

Title: A Machine Learning Approach to SMS Spam Detection

Currently, the use of cell phones has surged in the last decade, leading, among other things, to a problem regarding text messages with promotions from unscrupulous marketing agents, and with fraudulent messages that can lead us to have a bad day. To solve this problem, in this project we will focus building a machine learning model that accurately allows us to predict and identify when a received text message is spam or not (ham), with the help of the text body of SMS, while using a [SMS Spam Collection Dataset from kaggle.com](#) with a total of 5,169 unique and legitimate text messages downloaded from the Kaggle website.

Objectives:

- Analyze the provided SMS dataset to understand the characteristics of spam and legitimate (ham) messages.
- Explore the content of messages, examining factors like message length, common words, and phrases.
- Develop and train a spam detection model using a machine learning algorithm.
- Assess the performance of the model through testing and validation.
- Implement the trained model to classify new messages and measure its accuracy in

```
In [ ]: # Importing the Libraries
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

import pickle
```

```
In [ ]: # # Read the model from disk
# Loaded_model = pickle.load(open(filename, 'rb'))
```

```
In [ ]: # Importing the dataset
df = pd.read_csv('spam.csv', encoding='latin1')
df.head()
```

```
Out[ ]:   v1
          v2  Unnamed: 2  Unnamed: 3  Unnamed: 4
0  ham    Go until jurong point, crazy.. Available only ...
1  ham        Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3  ham    U dun say so early hor... U c already then say...
4  ham    Nah I don't think he goes to usf, he lives aro...
```

```
In [ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   v1          5572 non-null    object  
 1   v2          5572 non-null    object  
 2   Unnamed: 2   50 non-null     object  
 3   Unnamed: 3   12 non-null     object  
 4   Unnamed: 4   6 non-null      object  
dtypes: object(5)
memory usage: 217.8+ KB
```

Notice that there are three columns almost empty: `Unnamed: 2`, `Unnamed: 3`, and `Unnamed: 4`. Let's check unique values for these columns and be sure what we have here.

```
In [ ]: df['Unnamed: 2'].unique()
```

```
Out[ ]: array([nan, ' PO Box 5249',
   ' the person is definitely special for u..... But if the person is so special',
   ' HOWU DOIN? FOUNDURSELF A JOBYET SAUSAGE?LOVE JEN XXX\\'''',
   ' wanted to say hi. HI!!!\\" Stop? Send STOP to 62468"',
   'this wont even start..... Datz confidence..', 'GN',
   '.;-):-D"',
   'just been in bedbut mite go 2 thepub l8tr if uwana mt up?loads a luv Jenxxx.\\''',
   ' bt not his girlfrnd... G o o d n i g h t . . . @''',
   ' I\\'ll come up"',
   ' don\\t miss ur best life for anything... Gud nyt...',',
   ' just as a shop has to give a guarantee on what they sell. B. G.'',
   ' But at d end my love compromised me for everything:-(\\".. Gud mornin:-)''',
   ' the toughest is acting Happy with all unspoken pain inside..\\''',
   ' smoke hella weed\\''', '\\\" not \\\"what i need to do.\\''',
   'JUST GOT PAYED2DAY & I HAVBEEN GIVEN A£50 PAY RISE 4MY WORK & HAVEBEEN MADE PRESCHOOLCO
-ORDINATOR 2I AM FEELINGOOD LUV\\''',
   ' justthought iåõd sayhey! how u doin?nearly the endof me wk offdam nevamind!We will have
2Hook up sn if uwant m8? loveJen x.\\''',
   'JUST REALLYNEED 2DOCD.PLEASE DONTPLEASE DONTIGNORE MYCALLS',
   'u hav2hear it!c u sn xxxx\\''', " I don't mind",
   ' Dont Come Near My Body...!! Bcoz My Hands May Not Come 2 Wipe Ur Tears Off That Time..!G
ud ni8"',
   "Well there's still a bit left if you guys want to tonight",
   ' but dont try to prove\\" ..... Gud mrng...',',
   ' SHE SHUDVETOLD U. DID URGRAN KNOW?NEWAY',
   ' but watever u shared should be true\\"....',
   ' like you are the KING\\"...! OR \\\"Walk like you Dont care',
   ' HAD A COOL NYTHO', ' PO Box 1146 MK45 2WT (2/3)"',
   ' \\\"It is d wonderful fruit that a tree gives when it is being hurt by a stone.. Good ni
ght.....',
   ' we made you hold all the weed\\''',
   ' but dont try to prove it..\\".Gud noon....',
   ' its a miracle to Love a person who can\\t Love anyone except U...\\" Gud nyt...',',
   ' Gud night....',
   ' that\\'s the tiny street where the parking lot is"',
   'PROBPOP IN & CU SATTHEN HUNNY 4BREKKIE! LOVE JEN XXX. PSXTRA LRG PORTIONS 4 ME PLEASE
\\''',
   ' hopeSo hunny. i amnow feelin ill & ithink i may have tonsolitusaswell! damn iam layin i
n bedreal bored. lotsof luv me xxxx\\''',
   ' GOD said',
   ' always give response 2 who cares 4 U\\"... Gud night..swt dreams..take care"',
   ' HOPE UR OK... WILL GIVE U A BUZ WEDLUNCH. GO OUTSOMEWHERE 4 ADRINK IN TOWN..CUD GO 2WAT
ERSHD 4 A BIT? PPL FROMWRK WILL BTHERE. LOVE PETEXXX.\\''',
   ' b\\'coz nobody will fight for u. Only u & u have to fight for ur self & win the
battle. -VIVEKANAND- G 9t.. SD..',
   'DEVIOUSBITCH.ANYWAY',
   ' ENJOYIN INDIANS AT THE MO..yeP. SaLL g0oD HehE ;> hows bout u shexy? Pete Xx\\''''],
dtype=object)
```

```
In [ ]: df['Unnamed: 3'].unique()
```

```
Out[ ]: array([nan, ' MK17 92H. 450Ppw 16"', ' why to miss them', 'GE',
   'U NO THECD ISV.IMPORTANT TOME 4 2MORO\\''',
   'i wil tolerat.bcs ur my someone..... But',
   ' ILLSPEAK 2 U2MORO WEN IM NOT ASLEEP...\\''',
   'whoever is the KING\\"... Gud nyt"', ' TX 4 FONIN HON',
   ' \\\"OH No! COMPETITION\\. Who knew', 'IåõL CALL U\\''''],
dtype=object)
```

```
In [ ]: df['Unnamed: 4'].unique()
```

```
Out[ ]: array([nan, ' just Keep-in-touch\\\" gdeve.."', 'GNT:-)"',  
             ' Never comfort me with a lie\\\" gud ni8 and sweet dreams"',  
             ' CALL 2MVEN IM BK FRMCLOUD 9! J X\\""',  
             ' one day these two will become FREINDS FOREVER!"'], dtype=object)
```

As we saw here, these are reply messages for the initial one. So there is no relevant information here and we can drop these three columns.

1. Data Preprocessing

```
In [ ]: # removing unnecessary columns  
df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
```

```
In [ ]: df.head()
```

```
Out[ ]:
```

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
In [ ]: # changing v1 column to spam and text column to text  
df = df.rename(columns={"v1": "spam", "v2": "text"})  
df.head()
```

```
Out[ ]:
```

	spam	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
In [ ]: # Count of duplicated rows  
df.duplicated().sum()
```

```
Out[ ]: 403
```

There are 403 duplicated rows to drop.

```
In [ ]: #removing duplicates  
df = df.drop_duplicates()  
df.shape
```

```
Out[ ]: (5169, 2)
```

At this point I would like to understand better the features and the dataset in overall by plotting a few Graphs.

I woul like to start by checking whether the dataset is balanced or not in terms of the outcome variable `spam`.

```
In [ ]: # Creating a Pie chart for spam and ham
```

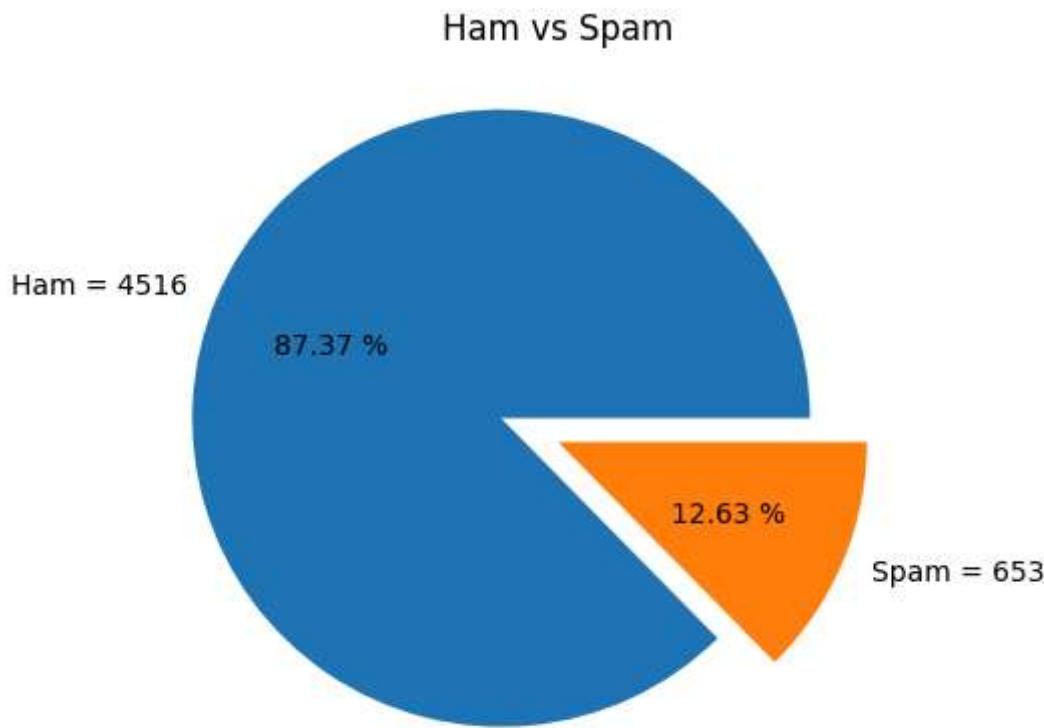
```
number_of_spam = df[df['spam'] == 'spam'].shape[0]
number_of_ham = df[df['spam'] == 'ham'].shape[0]

plt.figure(figsize=(6,5))

mail_categories = [number_of_ham, number_of_spam]
labels = [f"Ham = {number_of_ham}", f"Spam = {number_of_spam}"]
explode = [.2, 0]

plt.pie(mail_categories, labels=labels, explode=explode, autopct=".2f %%")
plt.title("Ham vs Spam")

plt.show()
```



```
In [ ]: # Removing punctuation
```

```
import string
string.punctuation

def remove_punctuation(text):
    return text.translate(str.maketrans('', '', string.punctuation))

df['text'] = df['text'].apply(remove_punctuation)
df.head()
```

Out[]:

	spam	text
0	ham	Go until jurong point crazy Available only in ...
1	ham	Ok lar Joking wif u oni
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor U c already then say
4	ham	Nah I dont think he goes to usf he lives aroun...

2. Feature Engineering

Following data cleaning, we proceed to feature engineering. The initial step entails encoding the 'spam' feature using scikit-learn's LabelEncoder class.

In []:

```
# # Using sci-kit Learn to Label encode the spam column

# from sklearn.preprocessing import LabelEncoder
# le = LabelEncoder()
# df['spam'] = le.fit_transform(df['spam'])
# df.head()

def manual_label_encoder(labels):

    encoding_map = {"spam": 0, "ham": 1} # Adjust class Labels and their corresponding values
    encoded_labels = []
    for label in labels:
        encoded_labels.append(encoding_map[label])
    return encoded_labels

df['spam'] = manual_label_encoder(df['spam'])

df.head()
```

Out[]:

	spam	text
0	1	Go until jurong point crazy Available only in ...
1	1	Ok lar Joking wif u oni
2	0	Free entry in 2 a wkly comp to win FA Cup fina...
3	1	U dun say so early hor U c already then say
4	1	Nah I dont think he goes to usf he lives aroun...

To enrich the model's feature set, we can introduce two additional numerical features: word count and message length

In []:

```
# Column to represent the Length of the text
df['length'] = df['text'].apply(len)
df.head()
```

Out[]:

	spam	text	length
0	1	Go until jurong point crazy Available only in ...	102
1	1	Ok lar Joking wif u oni	23
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	149
3	1	U dun say so early hor U c already then say	43
4	1	Nah I dont think he goes to usf he lives aroun...	59

In []:

```
# Column with the number of words

def count_words(text):
    return len(text.split())

df['words'] = df['text'].apply(count_words)
df.head()
```

Out[]:

	spam	text	length	words
0	1	Go until jurong point crazy Available only in ...	102	20
1	1	Ok lar Joking wif u oni	23	6
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	149	28
3	1	U dun say so early hor U c already then say	43	11
4	1	Nah I dont think he goes to usf he lives aroun...	59	13

Word tokenization:

Word tokenization, in the context of Natural Language Processing (NLP), is the process of splitting a piece of text into smaller units called tokens. The tokenizer splits the text based on predefined rules or delimiters. The most common delimiter is a whitespace character (space, tab, newline, etc.), resulting in individual words as tokens.

In []:

```
import nltk
from nltk.tokenize import word_tokenize

# Download necessary NLTK resources (do this only once)
nltk.download('punkt')

def tokenize_text_nltk(text):
    # Lowercase the text
    text = text.lower()
    # Tokenize using NLTK word_tokenize
    tokens = word_tokenize(text)
    return tokens

df['tokens_nltk'] = df['text'].apply(tokenize_text_nltk)

df.head()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\carlo\AppData\Roaming\nltk_data...
[nltk_data]     Package punkt is already up-to-date!
```

	spam	text	length	words	tokens_nltk
0	1	Go until jurong point crazy Available only in ...	102	20	[go, until, jurong, point, crazy, available, o...
1	1	Ok lar Joking wif u oni	23	6	[ok, lar, joking, wif, u, oni]
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	149	28	[free, entry, in, 2, a, wkly, comp, to, win, f...
3	1	U dun say so early hor U c already then say	43	11	[u, dun, say, so, early, hor, u, c, already, t...
4	1	Nah I dont think he goes to usf he lives aroun...	59	13	[nah, i, dont, think, he, goes, to, usf, he, l...

Removing Stop Words:

In Natural Language Processing (NLP), removing stop words is a technique where you eliminate frequently occurring, generic words from your text data. These stop words typically don't contribute much to the core meaning of a sentence and can even add noise to your analysis.

```
In [ ]: # removing stopwords

from nltk.corpus import stopwords

# Download necessary NLTK resources
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

def remove_stopwords(tokens):
    return [t for t in tokens if t not in stop_words]

df['tokens_nltk'] = df['tokens_nltk'].apply(remove_stopwords)

df.head()
```

[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\carlo\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!

	spam	text	length	words	tokens_nltk
0	1	Go until jurong point crazy Available only in ...	102	20	[go, jurong, point, crazy, available, bugis, n...
1	1	Ok lar Joking wif u oni	23	6	[ok, lar, joking, wif, u, oni]
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	149	28	[free, entry, in, 2, wkly, comp, win, fa, cup, fin...
3	1	U dun say so early hor U c already then say	43	11	[u, dun, say, so, early, hor, u, c, already, say]
4	1	Nah I dont think he goes to usf he lives aroun...	59	13	[nah, dont, think, goes, usf, lives, around, t...

Now let's add a new column to count the number of words after having removed the stop words.

```
In [ ]: # Column with the number of words for nltk tokens

def count_words_nltk(tokens):
    return len(tokens)
```

```
df['words_nltk'] = df['tokens_nltk'].apply(count_words_nltk)
df.head()
```

Out[]:

	spam	text	length	words	tokens_nltk	words_nltk
0	1	Go until jurong point crazy Available only in ...	102	20	[go, jurong, point, crazy, available, bugis, n...	16
1	1	Ok lar Joking wif u oni	23	6	[ok, lar, joking, wif, u, oni]	6
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	149	28	[free, entry, 2, wkly, comp, win, fa, cup, fin...	23
3	1	U dun say so early hor U c already then say	43	11	[u, dun, say, early, hor, u, c, already, say]	9
4	1	Nah I dont think he goes to usf he lives aroun...	59	13	[nah, dont, think, goes, usf, lives, around, t...	8

Note that the number of words decreased so we now have a cleaner text message.

Lemmatization:

In this step, Lemmatization uses a dictionary and morphological analysis to map a word to its base form, called a lemma. It considers the part of speech of the word and aims to identify the canonical form used in the dictionary. Lemmatization goes a step further, considering the part of speech (e.g., "better" becomes "good").

In []:

```
# Stemming/Lemmatization: using NLTK
from nltk.stem import WordNetLemmatizer

# Download necessary NLTK resources
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

def lemmatize_text(tokens):
    return [lemmatizer.lemmatize(t) for t in tokens]

df['lemmatized'] = df['tokens_nltk'].apply(lemmatize_text)

df.head()
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\carlo\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
```

	spam	text	length	words	tokens_nltk	words_nltk	lemmatized
0	1	Go until jurong point crazy Available only in ...	102	20	[go, jurong, point, crazy, available, bugis, n...]	16	[go, jurong, point, crazy, available, bugis, n...
1	1	Ok lar Joking wif u oni	23	6	[ok, lar, joking, wif, u, oni]	6	[ok, lar, joking, wif, u, oni]
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	149	28	[free, entry, 2, wkly, comp, win, fa, cup, fin...	23	[free, entry, 2, wkly, comp, win, fa, cup, fin...
3	1	U dun say so early hor U c already then say	43	11	[u, dun, say, early, hor, u, c, already, say]	9	[u, dun, say, early, hor, u, c, already, say]
4	1	Nah I dont think he goes to usf he lives aroun...	59	13	[nah, dont, think, goes, usf, lives, around, t...	8	[nah, dont, think, go, usf, life, around, though]

```
In [ ]: # Convert lemmatized tokens back to text
```

```
def tokens_to_text(tokens):
    return ' '.join(tokens)

df['lemmatized_text'] = df['lemmatized'].apply(tokens_to_text)
df.head()
```

	spam	text	length	words	tokens_nltk	words_nltk	lemmatized	lemmatized_text
0	1	Go until jurong point crazy Available only in ...	102	20	[go, jurong, point, crazy, available, bugis, n...]	16	[go, jurong, point, crazy, available, bugis, n...	go jurong point crazy available bugis n great ...
1	1	Ok lar Joking wif u oni	23	6	[ok, lar, joking, wif, u, oni]	6	[ok, lar, joking, wif, u, oni]	ok lar joking wif u oni
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	149	28	[free, entry, 2, wkly, comp, win, fa, cup, fin...	23	[free, entry, 2, wkly, comp, win, fa, cup, fin...	free entry 2 wkly comp win fa cup final tkts 2...
3	1	U dun say so early hor U c already then say	43	11	[u, dun, say, early, hor, u, c, already, say]	9	[u, dun, say, early, hor, u, c, already, say]	u dun say early hor u c already say
4	1	Nah I dont think he goes to usf he lives aroun...	59	13	[nah, dont, think, goes, usf, lives, around, t...	8	[nah, dont, think, go, usf, life, around, though]	nah dont think go usf life around though

3. Data visualization

```
In [ ]: from wordcloud import WordCloud
import matplotlib.pyplot as plt

df_spam = df[df['spam'] == 1]

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=600, background_color="white")

# Generate the word cloud
wordcloud.generate(df_spam['lemmatized_text'].to_string())
```

```
# Create a figure and display the word cloud
plt.figure(figsize=(8, 6))
plt.imshow(wordcloud)
plt.axis("off")
plt.title("Word Cloud for Ham Messages")
plt.show()
```



```
In [ ]: df_ham = df[df['spam'] == 0]

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=600, background_color="white")

# Generate the word cloud
wordcloud.generate(df_ham['lemmatized_text'].to_string())

# Create a figure and display the word cloud
plt.figure(figsize=(8, 6))
plt.imshow(wordcloud)
plt.axis("off")
plt.title("Word Cloud for Spam Messages")
plt.show()
```



N-grams:

Creating sequences of n words. Bigrams (2 words) and trigrams (3 words). N-grams can capture word relationships that single words might miss.

Single Words vs. Sequences: Individual words might not always convey the full meaning on their own. Bigrams (two-word sequences) and trigrams (three-word sequences) can capture the relationships between words, which can be crucial for tasks like:

- Sentiment Analysis: "not good" vs. "very good" expresses a clear difference in sentiment.
 - Topic Modeling: Identifying bigrams like "machine learning" or trigrams like "natural language processing" can help identify topics within text data.
 - Machine Translation: Capturing word order and phrasal meaning is essential for accurate translation.

In []: # N-grams

```
from nltk.util import ngrams

def extract_ngrams(tokens, n):
    return list(ngrams(tokens, n))

df['bigrams'] = df['lemmatized'].apply(lambda x: extract_ngrams(x, 2))
df['trigrams'] = df['lemmatized'].apply(lambda x: extract_ngrams(x, 3))

df.head()
```

Out[]:

	spam	text	length	words	tokens_nltk	words_nltk	lemmatized	lemmatized_text	bigrams	trigrams
0	1	Go until jurong point crazy Available only in ...	102	20	[go, jurong, point, crazy, available, bugis, n...]	16	[go, jurong, point, crazy, available, bugis, n...]	go jurong point crazy available bugis n great ...	[(go, jurong), (jurong, point), (point, jurong), (jurong, point), (point, crazy)...]	[(go, jurong), (jurong, point), (point, jurong), (jurong, point), (point, crazy), ...]
1	1	Ok lar Joking wif u oni	23	6	[ok, lar, joking, wif, u, oni]	6	[ok, lar, joking, wif, u, oni]	ok lar joking wif u oni	[(ok, lar), (lar, joking), (joking, (lar, joking, wif), (wif), (jokin...]	[(ok, lar, joking), (lar, joking, wif), (joking, wif), (jokin...]
2	0	Free entry in 2 a wkly comp to win FA Cup fina...	149	28	[free, entry, 2, wkly, comp, win, fa, cup, fin...]	23	[free, entry, 2, wkly, comp, win, fa, cup, fin...]	free entry 2 wkly comp win fa cup final tkts 2...	[(free, entry), (entry, 2), (2, wkly), (wkly), ...]	[(free, entry, 2), (entry, 2, wkly), (2, wkly), ...]
3	1	U dun say so early hor U c already then say	43	11	[u, dun, say, early, hor, u, c, already, say]	9	[u, dun, say, early, hor, u, c, already, say]	u dun say early hor u c already say	[(u, dun), (dun, say), (say, early), (early, h...]	[(u, dun, say), (dun, say, early), (say, early), (early, h...]
4	1	Nah I dont think he goes to usf he lives aroun...	59	13	[nah, dont, think, goes, usf, lives, around, t...]	8	[nah, dont, think, go, usf, life, around, though]	nah dont think go usf life around though	[(nah, dont), (dont, think), (think, go), (go,...]	[(nah, dont, think), (dont, think, go), (go,...)

Now we need to convert our text message into a numerical representation suitable for machine learning algorithms, and we have to options:

- TF-IDF: If you want to emphasize the importance of words that are frequent within a document but rare across the corpus (potentially good indicators of spam), TF-IDF might be a better choice.
- BoW: If a simple word count representation is sufficient for your task, BoW might be a more efficient option.

To potentially improve the accuracy of our spam/ham classification model, we can now incorporate TF-IDF (**Term Frequency-Inverse Document Frequency**). This technique assigns weights to words based on their frequency within a document and rarity across the entire corpus, potentially highlighting terms that are more indicative of spam text message.

In []:

```
# TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer

# Create a TfidfVectorizer object
vectorizer = TfidfVectorizer()
```

```

# Fit the vectorizer
vectorizer.fit(df['lemmatized_text'])

# Transform the text to a TF-IDF vector
X_tfidf = vectorizer.transform(df['lemmatized_text'])

```

Convert bigrams/trigrams lists into numerical representations:

Use CountVectorizer to create a sparse matrix where each row is a message and columns represent unique bigrams/trigrams with their counts.

Alternatively, apply feature hashing to bigrams/trigrams for a fixed-size vector representation.

```

In [ ]: from sklearn.feature_extraction.text import CountVectorizer

def custom_preprocessor(text):
    # Customize preprocessing for lists (e.g., join with a special character)
    return ' '.join(['_'.join(bigram) for bigram in text])

vectorizer_bigrams = CountVectorizer(preprocessor=custom_preprocessor)
vectorizer_bigrams.fit_transform(df['bigrams'])
X_bigrams = vectorizer_bigrams.transform(df['bigrams'])

vectorizer_trigrams = CountVectorizer(preprocessor=custom_preprocessor)
vectorizer_trigrams.fit_transform(df['trigrams'])
X_trigrams = vectorizer_trigrams.transform(df['trigrams'])

```

Concatenate these features into a single DataFrame or NumPy array for training your model

4. Modeling

Splitting the data

Now I will split the data into features/target variable and training and test sets.

```

In [ ]: from scipy.sparse import hstack

# Combine the sparse matrices
X_combined = hstack((X_tfidf, X_bigrams, X_trigrams))

y = df['spam']

```

```

In [ ]: from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.2, random_state=42)

```

Random forest

I will start by using `GridSearchCV` to tune a random forest model using parameter as follows:

- `estimator= rf`
- `param_grid= cv_params`
- `scoring= scoring`
- `cv: define the number of cross-validation folds you want (cv=_)`

- `refit`: indicate which evaluation metric you want to use to select the model (`refit=_`)

For this case I will set `refit` to `'f1'` given that:

In this case:

- Accuracy might be high if the model mostly predicts non-churn (majority class), even if it misses many actual churners (minority class).
- Precision is valuable to ensure customers flagged as likely to churn actually do churn, minimizing unnecessary interventions.
- Recall is crucial to identify as many churners as possible, even if it means some false positives (non-churners predicted as churn).
- F1-score provides a balance between precision and recall, giving a more holistic view of churn prediction performance.

```
In [ ]: # 1. Instantiate the random forest classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

rf = RandomForestClassifier(random_state=42)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [None],
              'max_features': [1.0],
              'max_samples': [1.0],
              'min_samples_leaf': [2],
              'min_samples_split': [2],
              'n_estimators': [300],
              }

# 3. Define a dictionary of scoring metrics to capture
scoring = ['accuracy', 'precision', 'recall', 'f1']

# 4. Instantiate the GridSearchCV object
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, cv=4, refit='f1')
```

```
In [ ]: %%time
rf_cv.fit(X_train, y_train)
```

CPU times: total: 15min 42s
Wall time: 44min 16s

```
Out[ ]: ▶      GridSearchCV      ⓘ ⓘ
  ▶ estimator: RandomForestClassifier
    ▶ RandomForestClassifier ⓘ
```

```
In [ ]: # Examine best score
rf_cv.best_score_
```

```
Out[ ]: 0.9788740229131649
```

Examine the best combination of hyperparameters.

```
In [ ]: # Examine best hyperparameter combo  
rf_cv.best_params_
```

```
Out[ ]: {'max_depth': None,  
         'max_features': 1.0,  
         'max_samples': 1.0,  
         'min_samples_leaf': 2,  
         'min_samples_split': 2,  
         'n_estimators': 300}
```

Use the `make_results()` function to output all of the scores of your model. Note that the function accepts three arguments.

```
In [ ]: def make_results(model_name:str, model_object, metric:str):  
    ...  
    Arguments:  
        model_name (string): what you want the model to be called in the output table  
        model_object: a fit GridSearchCV object  
        metric (string): precision, recall, f1, or accuracy  
  
    Returns a pandas df with the F1, recall, precision, and accuracy scores  
    for the model with the best mean 'metric' score across all validation folds.  
    ...  
  
    # Create dictionary that maps input metric to actual metric name in GridSearchCV  
    metric_dict = {'precision': 'mean_test_precision',  
                  'recall': 'mean_test_recall',  
                  'f1': 'mean_test_f1',  
                  'accuracy': 'mean_test_accuracy',  
                  }  
  
    # Get all the results from the CV and put them in a df  
    cv_results = pd.DataFrame(model_object.cv_results_)  
  
    # Isolate the row of the df with the max(metric) score  
    best_estimator_results = cv_results.iloc[cv_results[metric_dict[metric]].idxmax(), :]  
  
    # Extract accuracy, precision, recall, and f1 score from that row  
    f1 = best_estimator_results.mean_test_f1  
    recall = best_estimator_results.mean_test_recall  
    precision = best_estimator_results.mean_test_precision  
    accuracy = best_estimator_results.mean_test_accuracy  
  
    # Create table of results  
    table = pd.DataFrame({'model': [model_name],  
                          'precision': [precision],  
                          'recall': [recall],  
                          'F1': [f1],  
                          'accuracy': [accuracy],  
                          },  
                          )  
  
    return table
```

Pass the `GridSearch` object to the `make_results()` function.

```
In [ ]: results = make_results('RF cv', rf_cv, 'recall')  
results
```

	model	precision	recall	F1	accuracy
0	RF cv	0.967927	0.990074	0.978874	0.962515

XGBoost

As in Desicion Treen I will be using `GridSearchCV` to tune a random forest model using parameter as follows:

- `estimator= xgb`
- `param_grid= cv_params`
- `scoring= scoring`
- `cv: define the number of cross-validation folds you want (cv=_)`
- `refit: indicate which evaluation metric you want to use to select the model (refit=_)`

For this case I will also set `refit` to `'f1'` because of the resons I mentioned before.

```
In [ ]: # 1. Instantiate the XGBoost classifier
from xgboost import XGBClassifier

xgb = XGBClassifier(objective='binary:logistic', random_state=42, scale_pos_weight=10)

# 2. Create a dictionary of hyperparameters to tune
cv_params = {'max_depth': [6, 12],
             'min_child_weight': [3, 5],
             'learning_rate': [0.01, 0.1],
             'n_estimators': [300],
             'scale_pos_weight': [5, 10],
             }

# 3. Define a dictionary of scoring metrics to capture
scoring = ['accuracy', 'precision', 'recall', 'f1']

# 4. Instantiate the GridSearchCV object
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=4, refit='f1')
```

```
In [ ]: %%time
xgb_cv.fit(X_train, y_train)
```

CPU times: total: 2h 3min 43s
Wall time: 22min 45s

```
Out[ ]: ▶   GridSearchCV ⓘ ⓘ
          ▶ estimator: XGBClassifier
                  ▶ XGBClassifier
```

```
In [ ]: # Examine best score
xgb_cv.best_score_
```

```
Out[ ]: 0.9781287613411292
```

```
In [ ]: # Examine best parameters
xgb_cv.best_params_
```

```
Out[ ]: {'learning_rate': 0.1,
         'max_depth': 12,
         'min_child_weight': 3,
         'n_estimators': 300,
         'scale_pos_weight': 5}
```

```
In [ ]: # Call 'make_results()' on the GridSearch object
xgb_cv_results = make_results('XGB cv', xgb_cv, 'f1')
results = pd.concat([results, xgb_cv_results], axis=0)
results
```

	model	precision	recall	F1	accuracy
0	RF cv	0.967927	0.990074	0.978874	0.962515
0	XGB cv	0.964392	0.992280	0.978129	0.961062

5. Model Selection

Now, I will use the best random forest model and the best XGBoost model to predict on the test data. Whichever performs better will be selected as the champion model.

Random Forest

```
In [ ]: # Use random forest model to predict on test data

rf_test_preds = rf_cv.best_estimator_.predict(X_test)
```

The next function will help to generate a table with the scores from the predictions made

```
In [ ]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def get_test_scores(model_name:str, preds, y_test_data):
    """
    Generate a table of test scores.

    In:
        model_name (string): Your choice: how the model will be named in the output table
        preds: numpy array of test predictions
        y_test_data: numpy array of y_test data

    Out:
        table: a pandas df of precision, recall, f1, and accuracy scores for your model
    """

    accuracy = accuracy_score(y_test_data, preds)
    precision = precision_score(y_test_data, preds)
    recall = recall_score(y_test_data, preds)
    f1 = f1_score(y_test_data, preds)

    table = pd.DataFrame({'model': [model_name],
                          'precision': [precision],
                          'recall': [recall],
                          'F1': [f1],
                          'accuracy': [accuracy]
                         })

    return table
```

```
In [ ]: # Get validation scores for RF model
rf_val_scores = get_test_scores('RF Test', rf_test_preds, y_test)

# Append to the results table
results = pd.concat([results, rf_val_scores], axis=0)
results
```

```
Out[ ]:   model  precision      recall       F1  accuracy
0  RF cv  0.967927  0.990074  0.978874  0.962515
0  XGB cv  0.964392  0.992280  0.978129  0.961062
0  RF Test  0.967956  0.985377  0.976589  0.959381
```

XGBoost Classifier

```
In [ ]: # Use XGBoost model to predict on test data
xgb_test_preds = xgb_cv.best_estimator_.predict(X_test)

# Get test scores for XGBoost model
xgb_test_scores = get_test_scores('XGB test', xgb_test_preds, y_test)

# Append to the results table
results = pd.concat([results, xgb_test_scores], axis=0)
results
```

Out[]:

	model	precision	recall	F1	accuracy
0	RF cv	0.967927	0.990074	0.978874	0.962515
0	XGB cv	0.964392	0.992280	0.978129	0.961062
0	RF Test	0.967956	0.985377	0.976589	0.959381
0	XGB test	0.961790	0.991001	0.976177	0.958414

Noticed that the Champion Model here is the `Random Forest Classifier` given that the final scores indicate that performed better

6. Use champion model to predict on test data

In []:

```
# Use XGBoost model to predict on test data

# xgb_cv.best_estimator_ = Loaded_model # Use this Line to Load the model from disk

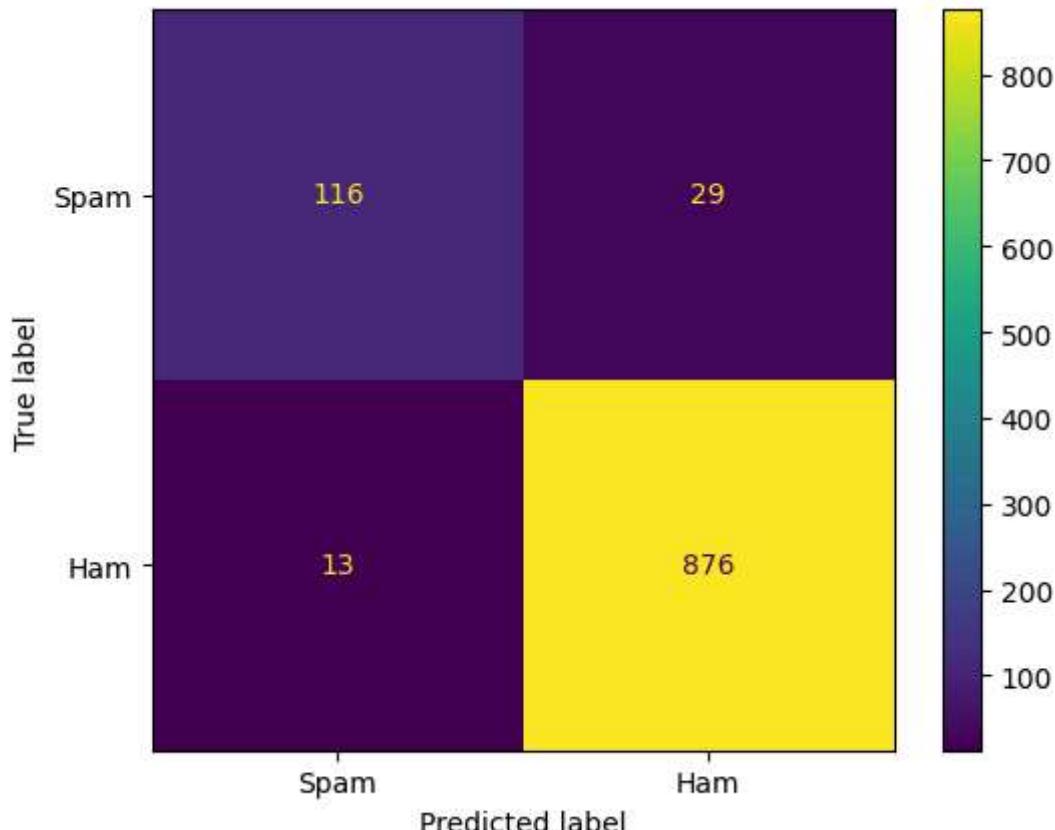
#xgb_test_preds = xgb_cv.best_estimator_.predict(X_test)
rf_test_preds = rf_cv.best_estimator_.predict(X_test)
```

In []:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate array of values for confusion matrix
cm = confusion_matrix(y_test, rf_test_preds, labels=xgb_cv.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                               display_labels=['Spam', 'Ham'])
disp.plot();
```



Values in the Matrix:

Cell (0, 0): True Positive (TP) - Number of correctly classified spam text message.

Cell (0, 1): False Negative (FN) - Number of spam text message incorrectly classified as ham.

Cell (1, 0): False Positive (FP) - Number of ham text message incorrectly classified as spam.

Cell (1, 1): True Negative (TN) - Number of correctly classified ham text message.

Conclusion:

- High TP and Low FP: The model effectively identifies **116** spam text message with minimal **13** false positives (correctly classifying most actual spam).
- High TN and Low FN: The model accurately classifies **876** ham text message, avoiding false **29** negatives (missing very little actual spam).

Since the model took around 60 minutes to be trained I will save it using `Pickle` library.

```
In [ ]: # # Save the model to disk  
# filename = 'xgb_model.sav'  
# pickle.dump(xgb_cv.best_estimator_, open(filename, 'wb'))
```