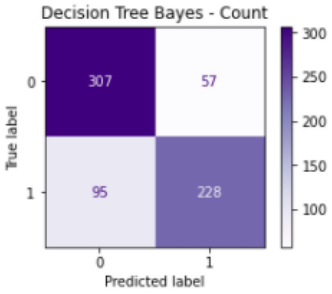
- **Support Vector Machine:**

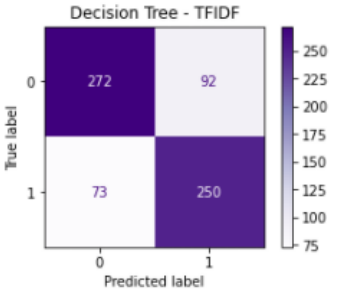





```
SVM
Count Vectorizer
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       364
           1       0.84      0.73      0.78       323

    accuracy                           0.81       687
   macro avg       0.81      0.81      0.81       687
weighted avg       0.81      0.81      0.81       687

--
TFIDF Vectorizer
              precision    recall  f1-score   support

           0       0.86      0.82      0.84       364
           1       0.81      0.85      0.83       323

    accuracy                           0.83       687
   macro avg       0.83      0.84      0.83       687
weighted avg       0.84      0.83      0.83       687
```

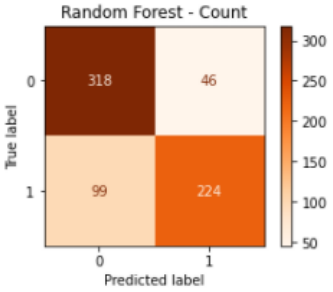
- **Decision Trees:**

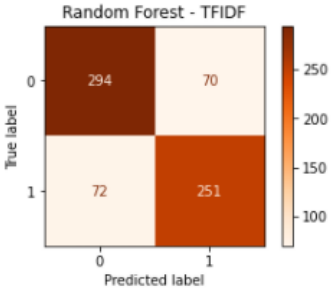





```
----------------------
Decision Tree
Count Vectorizer
              precision    recall  f1-score   support

           0       0.76      0.84      0.80       364
           1       0.80      0.71      0.75       323

    accuracy                           0.78       687
   macro avg       0.78      0.77      0.78       687
weighted avg       0.78      0.78      0.78       687

--
TFIDF Vectorizer
              precision    recall  f1-score   support

           0       0.79      0.75      0.77       364
           1       0.73      0.77      0.75       323

    accuracy                           0.76       687
   macro avg       0.76      0.76      0.76       687
weighted avg       0.76      0.76      0.76       687
```

- **Random Forest:**






```
----------------------
Random Forest
Count Vectorizer
              precision    recall  f1-score   support

           0       0.76      0.87      0.81       364
           1       0.83      0.69      0.76       323

    accuracy                           0.79       687
   macro avg       0.80      0.78      0.78       687
weighted avg       0.79      0.79      0.79       687

--
TFIDF Vectorizer
              precision    recall  f1-score   support

           0       0.80      0.81      0.81       364
           1       0.78      0.78      0.78       323

    accuracy                           0.79       687
   macro avg       0.79      0.79      0.79       687
weighted avg       0.79      0.79      0.79       687

----------------------
```

Reviewing the better precision, recall and f1-score, it appears our final selection of top 2 algorithms excluding LSTM would be:

- Support Vector Machines
- Logistic Regression

With current accuracies of 80% and above, we expect a significant growth after tweaking hyper-parameters.

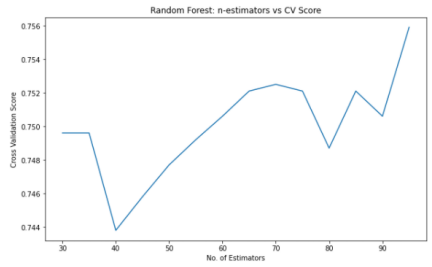
Cross Validation and Hyper-param Tuning

Before we go through an exhaustive cross-validation and tuning for our top 3 candidates, we can try applying simple cross validation to Random Forest and Decision Tree

- Random Forest

We try 10-fold cross validation against a bunch of n_estimator params

```
In [53]: #Performing cross validation
estimators = []
cv_scores_rf = []

#perform 10 fold cross validation
for k in range(30,100,5):
    estimators.append(k)
    dtc = RandomForestClassifier(n_estimators = k, random_state=42)
    scores = cross_val_score(dtc,X_train,y_train,cv=10, scoring = 'accuracy')
    cv_scores_rf.append(round(scores.mean(),4))

optimal_estimator = estimators[cv_scores_rf.index(max(cv_scores_rf))]

plt.figure(figsize = (10,6))
plt.title("Random Forest: n-estimators vs CV Score")
plt.plot(estimators, cv_scores_rf)
plt.xlabel('No. of Estimators')
plt.ylabel('Cross Validation Score')
plt.show()

print('The ideal n-estimator value is:', optimal_estimator)
print('Highest cross validation score we get is:', max(cv_scores_rf))
```



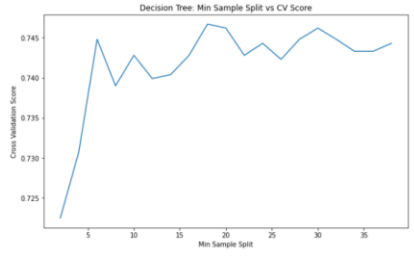

```
The ideal n-estimator value is: 95
Highest cross validation score we get is: 0.7559
```

It does not seem to make any impact for this algorithm, so we can keep the defaults providing us the accuracy of **79%**.

- Decision Tree

We try 10-fold cross validation against a bunch of sample_split param values

```
In [52]: #Decision Tree Cross Validation
sample_splits = []
cv_scores_dt = []

for k in range(2,40,2):
    sample_splits.append(k)
    dtc = tree.DecisionTreeClassifier(min_samples_split = k, random_state=42)
    scores = cross_val_score(dtc, X_train,y_train, cv=10, scoring = 'accuracy')
    cv_scores_dt.append(round(scores.mean(),4))

optimal_split_val = sample_splits[cv_scores_dt.index(max(cv_scores_dt))]

plt.figure(figsize = (10,6))
plt.title("Decision Tree: Min Sample Split vs CV Score")
plt.plot(sample_splits, cv_scores_dt)
plt.xlabel('Min Sample Split')
plt.ylabel('Cross Validation Score')
plt.show()

print('The ideal split value is:', optimal_split_val)
print('Highest cross validation score we get is:', max(cv_scores_dt))
```




```
The ideal split value is: 18
Highest cross validation score we get is: 0.7467
```

It does not seem to make any impact for this algorithm as well with our final accuracy being **75%**

For the **top 3 candidates**, we can now use RandomSearchCV module from sklearn.model_selection library to find the ideal mix of parameters, while verifying accuracy against a K-fold stratified cross-validator.

We use 10-fold cross validation with a bunch of parameter ranges to see the effect:

- Logistic Regression

Logistic Regression

In [131]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform

penalty = ['none', 'l1', 'l2', 'elasticnet']
solver=['newton-cg', 'lbfgs', 'liblinear', 'saga']
C = loguniform(1e-5, 100)
hyperparameters = dict(C=C, penalty=penalty,solver=solver)
logistic = linear_model.LogisticRegression()
rand_lr = RandomizedSearchCV(logistic, hyperparameters, scoring='accuracy', n_iter=200, cv=10, verbose=0, n_jobs=-1)
logRes_tuned = rand_lr.fit(X_train, y_train)

print('Best Score: %s' % logRes_tuned.best_score_)
print('Best Hyperparameters: %s' % logRes_tuned.best_params_)
```

```
Best Score: 0.7962314150368182
Best Hyperparameters: {'C': 2.923975011976315, 'penalty': 'l2', 'solver': 'newton-cg'}
```

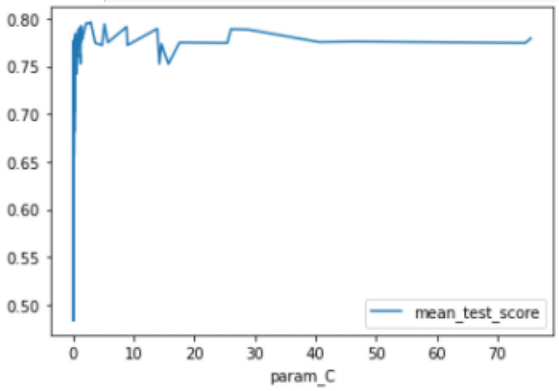
In [132]:
```
lr_pred_tuned = logRes_tuned.predict(X_test)
print(classification_report(y_test, lr_pred_tuned))
```

```
              precision    recall  f1-score   support

           0       0.83      0.84      0.83       364
           1       0.82      0.80      0.81       323

    accuracy                           0.82       687
   macro avg       0.82      0.82      0.82       687
weighted avg       0.82      0.82      0.82       687
```

For the results, we also plot the C param against the cross validation results to review its effect on the score

```
dftemp = pd.DataFrame(logRes_tuned.cv_results_)
```

In [258]:
```
dftemp[['param_C','mean_test_score']].drop_duplicates().sort_values(by='param_C').plot(x='param_C',y='mean_test_score')
```



We can see above, our accuracy did not improve from the cross validation, so we can safely use the best accuracy we received: **82%**

- SVM

```
kernel= ['linear','rbf'],
gamma= [1e-2,1e-1,1e+0,1e+1,1e+2],
C= [1e+0,1e+1,1e+2,1e+3,1e+4,1e+5,1e+6]
hyperparameters = dict(C=C, gamma=gamma,kernel=kernel)
svm_gen = svm.SVC()
rand_svm = RandomizedSearchCV(svm_gen, hyperparameters, scoring='accuracy', n_iter=100, cv=2, verbose=0, n_jobs=-1)
svm_tuned = rand_lr.fit(X_train_tfidf.toarray(), y_train)

print('Best Score: %s' % svm_tuned.best_score_)
print('Best Hyperparameters: %s' % svm_tuned.best_params_)
```

```
svm_pred_tuned = svm_tuned.predict(X_test_tfidf.toarray())
print(classification_report(y_test, svm_pred_tuned))
```

This, because of the way Support Vector Machine handles scaling did not yield a final result despite the limited set of params, lower cross validation and use of all CPUs with several hours of time dedicated on our personal machines.

Thus, we will be leaving this is as task to do on cloud systems. Referring to our original results,

	precision	recall	f1-score	support
0	0.86	0.82	0.84	364
1	0.81	0.85	0.83	323
accuracy			0.83	687
macro avg	0.83	0.84	0.83	687
weighted avg	0.84	0.83	0.83	687

We can conclude so far, the best accuracy we received: 83%

- LSTM

We ran into the same problem here as SVM, deep learning takes a very long time to run, and performing the model training several times with different parameters would take very long on a personal system.

Thus, because of limited time, we will be leaving this as a task to do before deployment.

For the purposes of the report, we can assume the best accuracy we received is same as before: 66%

Final Inference

After our analysis of the algorithms above, we can see how our candidates look:

- **Logistic Regression**:

```
Logistic Regression
Count Vectorizer
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       364
           1       0.82      0.80      0.81       323

    accuracy                           0.83       687
   macro avg       0.83      0.82      0.82       687
weighted avg       0.83      0.83      0.83       687
```

- **Support Vector Machine**:

```
              precision    recall  f1-score   support

           0       0.86      0.82      0.84       364
           1       0.81      0.85      0.83       323

    accuracy                           0.83       687
   macro avg       0.83      0.84      0.83       687
weighted avg       0.84      0.83      0.83       687
```

- **LSTM**:

```
lstm_classifier.evaluate(test_seq,test_labels)
22/22 [==============================] - 0s 11ms/step - loss: 1.4601 - accuracy: 0.6608
```

We can see our best algorithm so far is Support Vector Machines. Let us try a sample review and see how it performs:

Tina Busch

★☆☆☆☆ **Camera review**

Reviewed in the United States on January 7, 2008

Color: Pink bliss

very unhappy with the service. i bought this camera as a christmas gift. when it was opened and tried out, it was found that the camera did not work, it had vertical lines down the pictures taken. Since I had purchased it as a chrstmas gift, it had been past 15 days since the delivery date, so I am unable to return it. i am very unhappy with this service and i will never purchase anything off amazon again.

We copy the review text, clean up the stopwords and feed it to our classifier after transforming:

```
In [213]: def predict(text):
              text = ' '.join(basic_clean(text))
              text = vectorizer.transform([text])
              text = tfidf_transform.fit_transform(text)
              prediction = svc_classifier_tfidf.predict(text.toarray())
              if(prediction[0]=='0'): return 'Negative'
              else: return 'Positive'
```

```
In [17]: predict(input("Enter review here: "))
         Enter review here: very unhappy with the service. i bought this camera as a christmas gift. when it was opened and tried out, it was found that the camera did not work, it h
         ad vertical lines down the pictures taken. Since I had purchased it as a chrstmas gift, it had been past 15 days since the delivery date, so I am unable to return it. i am v
         ery unhappy with this service and i will never purchase anything off amazon again.
Out[17]: 'Negative'
```

Our model returned **Negative**, which is the valid output.

It appears to be working on a random review, let us try another one:

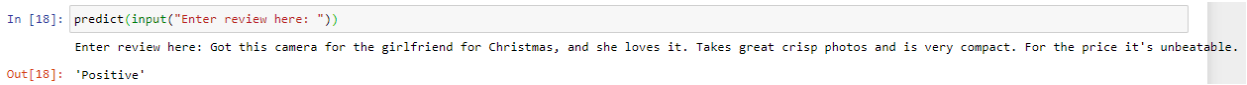
Elliot Brazitis

★★★★★ **Awesome Camera**

Reviewed in the United States on January 7, 2008

Color: Midnight black

Got this camera for the girlfriend for Christmas, and she loves it. Takes great crisp photos and is very compact. For the price it's unbeatable.



Our model returned **Positive**, which is the valid output again!

While we can do further validation by scraping the reviews from the internet, it appears for a few random samples our predictions are valid. Therefore, next step while deployment would be to fetch more reviews by either scraping or connecting to the API of different social media websites and validate the output.

Threats to Validity

While we can review accuracy, and other statistical identifiers to review how our model is performing and work on making it better, there are several other innate threats to validity which can affect our model in the long run.

We can identify multiple threats to sentiment mining external validity as:

- a mismatch in demographics of the reviewers sample
- bias due to reviewers' incidental experiences
- manipulation of reviews.

In order to minimize these, further research would be necessary to make sure:

- reviewer samples are from varied demographics,
- do not include manipulated reviews

The internal reviewer bias, unfortunately, cannot be removed practically on our end, since every comment contains some amount of subjectivity which will always affect the prediction. However, our main purpose of resolving thousands of interviews quickly with a single click can be done using this model, which can then be improved upon by using newer techniques in the future.

Additional Functionality

In addition to our core idea of sentiment analysis for classification, with our data we have another interesting function available:

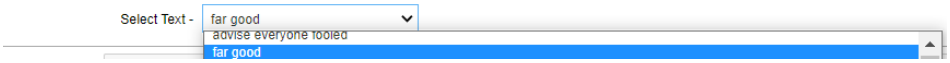
Finding similar reviews based on a given review:

- We can do this by using *Cosine similarity*:
    - Cosine Similarity measures the similarity between two vectors of an inner product space. Mathematically, it is the cosine of the angle between two vectors and determines whether two vectors are pointing in similar direction.
    - To do this we can use the inbuilt cosine_similarity function in sklearn library

```
In [117]: #cosine similarity based ranking on tfidf vector
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vec = TfidfVectorizer(ngram_range=(1,3))
trsfm=tfidf_vec.fit_transform(clean_sentences)
```

```
In [118]: from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
def get_similar(index):
    cos_arr = cosine_similarity(trsfm[index], trsfm)
    i = np.unique(cos_arr)[-2]
    return (clean_sentences[np.where(cos_arr == i)[1][0]], i)

class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'
```

```
In [120]: from IPython.display import *
import ipywidgets as widgets
from ipywidgets import *

options_sent = []
index=0
for i in clean_sentences:
    options_sent.append((i,index))
    index+=1
widget1 = Dropdown(options=options_sent,
    value=10,
    description='Select Text -',)
display(widget1)
```




```
In [124]: index = widget1.value
print("Given Sentence:", color.PURPLE + color.BOLD + clean_sentences[index] + color.END)
print("Most similar sentence:", color.RED + color.BOLD + get_similar(index)[0] + color.END,
      "\nCosine Similarity:",  color.BLUE + color.BOLD + str(round(get_similar(index)[1],3)*100), '%')
```

```
Given Sentence: far good
Most similar sentence: weeks far good
Cosine Similarity: 60.099999999999994 %
```

We can see above, providing a given sentence gave us a similar sentence in addition to the similarity percentage.

In our deployed website, we will extend this functionality to allow similar review searches.

References

- Datalab.uci.edu. 2020. [online] Available at: <http://www.datalab.uci.edu/papers/kdd2015_dimitris.pdf> [Accessed 3 December 2020].
- Monkeylearn.com. 2020. [online] Available at: <https://monkeylearn.com/blog/sentiment-analysis-of-product-reviews/> [Accessed 3 December 2020].
- Scikit-learn.org. 2020. *API Reference — Scikit-Learn 0.23.2 Documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/classes.html> [Accessed 3 December 2020].
- Fontanella, C., 2020. *The Best 12 Sentiment Analysis Tools In 2020*. [online] Blog.hubspot.com. Available at: <https://blog.hubspot.com/service/sentiment-analysis-tools> [Accessed 3 December 2020].
- Gartner. 2020. *Key Findings From The Gartner Customer Experience Survey*. [online] Available at: <https://www.gartner.com/en/marketing/insights/articles/key-findings-from-the-gartner-customer-experience-survey> [Accessed 3 December 2020].
- PyPI. 2020. *Pyldavis*. [online] Available at: <https://pypi.org/project/pyLDAvis/> [Accessed 3 December 2020].
- Nltk.org. 2020. *Python Module Index — NLTK 3.5 Documentation*. [online] Available at: <https://www.nltk.org/py-modindex.html> [Accessed 3 December 2020].
- Kaggle.com. 2020. *Sentiment Labelled Sentences Data Set*. [online] Available at: <https://www.kaggle.com/marklvl/sentiment-labelled-sentences-data-set> [Accessed 3 December 2020].
- Team, K., 2020. *Keras Documentation: Keras API Reference*. [online] Keras.io. Available at: <https://keras.io/api/> [Accessed 3 December 2020].
- Amueller.github.io. 2020. *Wordcloud For Python Documentation — Wordcloud 1.8.1 Documentation*. [online] Available at: <https://amueller.github.io/word_cloud/> [Accessed 3 December 2020].
- Amazon.com. 2020. *Awesome Camera.* [online] Available at: <https://www.amazon.com/gp/customer-reviews/R1MKFEY22C8V3G/ref=cm_cr_getr_d_rvw_ttl?ie=UTF8&ASIN=B000OQG9Y6> [Accessed 3 December 2020].
- Amazon.com. 2020. *Camera review.* [online] Available at: <https://www.amazon.com/gp/customer-reviews/R1WMJ59X5EMDS8/ref=cm_cr_getr_d_rvw_ttl?ie=UTF8&ASIN=B000OQG9Y6> [Accessed 3 December 2020].