Thiruvalluvar Government Arts College

Rasipuram – 637408

Optimizing Spam Filtering with Machine Language

Team Leader : Sakthivel S(20UCS2309)

Team Member : Ananth S(20UCS2297)

Team Member : Soundararajan A(20UCS2313)

Team Member : Premkumar V(20UCS2307)

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

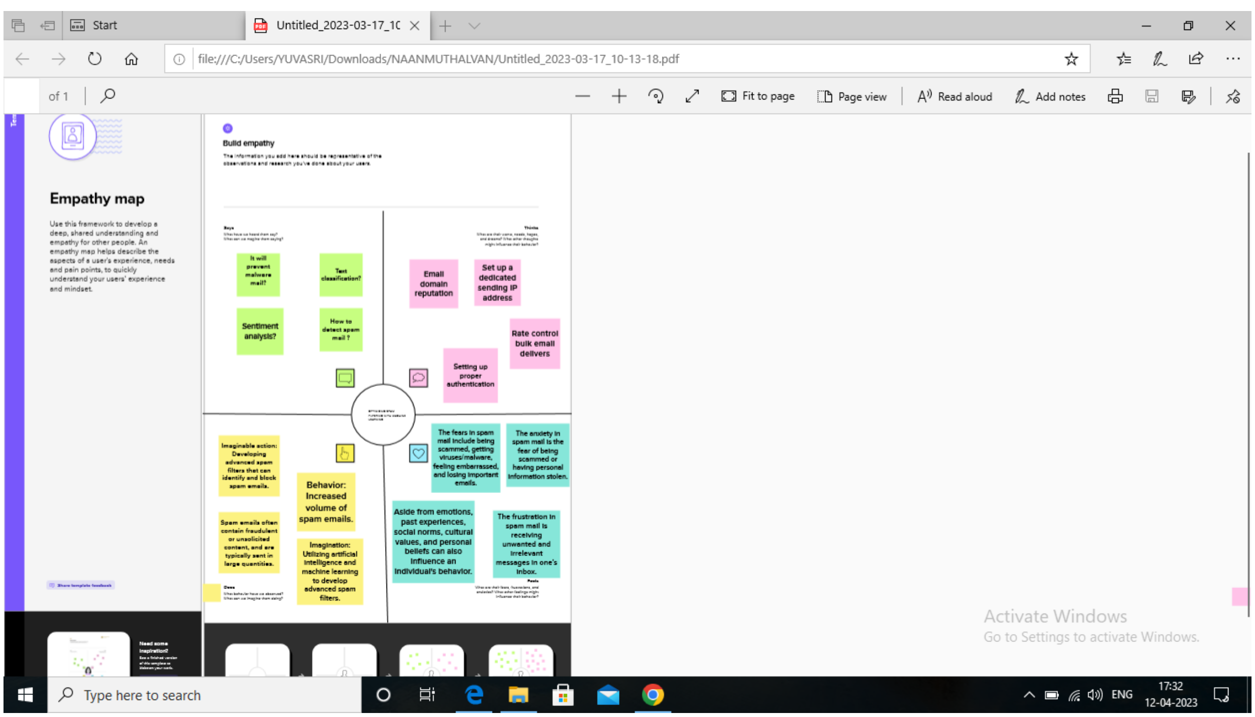
1.2 purpose

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.

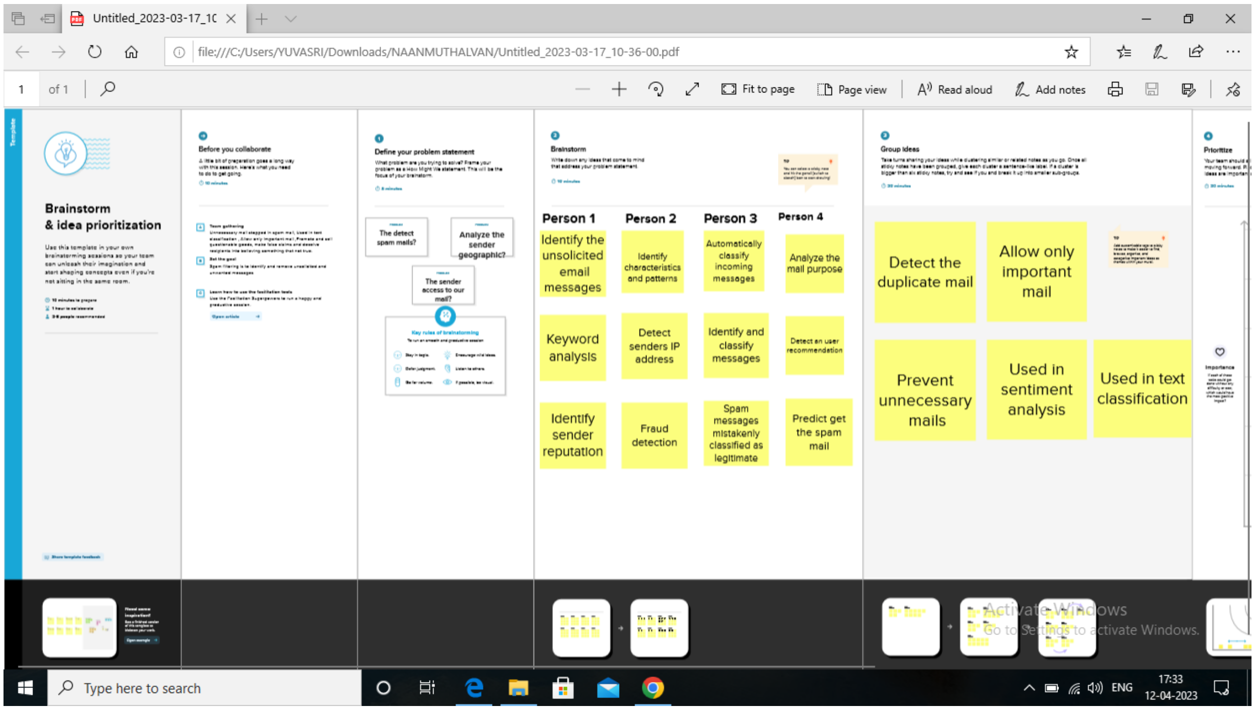
The use of machine learning algorithms in spam filtering can achieve a high degree of accuracy in identifying spam messages. By training the algorithms on large datasets of spam and non-spam messages, they can learn to identify patterns and characteristics of spam messages, allowing them to accurately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

2. problem Definition& Design Thinking

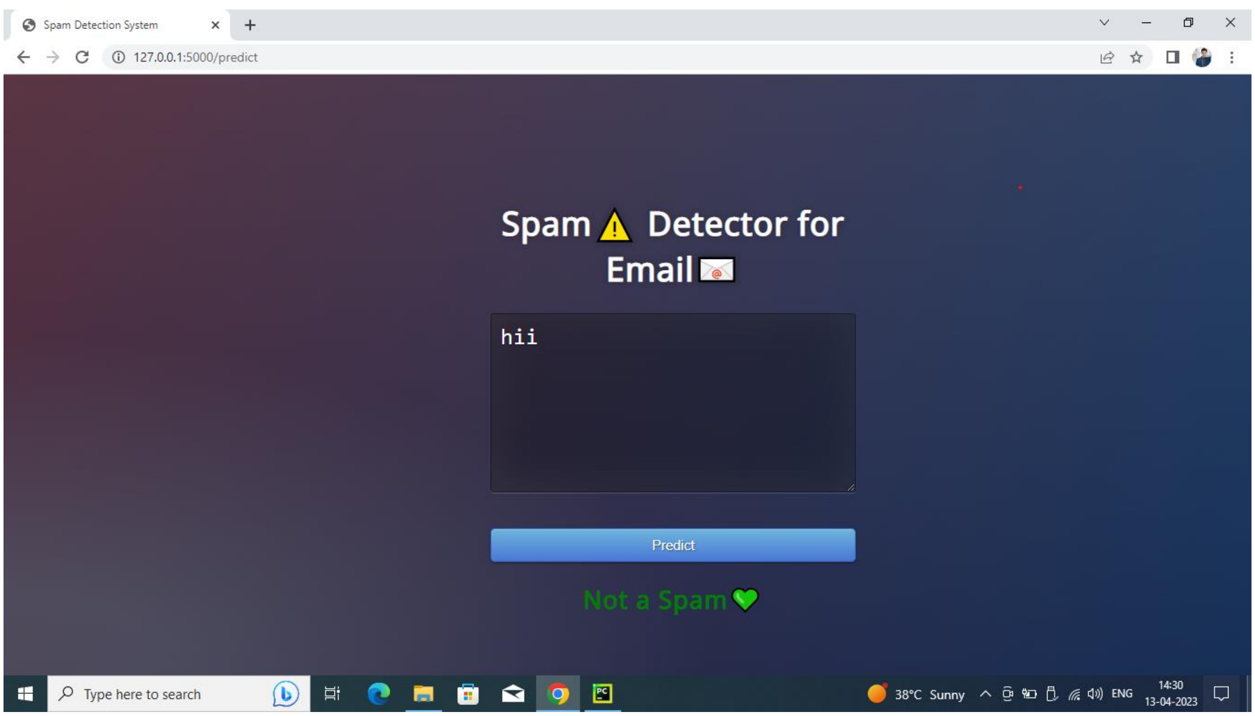
2.1 Empathy Map



2.2 Ideation & 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>
```