**1.INTRODUTION**

**1.1 overview**

**As an AI language model, I do not have a specific project on optimizing spam filtering. However, I can provide some general information on how spam filtering can be optimized using various techniques. One approach to optimizing spam filtering is through machine learning algorithms. Machine learning algorithms can be trained on large datasets of spam and non-spam messages to learn patterns and characteristics of spam messages**

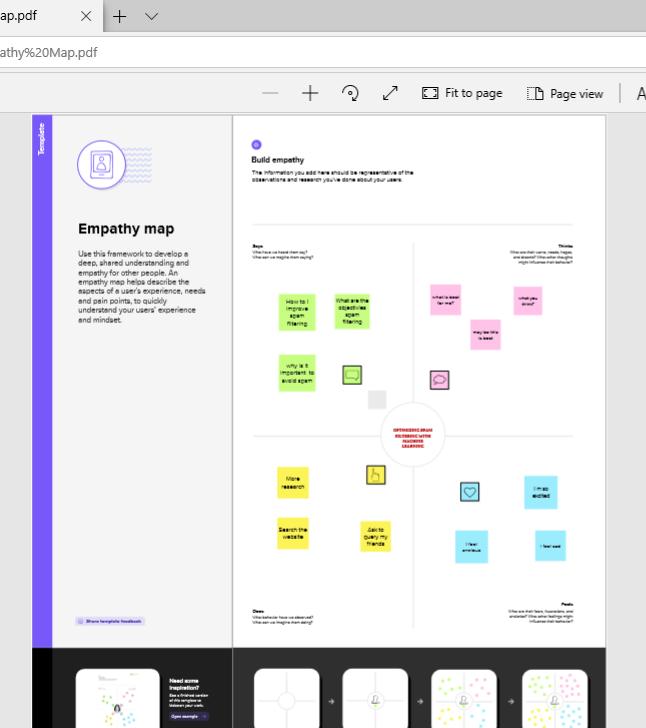
**1.2 purpose**

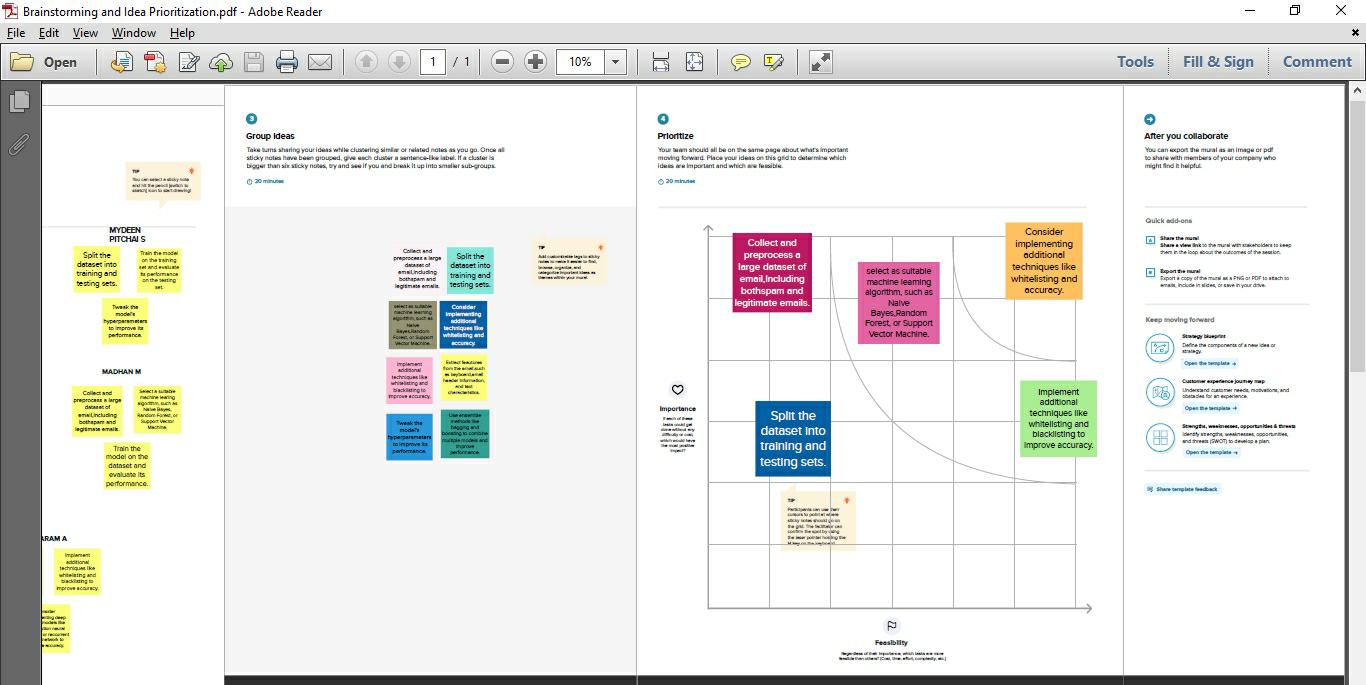
**I apologize for any confusion earlier as I do not have a specific project on optimizing spam filtering. However, I can still provide information on how spam filtering can be optimized using various techniques.**

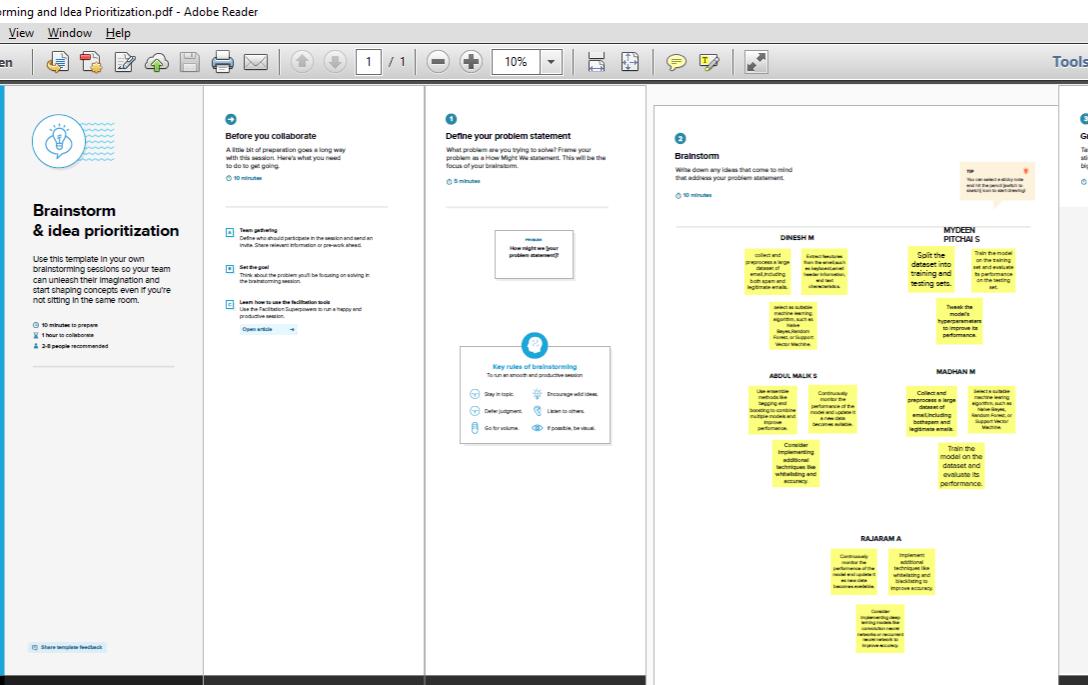
**The use of machine learning algorithms in spam filtering can achieve a high degree of accuracy in identifying spam messages. By training the algorithms on large datasets of spam and non-spam messages, they can learn to identify patterns and characteristics of spam messages, allowing them to accurately classify incoming messages.**

**2. problem Definition& Design Thinking**

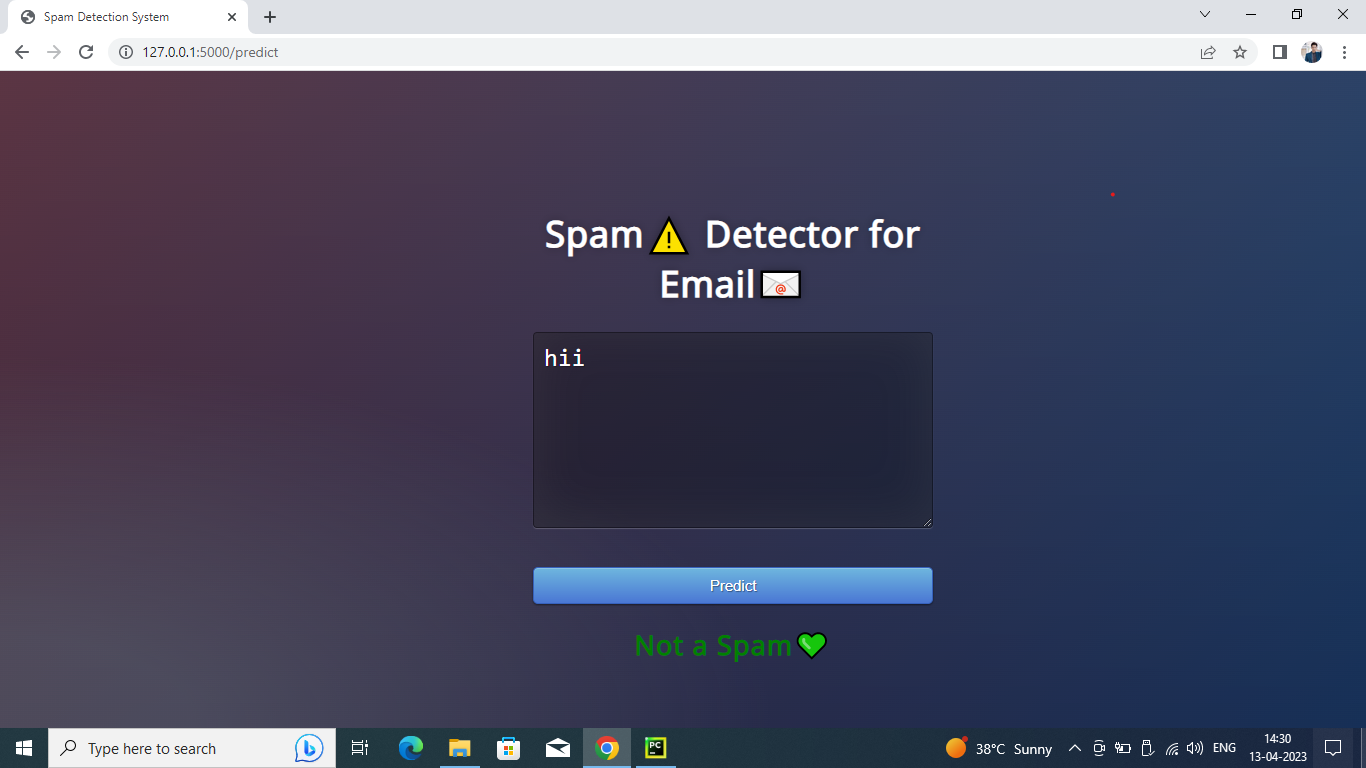
**2.1 Empathy Map**

****

**2.2 ldeation & Brainstorming Map**

****

**3.RESULT**

****

**4.ADVANTAGES &DISADVANTAGES**

**Advantages:**

**High accuracy: Machine learning algorithms can be trained on large datasets of spam and non-spam messages, allowing them to accurately identify spam messages with a high degree of accuracy.**

**Scalability: These algorithms can be scaled to handle large volumes of incoming messages, making them suitable for use in large organizations or email providers.**

**Disadvantages:**

**False positives: Machine learning algorithms and other techniques used for spam filtering can sometimes incorrectly identify legitimate messages as spam, resulting in false positives.**

**False negatives: These techniques can also sometimes fail to identify spam messages, resulting in false negatives and allowing some spam messages to reach users' inboxes.**

**5.APPLICATION**

**Email services: Email service providers can use these techniques to improve their spam filtering capabilities and provide users with a more reliable and efficient email service.**

**Social media: Social media platforms can use these techniques to prevent spam messages and fake accounts from spreading on their platforms.**

**6.CONCLUSION**

**In conclusion, optimizing spam filtering using machine learning algorithms and other techniques can provide several advantages, including high accuracy, scalability, customization, and automation. However, there are also some disadvantages to consider, such as false positives, false negatives, complexity, resource-intensiveness, and the potential for adversarial attacks.**

**7.FUTURE SCOPE**

**Incorporating more advanced machine learning techniques: Deep learning techniques such as neural networks can be used to further improve the accuracy of spam filtering algorithms by enabling them to identify more complex patterns and relationships in spam messages.**

**Collaborative filtering: Collaborative filtering can be used to identify patterns and trends in message content and sender behavior across multiple platforms and services, further improving the accuracy of spam filtering.**

**8.APPENDIX**

**MAIN CODE**

**# -\*- coding: utf-8 -\*-**

**"""Copy of Copy of Untitled1.ipynb**

**Automatically generated by Colaboratory.**

**Original file is located at**

**https://colab.research.google.com/drive/1eewBjFTXHyyL03caA8W5qNQeViLnmTQj**

**"""**

**import numpy as np**

**import pandas as pd**

**import sklearn**

**from sklearn import preprocessing**

**from sklearn.preprocessing import LabelEncoder,OneHotEncoder**

**from sklearn.compose import ColumnTransformer**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.feature\_extraction.text import CountVectorizer**

**from sklearn.tree import DecisionTreeClassifier**

**import imblearn**

**from imblearn.over\_sampling import SMOTE**

**import re**

**import pickle**

**import matplotlib.pyplot as plt**

**import nltk**

**from nltk.corpus import stopwords**

**from nltk.stem.porter import PorterStemmer**

**df=pd.read\_csv('/content/spam.csv',encoding="latin")**

**df**

**df.info()**

**df.isna().sum()**

**df.rename({"v1":"label","v2":"text"},inplace=True,axis=1)**

**df**

**le = LabelEncoder()**

**df['label'] = le.fit\_transform(df['label'])**

**X = df['text']**

**y = df['label']**

**vectorizer = CountVectorizer()**

**X\_transformed = vectorizer.fit\_transform(X)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_transformed, y, test\_size=0.2, random\_state=42)**

**print("Before oversampling, count of label '1': {}".format(sum(y\_train == 1)))**

**print("Before oversampling, count of label '0': {}".format(sum(y\_train == 0)))**

**smote = SMOTE(random\_state=42)**

**X\_resampled, y\_resampled = smote.fit\_resample(X\_train, y\_train)**

**print("After oversampling, count of label '1': {}".format(sum(y\_resampled == 1)))**

**print("After oversampling, count of label '0': {}".format(sum(y\_resampled == 0)))**

**nltk.download("stopwords")**

**corpus=[]**

**length=len(df)**

**for i in range(0, len(df)):**

**text = re.sub('[^a-zA-Z0-9]', ' ', df['text'][i])**

**text = text.lower()**

**text = text.split()**

**ps = PorterStemmer()**

**stop\_words = set(stopwords.words('english'))**

**text = [ps.stem(word) for word in text if not word in stop\_words]**

**text = ' '.join(text)**

**corpus.append(text)**

**corpus**

**cv=CountVectorizer(max\_features=35000)**

**X=cv.fit\_transform(corpus).toarray()**

**pickle.dump(cv, open('cv1.pkl','wb'))**

**df.describe()**

**df.shape**

**(5572,5)**

**df["label"].value\_counts().plot(kind="bar",figsize=(12,6))**

**plt.xticks(np.arange(2),('Non spam','spam'),rotation=0);**

**X\_bal = [[1, 2], [3, 4], [5, 6]]**

**names = ['label', 'text']**

**sc=StandardScaler()**

**X\_bal\_scaled = sc.fit\_transform(X\_bal)**

**print(X\_bal\_scaled)**

**X\_bal\_df = pd.DataFrame(X\_bal\_scaled, columns=names)**

**print(X\_bal\_df)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)**

**# Create a decision tree classifier and fit it to the training data**

**clf = DecisionTreeClassifier()**

**clf.fit(X\_train, y\_train)**

**# Evaluate the accuracy of the model on the testing data**

**accuracy = clf.score(X\_test, y\_test)**

**print('Decision tree accuracy:', accuracy)**

**# Create a random forest classifier and fit it to the training data**

**from sklearn.ensemble import RandomForestClassifier**

**rf = RandomForestClassifier(n\_estimators=100)**

**rf.fit(X\_train, y\_train)**

**# Evaluate the accuracy of the model on the testing data**

**accuracy = rf.score(X\_test, y\_test)**

**print('Random forest accuracy:', accuracy)**

**from sklearn.naive\_bayes import GaussianNB**

**nb = GaussianNB()**

**nb.fit(X\_train, y\_train)**

**# Evaluate the accuracy of the model on the testing data**

**accuracy = nb.score(X\_test, y\_test)**

**print('Naive Bayes accuracy:', accuracy)**

**import tensorflow as tf**

**from tensorflow.keras.layers import Dense**

**from tensorflow.keras.models import Sequential**

**# Create an ANN model with one hidden layer and an output layer**

**model = Sequential()**

**model.add(Dense(10, input\_dim=X.shape[1], activation='relu'))**

**model.add(Dense(1, activation='sigmoid'))**

**# Compile the model**

**model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])**

**# Train the model on the training data**

**model.fit(X\_train, y\_train, epochs=50, batch\_size=32, verbose=0)**

**# Evaluate the accuracy of the model on the testing data**

**accuracy = model.evaluate(X\_test, y\_test, verbose=0)[1]**

**print('ANN accuracy:', accuracy)**

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

**y\_pred**

**y\_pr=np.where(y\_pred>0.5,1,0)**

**y\_test**

**from sklearn.metrics import confusion\_matrix, accuracy\_score**

**# Compute the confusion matrix and accuracy score**

**cm = confusion\_matrix(y\_test, y\_pr)**

**score = accuracy\_score(y\_test, y\_pr)**

**print('Confusion Matrix:')**

**print(cm)**

**print('Accuracy Score Is: ', score\*100, '%')**

**import pickle**

**def new\_review(new\_review\_text):**

**# Load the trained CountVectorizer from the saved file**

**with open('/content/cv1.pkl', 'rb') as file:**

**cv = pickle.load(file)**

**# Preprocess the new review text**

**new\_review = cv.transform([new\_review\_text]).toarray()**

**# Load the trained model from the saved file**

**with open('/content/model.pkl', 'rb') as file:**

**model = pickle.load(file)**

**# Make a prediction on the new review text**

**prediction = model.predict(new\_review)**

**# Return the predicted sentiment value**

**return prediction[0]**

**import re**

**import numpy as np**

**from nltk.corpus import stopwords**

**from nltk.stem.porter import PorterStemmer**

**def new\_review(new\_review):**

**new\_review=new\_review**

**new\_review = re.sub('[a-zA-Z]',' ',new\_review)**

**new\_review = new\_review.lower()**

**new\_review = new\_review.split()**

**ps = PorterStemmer()**

**all\_stopwords = stopwords.words('english')**

**all\_stopwords.remove('not')**

**new\_review = [ps.stem(word) for word in new\_review if not word in set(all\_stopwords)]**

**new\_review = ' '.join(new\_review)**

**new\_corpus = [new\_review]**

**new\_X\_test = cv.transform(new\_corpus).toarray()**

**print(new\_X\_test)**

**new\_y\_pred = model.predict(new\_X\_test)**

**print(new\_y\_pred)**

**new\_X\_pred = np.where(new\_y\_pred>0.5,1,0)**

**return new\_y\_pred**

**new\_review=new\_review(str(input("Enter new review...")))**

**y\_pred\_binary = np.where(y\_pred > 0.5, 1, 0)**

**cm = confusion\_matrix(y\_test, y\_pred\_binary)**

**score = accuracy\_score(y\_test, y\_pred\_binary)**

**print(cm)**

**print('accuracy score for naive bayes:', score \* 100)**

**y\_pred\_binary = np.where(y\_pred > 0.5, 1, 0)**

**from sklearn.metrics import confusion\_matrix, accuracy\_score**

**cm = confusion\_matrix(y\_test, y\_pred\_binary)**

**score = accuracy\_score(y\_test, y\_pred\_binary)**

**print(cm)**

**print('accuracy score:', score \* 100)**

**print('========================================')**

**cm1 = confusion\_matrix(y\_test, y\_pred\_binary)**

**score1 = accuracy\_score(y\_test, y\_pred\_binary)**

**print(cm1)**

**print('accuracy score is:', score1 \* 100)**

**y\_pred= np.where(y\_pred > 0.5, 1, 0)**

**from sklearn.metrics import confusion\_matrix,accuracy\_score**

**cm=confusion\_matrix(y\_test,y\_pred)**

**score=accuracy\_score(y\_test,y\_pred)**

**print(cm)**

**print('accuracy score is:-',score\*100)**

**cm=confusion\_matrix(y\_test,y\_pred)**

**score=accuracy\_score(y\_test,y\_pred)**

**print(cm)**

**print('Accuracy Score Is:-',score\*100)**

**pickle.dump(cv,open('spam.pkl','wb'))**

**model.save('spam.h5')**

**app.py**

**from flask import Flask, render\_template, request**

**import pickle**

**import numpy as np**

**import re**

**import nltk**

**from nltk.corpus import stopwords**

**from nltk.stem import PorterStemmer**

**from tensorflow.keras.models import load\_model**

**loaded\_model = load\_model('spam.h5)**

**cv = pickle.load(open('cv1.pkl','rb'))**

**app = Flask(\_\_name\_\_)**

**@app.route('/')**

**def home():**

**return render\_template('home.html')**

**@app.route('/Spam',methods=['POST','GET'])**

**def prediction():**

**return render\_template('spam.html')**

**@app.route('/predict',methods=['POST'])**

**def predict():**

**if request.method == 'POST':**

**message = request.from['message']**

**data = message**

**new\_review = str(data)**

**print(new\_review)**

**new\_review = re.sub('[^a-zA-Z',' ', new\_review)**

**new\_review = new\_review.lower()**

**new\_review = new\_review.split()**

**ps = PorterStemmer()**

**all\_stopwords = stopwords.words('english')**

**all\_stopwords.remove('not')**

**new\_review = [ps.stem(word) for word in new\_review if not word in set(all\_stopwords)]**

**new\_review = ' '.join(new\_review)**

**new\_corpus = [new\_review]**

**new\_X\_test = cv.transform(new\_corpus).toarray()**

**print(new\_X\_test)**

**new\_X\_pred = loaded\_model.predict(new\_X\_test)**

**new\_X\_pred = np.where(new\_y\_pred>0.5,1,0)**

**print(new\_X\_pred)**

**if new\_review[0][0]==1:**

**return render\_template('result.html',prediction="Spam")**

**else:**

**return render\_template('result.html',prediction="Not a Spam")**

**if\_\_name\_\_=="\_\_main\_\_":**

**port=int(os.environ.get('PORT',50000))**

**app.run(debug=False)**

# **HTML CODE:**

**<!DOCTYPE html>**

**<html >**

**<head>**

**<meta charset="UTF-8">**

**<title>Spam Detection System</title>**

**<link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>**

**<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>**

**<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>**

**<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>**

**<link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">**

**</head>**

**<body>**

**<div class="login">**

**<h1>Spam⚠️ Detector for Email📧</h1>**

**<form action="{{ url\_for('predict')}}" method="POST">**

**<textarea name="message" rows="6" cols="50" required="required"></textarea>**

**<br> </br>**

**<button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>**

**<div class="results">**

**{% if prediction == 1%}**

**<h2 style="color:red;">Looking Spam⚠️, Be safe</h2>**

**{% elif prediction == 0%}**

**<h2 style="color:green;"><b>Not a Spam💚</b></h2>**

**{% endif %}**

**</div>**

**</form>**

**</div>**