**DEVELOPMENT PART-2 DOCUMENT**

Let's dive into the next steps of the project after the design thinking phase. This is where we will continue building the sentiment analysis solution by performing various activities, including feature engineering, model training, and evaluation.

**FEATURE ENGINEERING**

**OBJECTIVE** -Enhance the data and prepare it for model training by creating new features and optimizing the existing ones.

**N- GRAMS**- Consider using n-grams (e.g., bigrams, trigrams) to capture more context and phrases in the text data

**FEATURE SELECTION** -Identify and select the most relevant features that contribute to sentiment analysis.

**EMITICONS AND EMOJI ANALYSIS** - Explore the sentiment conveyed by emoticons and emojis in customer feedback. Convert them into meaningful features.

**MODEL TRAINING**

**OBJECTIVE-**Train machine learning or deep learning models to predict sentiment.

**SELECT ALGORITHMS -**Choose suitable algorithms (e.g., Logistic Regression, Random Forest, LSTM, Transformer) for sentiment classification. You may want to experiment with multiple models to find the best performer.

**SPLIT DATA -**Divide the dataset into training, validation, and testing sets. This ensures that you can evaluate the model's performance effectively.

**HYPERPARAMETER TUNING**-Optimize model hyperparameters to achieve the best performance. Use techniques like grid search or random search to fine-tune the model.

**MODEL EVALUATION**

**OBJECTIVE** -Assess the performance of the sentiment analysis models and understand how well they predict sentiment.

**METRICES**-Use appropriate evaluation metrics (e.g., accuracy, precision, recall, F1-score) to measure the model's performance. Choose metrics that are relevant to your specific business objectives.

**CROSS- VALIDATION-**Employ cross-validation techniques (e.g., k-fold cross-validation) to ensure the model's generalization and robustness.

**CONFUSION MATRIX ANALYSIS -**Analyze the confusion matrix to understand where the model is making errors (e.g., false positives, false negatives).

**INSIGHTS GENERATION**

**OBJECTIVE-** Dive deeper into insights by considering the model's predictions and performance.

**FEATURE IMPORTANCE-** Analyze which words, phrases, or features contribute the most to sentiment predictions. This helps you understand what drives customer sentiments.

**MISCLASSIFICATION ANALYSIS-**Examine misclassified reviews to identify common reasons for misclassification. This can lead to further improvements in the model.

**RECOMMENDATIONS-**Refine and update your recommendations based on the model's predictions and insights.

**DEPLOYMENT**

**OBJECTIVE**- Deploy the sentiment analysis model for real-time or batch analysis.

**API OR WEB APP**-Develop an API or web application that allows users to submit customer feedback and receive sentiment analysis results in real-time.

**SCALABILITY** -Ensure that the deployed solution can handle a large volume of data efficiently.

**MONITORING** -Implement monitoring and feedback loops to continuously improve the model and its performance in a production environment.

**FEEDBACK LOOP**

**OBJECTIVE**-Continuously improve the sentiment analysis system based on user feedback and changing customer sentiments.

**FEEDBACK GATHERING** -Collect feedback from users and business stakeholders regarding the accuracy and usefulness of the sentiment analysis.

**MODEL UPDATES**-Make periodic updates to the model based on feedback and changing customer sentiments to maintain its relevance and accuracy.

Let’s continue building the sentiment analysis solution by elaborating on the "Employing NLP techniques" and "Generating Insights" sections.

**EMPLOYING NLP TECHNIQUES**

**OBJECTIVE**-Implement Natural Language Processing (NLP) techniques for sentiment analysis on customer feedback.

**BAG OF WORDS(BOW)**

**DESCRIPTION** -The Bag of Words (BoW) technique creates a representation of text data by counting the frequency of words in each document. It's a basic but effective approach.

**IMPLEMENTATION**

- Create a vocabulary of unique words in the dataset.

- Count how many times each word in the vocabulary appears in each customer review.

- Use these word counts as features for sentiment analysis.

**WORD EMBEDDINGS**

**DESCRIPTION -**Word embeddings are pre-trained models that capture the semantic meaning and relationships between words.

**IMPLEMENTATION**

- Utilize pre-trained Word2Vec or GloVe models to convert words in customer reviews into vectors.

- Calculate the vector representation for each review by averaging the vectors of individual words.

- Use these vectors as features for sentiment analysis.

**TRANSFORMER MODELS**

**DESCRIPTION** -Transformer models, like BERT and GPT, are advanced deep learning models that understand context and relationships in text.

**IMPLEMENTATION**

- Fine-tune a pre-trained transformer model for sentiment analysis on your dataset.

- Use the model to predict the sentiment (positive, negative, neutral) of each customer review.

**GENERATING INSIGHTS**

**OBJECTIVE**- Extract meaningful insights from the sentiment analysis results.

**COMPETITOR ANALYSIS**

**DESCRIPTION -**Compare the sentiment scores of competitor products to identify strengths and weaknesses.

**IMPLEMENTATION**

- Calculate the average sentiment scores for each competitor product.

- Identify which products receive the most positive and negative feedback.

- Use these insights to understand market sentiment towards different products.

**CUSTOMER FEEDBACK TRENDS**

**DESCRIPTION** -Identify recurring themes or issues in customer feedback that require attention.

**IMPLEMENTATION**

- Use topic modeling techniques (e.g., Latent Dirichlet Allocation or LDA) to discover common topics in customer reviews.

- Analyze which topics are frequently mentioned and their associated sentiment.

- Identify areas where customers are most dissatisfied or satisfied.

**RECOMMENDATIONS**

**DESCRIPTION** -Provide actionable recommendations for product improvement or marketing strategies based on the insights gained.

**IMPLEMENTATION**

- Use the insights from competitor analysis and customer feedback trends to formulate specific recommendations.

- Provide actionable suggestions for improving product features, addressing recurring issues, or tailoring marketing campaigns to align with customer sentiment.

# **Feature Engineering**

Use TfidfVectorizer to convert a collection of raw documents to a matrix of TF-IDF features.

In [39]:

from sklearn.feature\_extraction.text import TfidfVectorizer

In [40]:

vectorizer = TfidfVectorizer(stop\_words='english')

In [41]:

X\_train = vectorizer.fit\_transform(X\_train)

X\_test = vectorizer.transform(X\_test)

In [42]:

X\_train

Out[42]:

<13006x12004 sparse matrix of type '<class 'numpy.float64'>'

with 112877 stored elements in Compressed Sparse Row format>

In [43]:

X\_test

Out[43]:

<1446x12004 sparse matrix of type '<class 'numpy.float64'>'

with 11710 stored elements in Compressed Sparse Row format>

# **Grid Search & Cross Validation**

**Create a grid search function with cross validation.**

In [44]:

from sklearn.experimental import enable\_halving\_search\_cv

from sklearn.model\_selection import HalvingGridSearchCV

In [45]:

def grid\_search(model, parameters):

*# Use f1\_weighted as scoring since we already know that the dataset has imbalance labels*

grid = HalvingGridSearchCV(estimator=model, param\_grid=parameters,factor=2, cv=5,

scoring='f1\_weighted',random\_state=42,error\_score=0)

grid.fit(X\_train, y\_train)

print('Best Score : ',grid.best\_score\_)

print('Best parameters : ',grid.best\_params\_)

# **Logistic Regression Model**

**Create a base model of logistic regression, then perform a grid search to find the best parameters for final model.**

In [46]:

from sklearn.linear\_model import LogisticRegression

In [47]:

logreg = LogisticRegression(class\_weight='balanced')

In [48]:

logreg\_param = [{'penalty':['l1','l2','none'],

'solver': ['liblinear', 'saga'],

'C':[0.001,0.01,0.1,1,10],

'multi\_class':['auto','ovr','multinomial']},

{'penalty':['l2','none'],

'solver': ['sag','newton-cg','lbfgs'],

'C':[0.001,0.01,0.1,1,10],

'multi\_class':['auto','ovr','multinomial']}]

In [49]:

*# grid\_search(logreg,logreg\_param)*

Use the grid search results - best parameters for final logistic regression model.

In [50]:

# logreg\_model = LogisticRegression(class\_weight='balanced',C=0.1,multi\_class='multinomial',penalty='l2',solver='sag')

# **SVC Model**

**Create a base model of Support Vector Classifier, then perform a grid search to find the best parameters for final model.**

In [56]:

from sklearn.svm import SVC

In [57]:

svc = SVC(class\_weight='balanced')

In [58]:

svc\_param = {'C':[0.001, 0.01, 0.1, 1, 10],

'kernel':['linear','poly','rbf','sigmoid'],

'gamma':['scale','auto',0.001, 0.01, 0.1, 1, 10]}

In [59]:

*# grid\_search(svc, svc\_param)*

Use the grid search results - best parameters for final SVC model.

In [60]:

SVC\_model = SVC(class\_weight='balanced',C=1,gamma=1,kernel='linear')

**Naive Bayes Model**

The Naive Bayes model here will use default parameters.

In [61]:

from sklearn.naive\_bayes import BernoulliNB, GaussianNB, MultinomialNB, ComplementNB

In [62]:

BNB\_model = BernoulliNB()

GNB\_model = GaussianNB()

MNB\_model = MultinomialNB()

CNB\_model = ComplementNB()

NB\_param = {}

In [63]:

grid\_search(BNB\_model,NB\_param)

Best Score : 0.6826802304531071

Best parameters : {}

In [64]:

grid\_search(MNB\_model,NB\_param)

Best Score : 0.5924478934755192

Best parameters : {}

In [65]:

grid\_search(CNB\_model,NB\_param)

Best Score : 0.7369253091498471

Best parameters : {}

In [66]:

grid2 = HalvingGridSearchCV(estimator=GNB\_model, param\_grid=NB\_param,factor=2, cv=5,

scoring='f1\_weighted',random\_state=42,error\_score=0)

grid2.fit(X\_train.toarray(), y\_train)

print('Best Score : ',grid2.best\_score\_)

print('Best parameters : ')

print(grid2.best\_params\_)

Best Score : 0.5148436062659207

Best parameters :

{}

Based on the results, MNB\_model and GNB\_model did not perform really well (with f1\_score below 0.6). So, I will not use them for final evaluation. I will use BNB\_model and CNB\_model instead.

In [67]:

BNB\_model = BernoulliNB()

# CNB\_model = ComplementNB()

# **Decision Tree Model**

**Create a base model of decision tree, then perform a grid search to find the best parameters for final model.**

In [68]:

from sklearn.tree import DecisionTreeClassifier

In [69]:

DecTree = DecisionTreeClassifier(class\_weight='balanced', random\_state=42)

In [70]:

DT\_param = {'criterion':['gini','entropy','log\_loss'],

'max\_features':['sqrt','log2', None],

'max\_depth':[None,5,6,7,8,9,10,11,12,13,14,15]}

In [71]:

*# grid\_search(DecTree,DT\_param)*

Use the grid search results - best parameters for final Decision Tree model.

In [72]:

DTC\_model = DecisionTreeClassifier(criterion='gini',max\_features=None,max\_depth=None,class\_weight='balanced', random\_state=42)

**Bagging Classifier Model**

Bagging Classifier Model will use final decision tree model (DTC\_model) as base estimator.

In [78]:

from sklearn.ensemble import BaggingClassifier

In [79]:

Bagging = BaggingClassifier(base\_estimator=DTC\_model,random\_state=42)

In [80]:

bag\_param = {'n\_estimators':[50,100,150,200,250,300,400],

'bootstrap':[True,False]}

In [81]:

*# grid\_search(Bagging,bag\_param)*

Use the grid search results - best parameters for final Bagging Classifier model.

In [82]:

### Bagging\_model = BaggingClassifier(base\_estimator=DTC\_model,random\_state=42,bootstrap=True,n\_estimators=300)

### **Bag of words (BOW) feature extraction**

from sklearn.feature\_extraction.text import CountVectorizer

#from sklearn.feature\_extraction.text import TfidfVectorizer

vocabulary\_size = 5000

# Tweets have already been preprocessed hence dummy function will be passed in

# to preprocessor & tokenizer step

count\_vector = CountVectorizer(max\_features=vocabulary\_size,

# ngram\_range=(1,2), # unigram and bigram

preprocessor=lambda x: x,

tokenizer=lambda x: x)

#tfidf\_vector = TfidfVectorizer(lowercase=True, stop\_words='english')

# Fit the training data

X\_train = count\_vector.fit\_transform(X\_train).toarray()

# Transform testing data

X\_test = count\_vector.transform(X\_test).toarray()

#import sklearn.preprocessing as pr

# Normalize BoW features in training and test set

#X\_train = pr.normalize(X\_train, axis=1)

#X\_test = pr.normalize(X\_test, axis=1)

# print first 200 words/tokens

print(count\_vector.get\_feature\_names()[0:200])

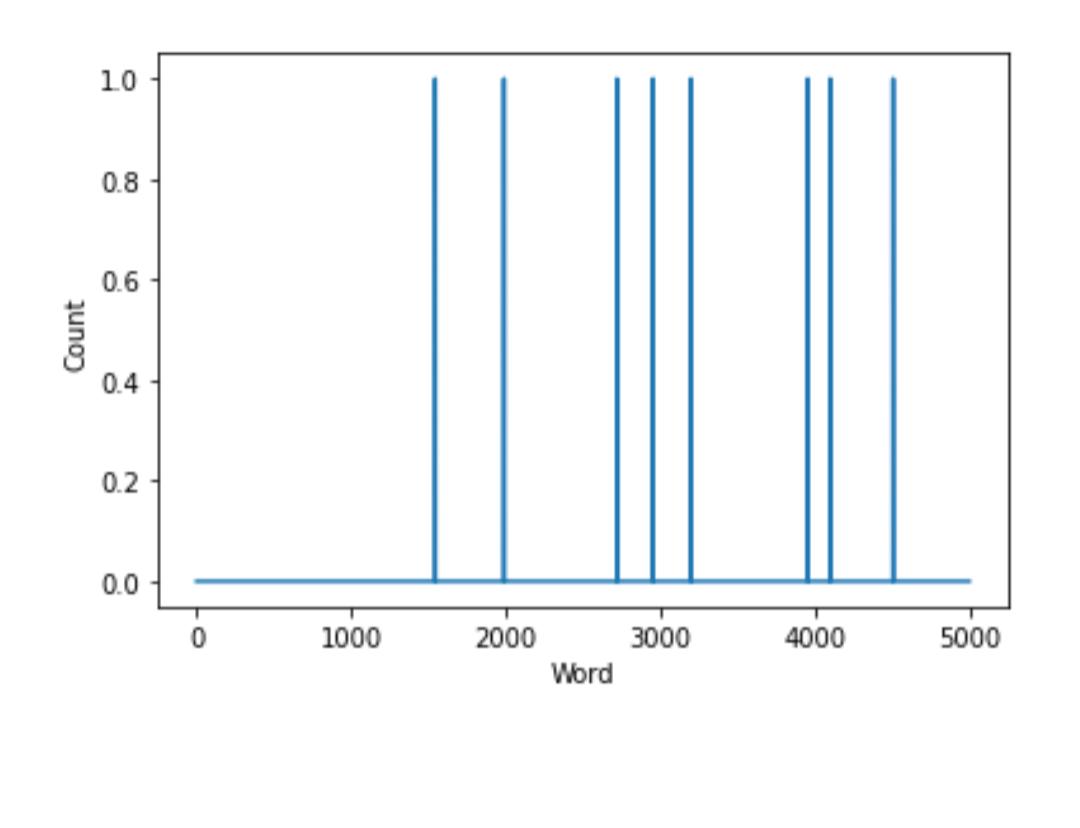
Plot the BoW feature vector

plt.plot(X\_train[2,:])

plt.xlabel('Word')

plt.ylabel('Count')

plt.show()



### **Model Confusion Matrix**

from sklearn.metrics import confusion\_matrix

def plot\_confusion\_matrix(model, X\_test, y\_test):

'''Function to plot confusion matrix for the passed model and the data'''

sentiment\_classes = ['Negative', 'Neutral', 'Positive']

# use model to do the prediction

y\_pred = model.predict(X\_test)

# compute confusion matrix

cm = confusion\_matrix(np.argmax(np.array(y\_test),axis=1), np.argmax(y\_pred, axis=1))

# plot confusion matrix

plt.figure(figsize=(8,6))

sns.heatmap(cm, cmap=plt.cm.Blues, annot=True, fmt='d',

xticklabels=sentiment\_classes,

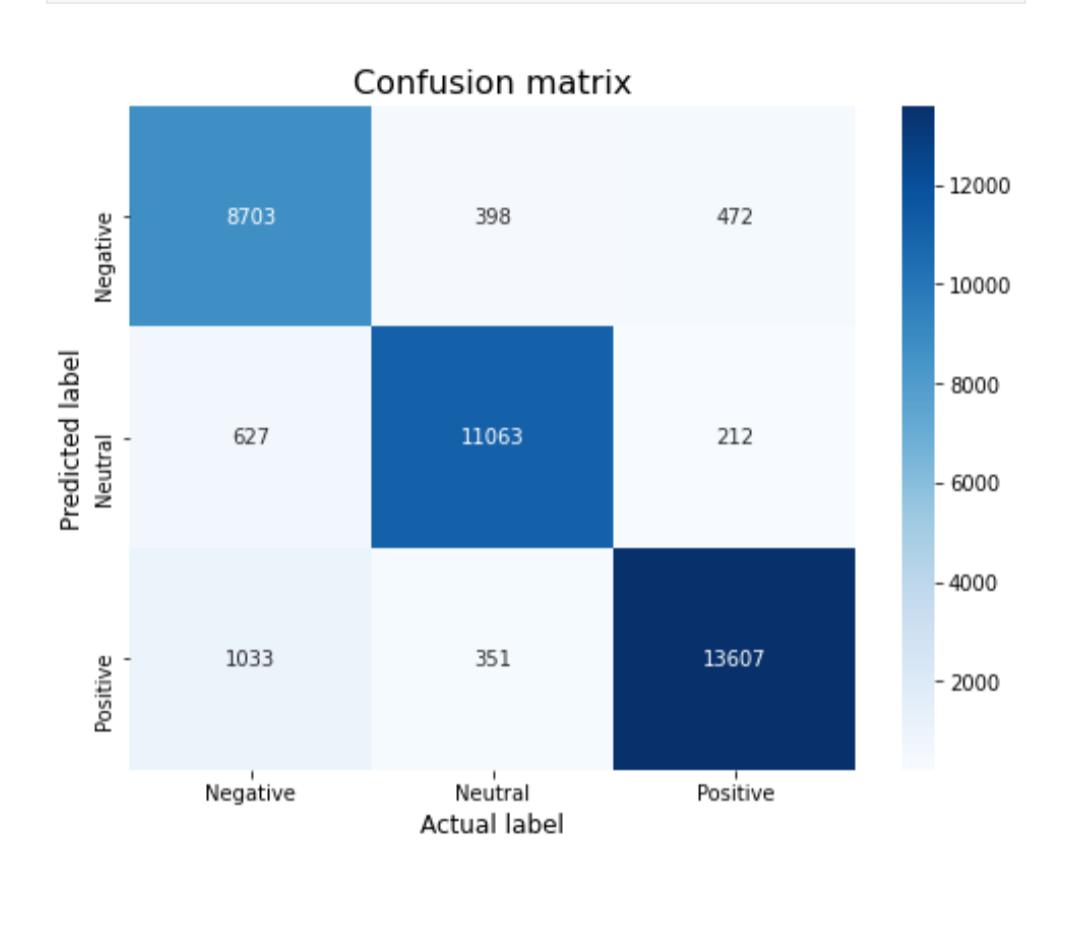
yticklabels=sentiment\_classes)

plt.title('Confusion matrix', fontsize=16)

plt.xlabel('Actual label', fontsize=12)

plt.ylabel('Predicted label', fontsize=12)

plot\_confusion\_matrix(model, X\_test, y\_test)



**Final Model Evaluation**

**List all the final models we obtain from grid search above.**

In [94]:

models = [logreg\_model, KNN\_model, SVC\_model, BNB\_model, CNB\_model, DTC\_model,

RFC\_model, Bagging\_model, Adaboost\_model, GB\_model]

In [95]:

from sklearn.metrics import accuracy\_score,f1\_score, ConfusionMatrixDisplay,classification\_report

In [96]:

accuracy\_scores = []

f1\_scores = []

In [97]:

for model **in** models:

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

acc = accuracy\_score(y\_test,y\_pred)

accuracy\_scores.append(acc)

f1 = f1\_score(y\_test,y\_pred,average='weighted')

f1\_scores.append(f1)

print(model)

print()

print(classification\_report(y\_test,y\_pred,labels=model.classes\_))

print()

ConfusionMatrixDisplay.from\_predictions(y\_test,y\_pred,labels=model.classes\_)

print()

#### **Check the Preformance**

from tensorflow.keras.models import load\_model

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

best\_model = load\_model('best.h5')

y\_pred = best\_model.predict(X\_test)

y\_pred = np.argmax(y\_pred, axis=1)

y\_true = np.argmax(y\_test, axis=1)

accuracy = accuracy\_score(y\_true, y\_pred)

precision = precision\_score(y\_true, y\_pred, average='weighted')

recall = recall\_score(y\_true, y\_pred, average='weighted')

f1 = f1\_score(y\_true, y\_pred, average='weighted')

print('Accuracy:', accuracy)

print('Precision:', precision)

print('Recall:', recall)

print('F1 score:', f1)

92/92 [==============================] - 1s 12ms/step

Accuracy: 0.7646857923497268

Precision: 0.7557311987294797

Recall: 0.7646857923497268

F1 score: 0.7584505674332324

After performing cross validation and hyper parameter tuning via grid search, also evaluating the final 10 models to unseen dataset, here are some conclusion.

The best model is SVC (C=1, gamma=1, kernel=’linear’) with 76.8% accuracy and 77.5% f1 score.

Decision Tree improves significantly after used on Bagging Classifier.

Neutral is the hardest class label to predict accurately.

By following these additional steps, we'll not only build a sentiment analysis system but also ensure that it evolves over time to adapt to changing customer sentiments and provide valuable insights to guide business decisions. This iterative approach is crucial for maintaining the system's effectiveness and relevance. By employing NLP techniques and generating insights from the sentiment analysis, we’ll be able to not only understand customer sentiment but also derive actionable recommendations that can drive business decisions, improve products, and enhance the company's competitive edge in the market. This process can be iterative, with continuous updates and improvements based on changing customer sentiments and feedback.