Text Mining and Sentiment Analysis

Business Objective:

Text mining is the process of examining large collections of text and converting the unstructured text data into structured data for further analysis like visualization and model building. We will utilize the power of text mining to do an in-depth analysis of customer reviews on a cell phone model. Customer reviews are a great source of “Voice of customer” and could offer tremendous insights into what customers like and dislike about a product. As an investigative tool, TM is capable of supplying insights and knowledge from social media data, of a company. It is able to reveal the competitor strategy, by mining their Facebook or Twitter, for information on how customer engagement, promotion of services and customer bonding should be conducted.

The Project is about applying Text mining and sentiment Analysis to user generated content from distinct sources, in the cell phones domain, with the purpose of detection and discovery of defects in a product.

The business value is derived as an early discovery of defects can help manufacturers in quality improvement, thus minimize selling and production of defective product units. Meaning, customer dissatisfaction is decreased as well as the warranties costs and defect-associated costs.

Our main objective is to gain insights and knowledge by applying TM on data gathered from online forums on reviews, related to cell phones. Revealing how different entities (brands), by their frequency in the discussions, co-occur in the data. The analysis showcase's the ability to quantify what consumers wrote about each Phone Model, being externally validated with survey data. The Project shows TM value in capable of specifying a sentiment e.g. “problem”, and its co-occurrence with terms like camera, battery for the brand Samsung or any other brand. Applying a similar approach could be used to identify ways of quality improvement, by using the technology for quality control, identifying what aspects of a cell phone (or product) that lead to customers’ disapproval. By applying Applied Analysis techniques to identify and monitor underlying sentiments in text, written by consumers in reviews. The result is the capability of monitoring emotion such as surprise, anger, fear or sadness, on events or topics in the project. The findings give implications of benefits for use in large scale projects, to highlight issues needing to be addressed, for quality assurance

User Story:

As TM reveal how the words, as they appear in a review and in what context, lead to certain ratings, Mobile Phone brands can gain the insights and knowledge about their products. The analysis therefore suggests, what the factors leading to better satisfaction, for a phone experience, are. Here, for example, battery life was an important factor to high satisfaction rating, and words related to features. Knowing what factors tacitly lead to certain ratings is of high business value, as the knowledge lets the Brand know exactly what they could improve for better customer relationships (also service improvement).

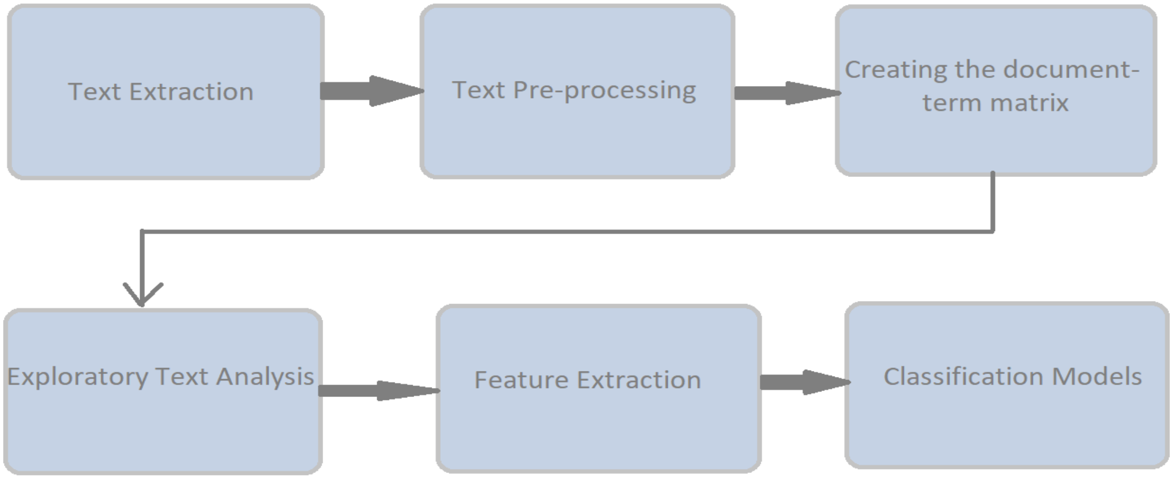
Underlying data science problem:

Our vision for this project is to work with opinion driven resources such as online reviews and blogs. Our project focuses on text mining and sentiment analysis where the problem drills down to classification for modern data scientists in the form of factual (objective) based and opinion (subjective) based classification. Primarily, the focus will be on solving the problem of recommending the right product to customers pertaining to cell phone devices.

The aim is to use text from surveys/reviews and determine likelihood of a product being recommended by a customer. The model will be such that it will follow a supervised machine learning technique with being labelled with a positive or negative sentiment being trained on some test data sample.
This further supports the concept of predicting user experience which is one of the known and popular data science problem that majority data scientists are working to optimize.

Tools and Techniques:

About the data set:
The dataset that we will be using for the project is from Kaggle and is about different mobile phone models, revolving around the reviews written by customers. This data set includes 7 csv data files, which have a total of 1.4 million user ratings and reviews for different brands of cell phones. Each row corresponds to a customer review, and includes the variables: phone_url, date, lang, country, source, domain, score, score_max, extract, author and product.
We'll be using the Classification Models to understand various aspects of text mining, that are based on the review text as the independent variable to predict whether a customer recommends a product. The focus will be to understand differences between customers who recommend a product and those who don't. Text Mining includes the following steps:



Step 1: Text Extraction

Sklearn.model_selection is package in python for splitting data into training and test sets. CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.

Step 2: Text Pre-processing:

Text pre-processing includes data cleanup techniques and transformation of the documents in the training data to a document-term matrix.

Step 3: Document-Term-Matrix:

A document-term matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. The pre-processed and cleaned up data is converted into a matrix called the document term matrix.

Step 4: Exploratory text analysis:

We can use TFIDFVectorizer for word frequency scores that try to highlight words that are more frequent in a document. We will then look at how to create bi-grams and tri-grams and perform some exploratory analysis on the same.

n-gram: All the analysis that we have done so far have been based on single words that are called as Unigrams. However, it can be very insightful to look at multiple words. This is called as N-grams in text mining, where N stands for the number of words. For example, bi-gram contains 2 words.

Step 5: Feature Extraction

The exploratory text analysis has given several insights based on the customer reviews. In terms of classification algorithms used, there is not much of a difference between data and text input. We will try 3 of the most popular classification algorithms — XGBOOST, Random forest and logistic regression.

Tokenisation: Tokenisation is the process of decomposing text into distinct pieces or tokens. Once tokenisation is done, after all the pre-processing, it is possible to construct a dataframe where each row represents a document and each column represents a distinct token and each cell gives the count of the token for a document.

Step 6: Building the Classification Models

In this step, we will have the document frequency matrix which is pre-processed, treated and ready to be used for classification.

Implementation:

Data Set:

This data set includes 4 csv data files, which have user ratings and reviews for different brands of cell phones.

The Data set is taken from the link below and uploaded on S3 bucket.

Link : https://www.kaggle.com/masaladata/14-million-cell-phone-reviews

Columns in Data set:

Phone_url : url from where the reviews are taken
Date: date of review.
Lang : Language of the given review.
Country : country name where user belongs
Source : source from where reviews are taken.
Domain : website
Score : ratings
Score_max : maximum rating
Extract: reviews on which text mining is performed.
Product : cell phone model

Setting Storage buckets:

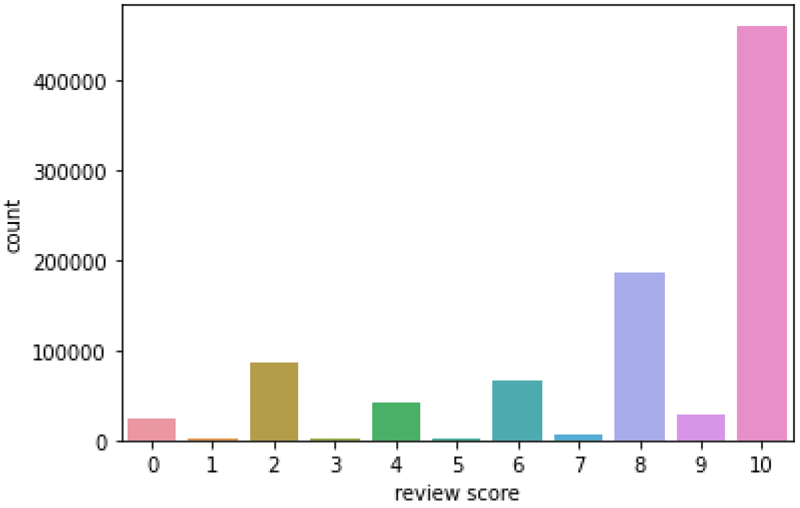
For accessing aws service, we need access key and secret key.
There a really good python package called Boto3 which let us work with every aws service.
S3 : S3 is Amazon’s storage bucket.
Basic Principles:
Download amazon security credentials from Amazons dashboard.
Save the credentials in a variable/environment variables.
Create s3 client with the help of Boto3.client()
S3 clients gets connected tio my s3_client.

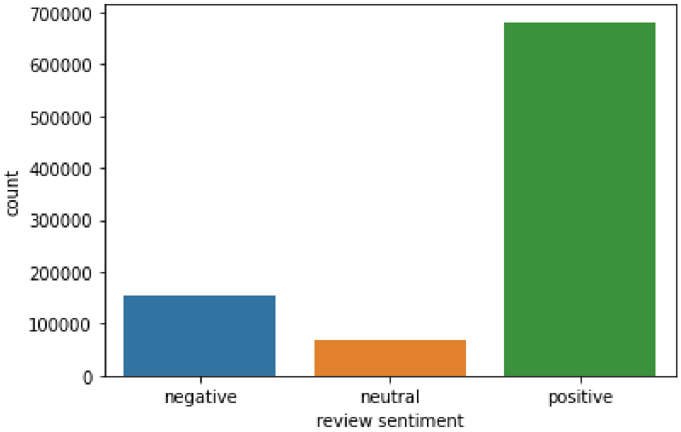
```
s3_client = boto3.client('s3',  aws_access_key_id=access_key, aws_secret_access_key=secret_key)
s3 = boto3.resource('s3', aws_access_key_id=access_key, aws_secret_access_key=secret_key)

for bucket in s3.buckets.all():
print(bucket.name)
```

```
csv_file_list = ["s3://arsalanmubeenbucket/phone_user_review_file_1.csv",
"s3://arsalanmubeenbucket/phone_user_review_file_2.csv",
"s3://arsalanmubeenbucket/phone_user_review_file_3.csv",
"s3://arsalanmubeenbucket/phone_user_review_file_4.csv"]
```

Here, we have appended all CSVs in one list.
Some EDA stuff:
NaN count in score(graph)





Positive, Negative, Neutral score count
Removing NaNs
Converting scores(ratings) into Negative, Positive and Nuetral Labels (by assigning 0 to Negative, 1 to Neutral and 2 to Positive)
Plot a graph of newly created labels(classes) using seaborne library
Fetching Data using SQL:The expression below is used to fetch languages and there corresponding number of reviews/comments.

```
output = sqldf("select lang as Languages ,count(1) as Counts_of_Comments from merged_df group by lang")
```

Working with only English Reviews

```
df_ENG=merged_df.loc[merged_df['lang'] == 'en']
```

Using a function that checks the language is English in actual.

```
def f(row):
String = row['extract']
try:
val = detect(String)
except TypeError:
String= re.sub("[^a-zA-Z]"," ",str(String))
val = detect(String)
except :
print(String)
val = "nan"
return val
```

deleting outliers

Text Preprocessing:

It includes the following
Converting words in reviews column in lower case
Removing Punctuations
Lammizatation: converting a word in its route form
Removing stopwords
Bag of Words
Train and Test Split
Using Bert/Naïve Bayes.

Naive Bayes:

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

Likelihood

Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

Posterior Probability

Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
P(c) is the prior probability of class.
P(x|c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

Splitting into train and test using sklearn.model's train_test_split API:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0).to(device)
```

Applying model:

```
from sklearn.naive_bayes import MultinomialNB
```

Accessing Multinomial function:

```
classifier=MultinomialNB()
```

Fitting data in x and y train and checking for accuracy:

```
classifier.fit(X_train, y_train)
pred = classifier.predict(X_test)
score = metrics.accuracy_score(y_test, pred)
score
```

Bert:

BERT (Bidirectional Encoder Representations from Transformers) is a recent paper("Attention is All You Need") published by researchers at Google AI Language.

BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible.

So, we are using the version of BERT which is the base model with the 12 transformer encoders. We have used cased version because the casing of words matters when we are doing sentiment analysis for reviews.

```
PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
```

Sentiment Classification with BERT:

Given that we used the Hugging face library and initialize the BERT model from that.

```
bert_model = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

Calling the resulting model using input_ids and attention mask. And this will return two things:
Last hidden state from the top-most encoder.
Pooled output

```
last_hidden_state, pooled_output = bert_model(
input_ids=data['input_ids'],
attention_mask=data['attention_mask']
)
```

Building Sentiment Classifier:

We created a new class from nn module in which we are going to accept number of classes that we are going to have and have to feed data to our BERT model. We have creted a dropout layer and specify probability of dropout as 0.3and I have specified number of hidden_size unit and number of classes. This is the way we can use the BERT model.

Next , we defined a forward function implementing the base forward and get pooled output from our BERT model.

Next, we got the output from the dropout then applied the output layer classifier.

```
class SentimentClassifier(nn.Module):

def __init__(self, n_classes):
super(SentimentClassifier, self).__init__()
self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
self.drop = nn.Dropout(p=0.3)
self.out = nn.Linear(self.bert.config.hidden_size, n_classes)

def forward(self, input_ids, attention_mask):
_, pooled_output = self.bert(
input_ids=input_ids,
attention_mask=attention_mask,
return_dict=False
)
output = self.drop(pooled_output)
return self.out(output)
```

Creating instance of classifier by taking length of the class name.

```
model = SentimentClassifier(len(class_names))
```

Move model to GPU device.

```
model = model.to(device)
```

This data contains input_ids and attention masks. Moving bopth on GPU device.

```
input_id = data['input_ids'].to(device)
attention_masks = data['attention_mask'].to(device)
```

The shape of the input_ids and attention_mask should correspond to the batch size which is 16 and the number of tokens is 100.

```
print(input_ids.shape) # batch size x seq length
print(attention_mask.shape)
```

Running our sentiment Classifier.It will show probability of each class.

```
outputs  =  model(input_id, attention_masks)
```

Building Training:

AdamW is an optimizer which is an algorithm of weight decay fix. And we passed it in the model parameters along with learning rate which is 2e-5 which goes good on fine-tuning the BERT model. Then we used linear scheduler and for that we calculated the stepsand number of batches we have in train_data_loader times the number of epochs that is 10 in our case. Using that we created scheduler which is provided by hugging face library. By passing in the optimizer and total steps , we got the loss function which is going to be Cross Entropy because we are doing classification and moved it over GPU.

```
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS
scheduler = get_linear_schedule_with_warmup(
optimizer,
num_warmup_steps=0,
num_training_steps=total_steps
)
loss_fn = nn.CrossEntropyLoss().to(device)
```

Train_epoch function: The function is going to get over the training data atleast for 1 epoch and used that to apply BERT propagation for sentiment classifier. The function accepts the model, dataloader, loss function, optimizer, device, scheduler, n examples because we will calculate the loss and the training accuracy.

Put the model in training mode so that our dropout is actually applied at the training process
Store loss and correct predictions
Iterate each example in data loader
Take input_ids , attention_masks and the targets of the labels for which we are predicting the sentiment.
Move input_ids, attention_mask and targets over GPU
Take outputs of the model and attention mask
Take predictions that are going to be the classes that have highest probability with torch.max along the one dimension.
Apply loss function over the model outputs
Use targets to calculate number of correct predictions by using torch.sum
Get all the predictions that are same as targets
Append the losses
Apply backward propagation

Do some Gradient clipping on our model by using normalization of 1 so this is basically duplicating the steps that the original paper has described during fine-tuning the BERT model. For instance. If youir gradient is too large, your training procedure will go unstable.
Optimization
Making gradience equal to zero
Last thing is the number of correct predictions and loss for this training epoch, we are going to take predictions and divide by number of example and take loss to calculate mean loss.

```
def train_epoch(
model,
data_loader,
loss_fn,
optimizer,
device,
scheduler,
n_examples
):
model = model.train()

losses = []
correct_predictions = 0

for d in data_loader:
input_ids = d["input_ids"].to(device)
attention_mask = d["attention_mask"].to(device)
targets = d["targets"].to(device)

outputs = model(
input_ids=input_ids,
attention_mask=attention_mask
)

_, preds = torch.max(outputs, dim=1)
loss = loss_fn(outputs, targets)

correct_predictions += torch.sum(preds == targets)
losses.append(loss.item())

loss.backward()
nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
optimizer.step()
scheduler.step()
optimizer.zero_grad()

return correct_predictions.double() / n_examples, np.mean(losses)
```

Evaluating the current model:

The function is going to get over the evaluation data loader.

Creating training Loop:

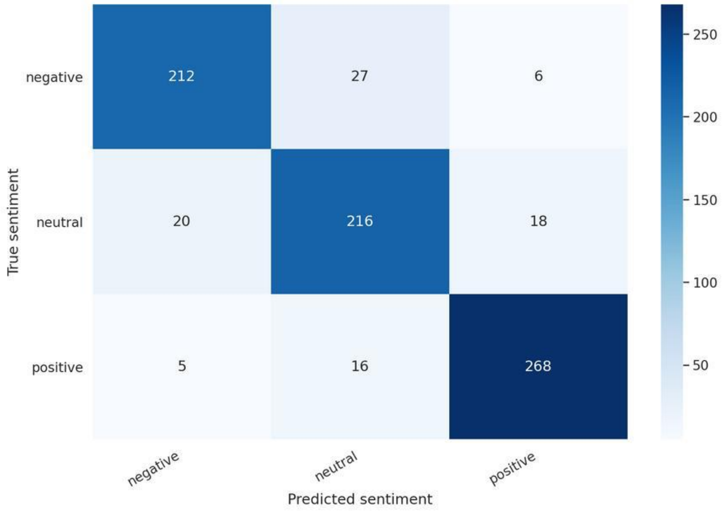
We are going to create training loop with the help of training epoch and evaluate functions. Here we will define dictionary containing lists and name it 'history'.

Evaluation:

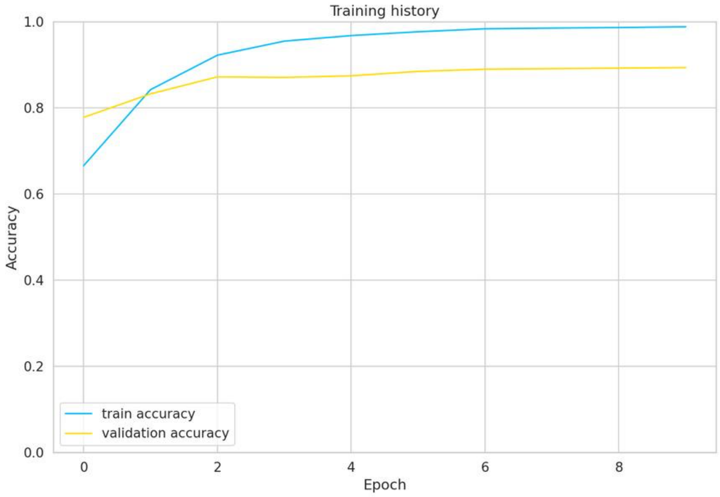
Evaluation is performed on test data loader. We got test accuracy and test loss by calling evaluate model.
So the accuracy is 79% from the test set which is quit good.
Classification report Evaluation using scikit library:
The precision of Negative and positive reviews have high/low precision.

Confusion Matrix & Training History :

Our model is doing quite well on Negative , Neutral and Positive reviews.
Positive Sentiment :
Total Positives = 324
Correctly Predicted Positives = 268
Negative Sentiment:
Total Negatives = 245
Correctly Predicted Negatives = 212
Neutral Sentiment :
Total Neutrals = 254
Correctly Predicted Neutrals = 216



The blue line in the training history shows increasing train accuracy of the model.


Training history
Accuracy
Epoch
train accuracy
validation accuracy