**22CB903  
MINI-PROJECT 2**

OBJECTIVE:

To build a text classification model that classifies text messages as either "ham" (non-spam) or "spam", using two different feature extraction methods: TF-IDF (Term Frequency-Inverse Document Frequency) and Word Embeddings (Word2Vec).

ALGORITHM:

Data Preprocessing:

1. Load the dataset containing text messages and their respective labels (ham/spam).
2. Text Cleaning:

* Convert the text to lowercase.
* Remove punctuation and special characters.
* Remove digits if present.
* Tokenize the text by splitting it into words.

1. Label Encoding: Convert the "ham" and "spam" labels into numerical values (e.g., ham → 0, spam → 1).

Feature Engineering:

1. TF-IDF Approach:

* Use the TfidfVectorizer from scikit-learn to convert the cleaned text into numerical features. This assigns a weight to each word based on its frequency in the document and its inverse frequency in the entire dataset.
* Transform the text data into TF-IDF features for both training and test sets.

1. Word Embeddings Approach (Word2Vec):

* Train a Word2Vec model on the tokenized text to learn 100-dimensional word embeddings.
* For each message, compute the average word vector by averaging the vectors of all the words in the message.
* Use these average vectors as features for both training and test sets.

Model Training:

1. Train a Logistic Regression model using the extracted features from both the TF-IDF and Word Embedding (Word2Vec) approaches.

Model Evaluation:

1. Predict the labels for the test set using both models (TF-IDF and Word2Vec).
2. Evaluate the performance using the following metrics:

* Accuracy: The proportion of correctly classified messages.
* Precision: The proportion of predicted positive cases (spam) that are truly positive.
* Recall: The proportion of actual positive cases (spam) that were correctly identified.
* F1-score: The harmonic mean of precision and recall.

1. Compare the performance of both models based on these metrics.

CODE:

import pandas as pd

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

from sklearn.preprocessing import LabelEncoder

import re

import string

from gensim.models import Word2Vec

import numpy as np

# Data preprocessing function: clean and tokenize the text

def preprocess\_text(text):

text = text.lower() # Lowercase the text

text = re.sub(f"[{string.punctuation}]", "", text) # Remove punctuation

text = re.sub(r"\d+", "", text) # Remove digits

return text

# Load the dataset

dataset = pd.read\_csv(r"C:\stuff\college stuff\study\cb903\22CB903-Machine-Learning-MiniProjects\02-Text Classification\dataset.csv")

# Preprocess the text messages

dataset['cleaned\_message'] = dataset['Message'].apply(preprocess\_text)

# Encode the labels: ham -> 0, spam -> 1

label\_encoder = LabelEncoder()

dataset['label'] = label\_encoder.fit\_transform(dataset['Category'])

# Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

dataset['cleaned\_message'], dataset['label'], test\_size=0.2, random\_state=42

)

# Step 2: TF-IDF Feature Engineering

tfidf = TfidfVectorizer(max\_features=5000)

X\_train\_tfidf = tfidf.fit\_transform(X\_train)

X\_test\_tfidf = tfidf.transform(X\_test)

# Step 3: Train a Logistic Regression model on TF-IDF features

lr\_model = LogisticRegression(max\_iter=1000)

lr\_model.fit(X\_train\_tfidf, y\_train)

# Make predictions and evaluate the model

y\_pred\_tfidf = lr\_model.predict(X\_test\_tfidf)

report\_tfidf = classification\_report(y\_test, y\_pred\_tfidf, target\_names=['ham', 'spam'])

print("TF-IDF Model Performance:\n", report\_tfidf)

# Step 4: Word Embedding (Word2Vec) Feature Engineering

# Tokenize the cleaned text for Word2Vec input

tokenized\_messages = dataset['cleaned\_message'].apply(lambda x: x.split())

# Train a Word2Vec model

word2vec\_model = Word2Vec(sentences=tokenized\_messages, vector\_size=100, window=5, min\_count=1, workers=4)

# Function to get the average word vector for a sentence

def get\_avg\_word2vec(sentence, model, vector\_size):

words = sentence.split()

avg\_vector = np.zeros(vector\_size)

valid\_words = 0

for word in words:

if word in model.wv:

avg\_vector += model.wv[word]

valid\_words += 1

if valid\_words > 0:

avg\_vector /= valid\_words

return avg\_vector

# Transform the dataset using the average word vectors

X\_train\_w2v = np.array([get\_avg\_word2vec(sentence, word2vec\_model, 100) for sentence in X\_train])

X\_test\_w2v = np.array([get\_avg\_word2vec(sentence, word2vec\_model, 100) for sentence in X\_test])

# Train a Logistic Regression model on Word2Vec features

lr\_w2v\_model = LogisticRegression(max\_iter=1000)

lr\_w2v\_model.fit(X\_train\_w2v, y\_train)

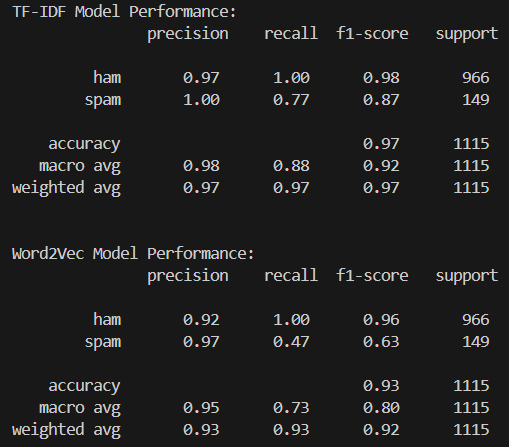
# Make predictions and evaluate the Word2Vec model

y\_pred\_w2v = lr\_w2v\_model.predict(X\_test\_w2v)

report\_w2v = classification\_report(y\_test, y\_pred\_w2v, target\_names=['ham', 'spam'])

print("\nWord2Vec Model Performance:\n", report\_w2v)

OUTPUT:



OBSERVATIONS & COMPARISION:

|  |  |  |
| --- | --- | --- |
| **Metric** | TF-IDF Model (Logistic Regression) | Word2Vec Model (Logistic Regression) |
| Accuracy | 97% | 93% |
| Precision (Ham) | 97% | 92% |
| Precision (Spam) | 100% | 97% |
| Recall (Ham) | 100% | 100% |
| Recall (Spam) | 77% | 47% |
| F1-score (Ham) | 98% | 96% |
| F1-score (Spam) | 87% | 63% |
| Macro Avg (F1) | 92% | 80% |
| Weighted Avg (F1) | 97% | 92% |

1. Accuracy:

* The TF-IDF model achieved higher accuracy (97%) compared to the Word2Vec model (93%).

1. Precision:

* Both models have excellent precision for detecting spam, but TF-IDF has a slight edge in precision for "ham" (non-spam) as well.
* TF-IDF: 97% (ham) and 100% (spam) precision.
* Word2Vec: 92% (ham) and 97% (spam) precision.

1. Recall:

* TF-IDF has significantly higher recall for spam (77%) compared to Word2Vec (47%), meaning the TF-IDF model is much better at identifying spam messages.
* Both models have perfect recall (100%) for detecting "ham" (non-spam).

1. F-1 Score:

* The F1-score for spam detection is much higher in the TF-IDF model (87%) compared to Word2Vec (63%). This indicates that TF-IDF balances precision and recall more effectively for spam classification.
* The F1-score for "ham" is high for both models, with TF-IDF slightly outperforming.

1. Macro and Weighted Averages:

* The macro average (average of precision and recall for each class) for TF-IDF (92%) is higher than for Word2Vec (80%).
* The weighted average of F1-scores also shows a clear advantage for TF-IDF, with 97% vs 92% for Word2Vec.

CONCLUSION:

* TF-IDF is the superior model in this case, especially in terms of accuracy, recall, and F1-score for spam detection. It performs better at balancing precision and recall, especially for the minority "spam" class.
* Word2Vec lags in spam detection due to lower recall, though its precision remains high.

*By*

*S Sharvesh Guru*

*CSBS* | *111722202043*