CMSC 197 Machine Problem 3: Naive Bayes Spam Filter

Author: Chester Ken Gallego

Date: October 8, 2024

Link to the github repository: https://github.com/chstrkn/CMSC197

Method of Implementation

Preprocessing

In [5]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import re
import email
import codecs
from sklearn import metrics
from sklearn.metrics import accuracy_score, recall_score, precision_score
```

In [6]:
```
# Initialize the main dataframe
df = pd.DataFrame(columns = ['folder', 'file', 'email_message', 'classification'])

# Initialize the labels dataframe
labels_path = "trec06p-cs280/labels"
df_labels = pd.read_csv(labels_path, sep=" ", header=None)

# Define columns for the labels dataframe
df_labels.columns = ["classification", "filepath"]

# Convert 'ham' to 0 and 'spam' to 1
df_labels["classification"] = df_labels["classification"].apply(lambda kv: 0 if kv

# Remove "../data/" from the filepath
df_labels["filepath"] = df_labels["filepath"].apply(lambda kv: kv.replace("../data/
df_labels
```

Out[6]:

	classification	filepath
0	0	000/000
1	1	000/001
2	1	000/002
3	0	000/003
4	1	000/004
...	...	...
37817	1	126/017
37818	1	126/018
37819	1	126/019
37820	1	126/020
37821	1	126/021

37822 rows × 2 columns

In [7]:
```
# List the folders in the data directory
folder_path = "trec06p-cs280/data"
folders = os.listdir(folder_path)
folders
```

Out[7]:
```
['000',
 '001',
 '002',
 '003',
 '004',
 '005',
 '006',
 '007',
 '008',
 '009',
 '010',
 '011',
 '012',
 '013',
 '014',
 '015',
 '016',
 '017',
 '018',
 '019',
 '020',
 '021',
 '022',
 '023',
 '024',
 '025',
 '026',
 '027',
 '028',
 '029',
 '030',
 '031',
 '032',
 '033',
 '034',
 '035',
 '036',
 '037',
 '038',
 '039',
 '040',
 '041',
 '042',
 '043',
 '044',
 '045',
 '046',
 '047',
 '048',
 '049',
 '050',
 '051',
 '052',
 '053',
 '054',
 '055',
```

```
'056',
'057',
'058',
'059',
'060',
'061',
'062',
'063',
'064',
'065',
'066',
'067',
'068',
'069',
'070',
'071',
'072',
'073',
'074',
'075',
'076',
'077',
'078',
'079',
'080',
'081',
'082',
'083',
'084',
'085',
'086',
'087',
'088',
'089',
'090',
'091',
'092',
'093',
'094',
'095',
'096',
'097',
'098',
'099',
'100',
'101',
'102',
'103',
'104',
'105',
'106',
'107',
'108',
'109',
'110',
'111',
```

```
 '112',
 '113',
 '114',
 '115',
 '116',
 '117',
 '118',
 '119',
 '120',
 '121',
 '122',
 '123',
 '124',
 '125',
 '126']
```

In [8]:

```python
# Get stop words
stop_words = []
with open('stop_words.txt', 'r') as file:
    stop_words_list = file.read().splitlines()
    stop_words = [word for word in stop_words_list]
stop_words
```

Out[8]:
```
['a',
 'able',
 'about',
 'above',
 'abst',
 'accordance',
 'according',
 'accordingly',
 'across',
 'act',
 'actually',
 'added',
 'adj',
 'adopted',
 'affected',
 'affecting',
 'affects',
 'after',
 'afterwards',
 'again',
 'against',
 'ah',
 'all',
 'almost',
 'alone',
 'along',
 'already',
 'also',
 'although',
 'always',
 'am',
 'among',
 'amongst',
 'an',
 'and',
 'announce',
 'another',
 'any',
 'anybody',
 'anyhow',
 'anymore',
 'anyone',
 'anything',
 'anyway',
 'anyways',
 'anywhere',
 'apparently',
 'approximately',
 'are',
 'aren',
 'arent',
 'arise',
 'around',
 'as',
 'aside',
 'ask',
```

```
'asking',
'at',
'auth',
'available',
'away',
'awfully',
'b',
'back',
'be',
'became',
'because',
'become',
'becomes',
'becoming',
'been',
'before',
'beforehand',
'begin',
'beginning',
'beginnings',
'begins',
'behind',
'being',
'believe',
'below',
'beside',
'besides',
'between',
'beyond',
'biol',
'both',
'brief',
'briefly',
'but',
'by',
'c',
'ca',
'came',
'can',
'cannot',
"can't",
'cause',
'causes',
'certain',
'certainly',
'co',
'com',
'come',
'comes',
'contain',
'containing',
'contains',
'could',
'couldnt',
'd',
'date',
```

```
'did',
"didn't",
'different',
'do',
'does',
"doesn't",
'doing',
'done',
"don't",
'down',
'downwards',
'due',
'during',
'e',
'each',
'ed',
'edu',
'effect',
'eg',
'eight',
'eighty',
'either',
'else',
'elsewhere',
'end',
'ending',
'enough',
'especially',
'et',
'et-al',
'etc',
'even',
'ever',
'every',
'everybody',
'everyone',
'everything',
'everywhere',
'ex',
'except',
'f',
'far',
'few',
'ff',
'fifth',
'first',
'five',
'fix',
'followed',
'following',
'follows',
'for',
'former',
'formerly',
'forth',
'found',
```

```
'four',
'from',
'further',
'furthermore',
'g',
'gave',
'get',
'gets',
'getting',
'give',
'given',
'gives',
'giving',
'go',
'goes',
'gone',
'got',
'gotten',
'h',
'had',
'happens',
'hardly',
'has',
"hasn't",
'have',
"haven't",
'having',
'he',
'hed',
'hence',
'her',
'here',
'hereafter',
'hereby',
'herein',
'heres',
'hereupon',
'hers',
'herself',
'hes',
'hi',
'hid',
'him',
'himself',
'his',
'hither',
'home',
'how',
'howbeit',
'however',
'hundred',
'i',
'id',
'ie',
'if',
"i'll",
```

```
'im',
'immediate',
'immediately',
'importance',
'important',
'in',
'inc',
'indeed',
'index',
'information',
'instead',
'into',
'invention',
'inward',
'is',
"isn't",
'it',
'itd',
"it'll",
'its',
'itself',
"i've",
'j',
'just',
'k',
'keep',
'keeps',
'kept',
'keys',
'kg',
'km',
'know',
'known',
'knows',
'l',
'largely',
'last',
'lately',
'later',
'latter',
'latterly',
'least',
'less',
'lest',
'let',
'lets',
'like',
'liked',
'likely',
'line',
'little',
"'ll",
'look',
'looking',
'looks',
'ltd',
```

```
'm',
'made',
'mainly',
'make',
'makes',
'many',
'may',
'maybe',
'me',
'mean',
'means',
'meantime',
'meanwhile',
'merely',
'mg',
'might',
'million',
'miss',
'ml',
'more',
'moreover',
'most',
'mostly',
'mr',
'mrs',
'much',
'mug',
'must',
'my',
'myself',
'n',
'na',
'name',
'namely',
'nay',
'nd',
'near',
'nearly',
'necessarily',
'necessary',
'need',
'needs',
'neither',
'never',
'nevertheless',
'new',
'next',
'nine',
'ninety',
'no',
'nobody',
'non',
'none',
'nonetheless',
'noone',
'nor',
```

```
'normally',
'nos',
'not',
'noted',
'nothing',
'now',
'nowhere',
'o',
'obtain',
'obtained',
'obviously',
'of',
'off',
'often',
'oh',
'ok',
'okay',
'old',
'omitted',
'on',
'once',
'one',
'ones',
'only',
'onto',
'or',
'ord',
'other',
'others',
'otherwise',
'ought',
'our',
'ours',
'ourselves',
'out',
'outside',
'over',
'overall',
'owing',
'own',
'p',
'page',
'pages',
'part',
'particular',
'particularly',
'past',
'per',
'perhaps',
'placed',
'please',
'plus',
'poorly',
'possible',
'possibly',
'potentially',
```

```
'pp',
'predominantly',
'present',
'previously',
'primarily',
'probably',
'promptly',
'proud',
'provides',
'put',
'q',
'que',
'quickly',
'quite',
'qv',
'r',
'ran',
'rather',
'rd',
're',
'readily',
'really',
'recent',
'recently',
'ref',
'refs',
'regarding',
'regardless',
'regards',
'related',
'relatively',
'research',
'respectively',
'resulted',
'resulting',
'results',
'right',
'run',
's',
'said',
'same',
'saw',
'say',
'saying',
'says',
'sec',
'section',
'see',
'seeing',
'seem',
'seemed',
'seeming',
'seems',
'seen',
'self',
'selves',
```

```
'sent',
'seven',
'several',
'shall',
'she',
'shed',
"she'll",
'shes',
'should',
"shouldn't",
'show',
'showed',
'shown',
'showns',
'shows',
'significant',
'significantly',
'similar',
'similarly',
'since',
'six',
'slightly',
'so',
'some',
'somebody',
'somehow',
'someone',
'somethan',
'something',
'sometime',
'sometimes',
'somewhat',
'somewhere',
'soon',
'sorry',
'specifically',
'specified',
'specify',
'specifying',
'state',
'states',
'still',
'stop',
'strongly',
'sub',
'substantially',
'successfully',
'such',
'sufficiently',
'suggest',
'sup',
'sure',
't',
'take',
'taken',
'taking',
```

```
'tell',
'tends',
'th',
'than',
'thank',
'thanks',
'thanx',
'that',
"that'll",
'thats',
"that've",
'the',
'their',
'theirs',
'them',
'themselves',
'then',
'thence',
'there',
'thereafter',
'thereby',
'thered',
'therefore',
'therein',
"there'll",
'thereof',
'therere',
'theres',
'thereto',
'thereupon',
"there've",
'these',
'they',
'theyd',
"they'll",
'theyre',
"they've",
'think',
'this',
'those',
'thou',
'though',
'thoughh',
'thousand',
'throug',
'through',
'throughout',
'thru',
'thus',
'til',
'tip',
'to',
'together',
'too',
'took',
'toward',
```

```
'towards',
'tried',
'tries',
'truly',
'try',
'trying',
'ts',
'twice',
'two',
'u',
'un',
'under',
'unfortunately',
'unless',
'unlike',
'unlikely',
'until',
'unto',
'up',
'upon',
'ups',
'us',
'use',
'used',
'useful',
'usefully',
'usefulness',
'uses',
'using',
'usually',
'v',
'value',
'various',
"'ve",
'very',
'via',
'viz',
'vol',
'vols',
'vs',
'w',
'want',
'wants',
'was',
"wasn't",
'way',
'we',
'wed',
'welcome',
"we'll",
'went',
'were',
"weren't",
"we've",
'what',
'whatever',
```

```
"what'll",
'whats',
'when',
'whence',
'whenever',
'where',
'whereafter',
'whereas',
'whereby',
'wherein',
'wheres',
'whereupon',
'wherever',
'whether',
'which',
'while',
'whim',
'whither',
'who',
'whod',
'whoever',
'whole',
"who'll",
'whom',
'whomever',
'whos',
'whose',
'why',
'widely',
'willing',
'wish',
'with',
'within',
'without',
"won't",
'words',
'world',
'would',
"wouldn't",
'www',
'x',
'y',
'yes',
'yet',
'you',
'youd',
"you'll",
'your',
'youre',
'yours',
'yourself',
'yourselves',
"you've",
'z',
'zero']
```

In [9]:
```python
# Function to drop useless information from the email
def drop_info(msg):
    # Convert the message to lowercase
    msg = msg.lower()
    # Remove all characters except letters and spaces
    msg = re.sub(r'[^a-zA-Z\s]', '', msg)
    # Split the message into a list of words
    words = msg.split()
    # Remove stop words
    words = [word for word in words if word not in stop_words]
    # Join the words back into a single message
    msg = " ".join(words)
    msg = msg.strip("'")
    return msg

# Function to extract the message from a parsed email
def get_msg(parsed):
    msg = ""
    # Iterate over the parts of the email if it is a multipart email
    if parsed.is_multipart():
        for part in parsed.walk():
            if part.get_content_type() == 'text/plain':
                msg = part.get_payload()
                break
    else:
        msg = parsed.get_payload()
    return msg
```

In [10]:
```python
# Read each email and store them in the main dataframe
for folder in folders:
    # Get the files inside the folder
    files = os.listdir(f"{folder_path}/{folder}")
    for file in files:
        # Open the email file, preprocessed in the standard character set ISO-8859-
        with open(f"{folder_path}/{folder}/{file}", "r", encoding="ISO-8859-1") as
            read_email = e_mail.read()
            parsed_email = email.message_from_string(read_email)
            msg = get_msg(parsed_email)
            msg = drop_info(msg)
            # Get the classification of the email based on the labels dataframe
            labels_classification = df_labels[df_labels['filepath'] == f"{folder}/{
            # Concatenate the data into the main dataframe
            df = pd.concat([df, pd.DataFrame([[folder, file, msg, labels_classifica
df
```

Out[10]:

	folder	file	email_message	classification
0	000	000	mailing list queried weeks ago running set arc...	0
1	000	001	luxury watches buy rolex rolex cartier bvlgari...	1
2	000	002	academic qualifications prestigious nonacc red...	1
3	000	003	greetings verify subscription planfans list ch...	0
4	000	004	chauncey conferred luscious continued tonsillitis	1
...	...	...	...	...
37817	126	017	great news expec ted infinex ventures infx pri...	1
37818	126	018	oil sector going crazy weekly gift kkpt thing ...	1
37819	126	019	httpvdtobjdocscaninfo suffering pain depressio...	1
37820	126	020	prosperous future increased money earning powe...	1
37821	126	021	moat coverall cytochemistry planeload salk	1

37822 rows × 4 columns

In [11]:

```
# Save the preprocessed emails to a CSV file
df.to_csv("preprocessed_emails.csv", index=False)

# Read the preprocessed emails from the CSV file
df2 = pd.read_csv("preprocessed_emails.csv")

# Split the dataset into training and testing sets
# Folders 0-70: Training Set
training_df = df2[df2['folder'] < 71]
# Folders 71-126: Testing Set
testing_df = df2[df2['folder'] >= 71]

# Classify training set into ham and spam
training_ham_df = training_df[training_df['classification'] == 0]
training_spam_df = training_df[training_df['classification'] == 1]

df2
```

Out[11]:

	folder	file	email_message	classification
0	0	0	mailing list queried weeks ago running set arc...	0
1	0	1	luxury watches buy rolex rolex cartier bvlgari...	1
2	0	2	academic qualifications prestigious nonacc red...	1
3	0	3	greetings verify subscription planfans list ch...	0
4	0	4	chauncey conferred luscious continued tonsillitis	1
...	...	...	...	...
37817	126	17	great news expec ted infinex ventures infx pri...	1
37818	126	18	oil sector going crazy weekly gift kkpt thing ...	1
37819	126	19	httpvdtobjdocscaninfo suffering pain depressio...	1
37820	126	20	prosperous future increased money earning powe...	1
37821	126	21	moat coverall cytochemistry planeload salk	1

37822 rows × 4 columns

In [12]:
```
training_ham_df
```

Out[12]:

	folder	file	email_message	classification
0	0	0	mailing list queried weeks ago running set arc...	0
3	0	3	greetings verify subscription planfans list ch...	0
5	0	5	quiet quiet well straw poll plan running	0
6	0	6	working departed totally bell labs recommended...	0
10	0	10	greetings mass acknowledgement signed planfans...	0
...	...	...	...	...
21270	70	270	equation generate prime numbers equation theor...	0
21271	70	271	equation generate prime numbers equation theor...	0
21288	70	288	dear dmdx users guidance generating dmdx item ...	0
21293	70	293	built handyboard works great testmotor passes ...	0
21298	70	298	mounted isu infrared demodulator hb realised r...	0

7523 rows × 4 columns

In [13]:
```
training_spam_df
```

Out[13]:

	folder	file	email_message	classification
1	0	1	luxury watches buy rolex rolex cartier bvlgari...	1
2	0	2	academic qualifications prestigious nonacc red...	1
4	0	4	chauncey conferred luscious continued tonsillitis	1
7	0	7	nbc today body diet beaches magazines hollywoo...	1
8	0	8	oil sector going crazy weekly gift kkpt thing ...	1
...	...	...	...	...
21294	70	294	txtadd	1
21295	70	295	btijclnab binpqnejgmb httpgethighbizez bldb xi...	1
21296	70	296	special offer adobe video collection adobe pre...	1
21297	70	297	doctype html public wcdtd html transitionalen ...	1
21299	70	299	httptmqmctoverpacenet suffering pain depressio...	1

13777 rows × 4 columns

In [14]:
```
testing_df
```

Out[14]:

	folder	file	email_message	classification
21300	71	0	hesitantly derive perverse satisfaction clodho...	1
21301	71	1	things perform experiment display will remain ...	0
21302	71	2	best offer month viggra ci ialis vaiium xa naa...	1
21303	71	3	de ar wne cr doesnt matter ow real st mmed ia ...	1
21304	71	4	special offer adobe video collection adobe pre...	1
...	...	...	...	...
37817	126	17	great news expec ted infinex ventures infx pri...	1
37818	126	18	oil sector going crazy weekly gift kkpt thing ...	1
37819	126	19	httpvdtobjdocscaninfo suffering pain depressio...	1
37820	126	20	prosperous future increased money earning powe...	1
37821	126	21	moat coverall cytochemistry planeload salk	1

16522 rows × 4 columns

In [15]:
```
# Dictionary where the key is the word and the value is the word count
word_count = {}

for index, row in training_df.iterrows():
```

```
    for word in str(row['email_message']).split():
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1

# Sort the dictionary by value in descending order
sorted_word_count = sorted(word_count.items(), key=lambda kv: kv[1], reverse=True)

# Extract the 10,000 most common words
most_common_words = dict(sorted_word_count[:10000])
most_common_words
```

Out[15]:

```
{'bb': 16763,
 'td': 12700,
 'will': 11322,
 'tr': 10017,
 'br': 5751,
 'width': 5392,
 'board': 5171,
 'company': 4403,
 'price': 4391,
 'email': 4051,
 'list': 3935,
 'gold': 3922,
 'nil': 3830,
 'time': 3816,
 'help': 3779,
 'send': 3650,
 'message': 3589,
 'dont': 3571,
 'subject': 3567,
 'size': 3541,
 'adobe': 3490,
 'crustl': 3295,
 'body': 3259,
 'received': 3091,
 'program': 3081,
 'html': 3076,
 'work': 2778,
 'wrote': 2670,
 'ms': 2664,
 'well': 2647,
 'professional': 2628,
 'border': 2516,
 'good': 2476,
 'number': 2469,
 'university': 2437,
 'widthfont': 2414,
 'problem': 2360,
 'table': 2300,
 'widthtd': 2294,
 'file': 2245,
 'stock': 2243,
 'handyboard': 2231,
 'hb': 2202,
 'font': 2199,
 'color': 2198,
 'bit': 2191,
 'office': 2161,
 'de': 2160,
 'info': 2131,
 'windows': 2094,
 'add': 2062,
 'microsoft': 2060,
 'current': 2054,
 'studies': 2004,
 'contenttype': 1973,
 'news': 1968,
```

```
'code': 1955,
'development': 1935,
'corp': 1930,
'pro': 1926,
'find': 1923,
'china': 1918,
'head': 1893,
'womens': 1887,
'great': 1850,
'pt': 1804,
'read': 1797,
'system': 1775,
'today': 1763,
'best': 1763,
'people': 1756,
'power': 1753,
'motor': 1745,
'call': 1737,
'ic': 1723,
'save': 1702,
'fax': 1700,
'text': 1693,
'height': 1654,
'handy': 1636,
'unsubscribe': 1631,
'data': 1628,
'days': 1617,
'address': 1611,
'ive': 1602,
'mail': 1574,
'set': 1569,
'div': 1565,
'site': 1535,
'market': 1523,
'textplain': 1514,
'free': 1511,
'faceverdana': 1488,
'better': 1484,
'life': 1470,
'cellpadding': 1460,
'additional': 1446,
'xp': 1446,
'oil': 1445,
'meta': 1433,
'web': 1426,
'big': 1417,
'port': 1415,
'control': 1404,
'offer': 1398,
'software': 1394,
'majordomovimsedu': 1386,
'years': 1385,
'money': 1385,
'day': 1366,
'cellspacing': 1364,
'computer': 1363,
```

```
'campaign': 1356,
'rating': 1351,
'gas': 1346,
'reviews': 1335,
'retail': 1330,
'forward': 1321,
'cart': 1304,
'contenttransferencoding': 1299,
'robot': 1289,
'technology': 1272,
'experience': 1242,
'service': 1236,
'order': 1235,
'week': 1231,
'ra': 1223,
'energy': 1220,
'messages': 1213,
'version': 1209,
'potential': 1206,
'project': 1198,
'based': 1196,
'exploration': 1192,
'going': 1185,
'mimeversion': 1177,
'long': 1174,
'real': 1172,
'special': 1165,
'problems': 1163,
'cantex': 1160,
'things': 1159,
'thu': 1157,
'wmstlumddumdedu': 1156,
'full': 1150,
'motors': 1143,
'provide': 1142,
'pm': 1140,
'helvetica': 1136,
'post': 1127,
'high': 1126,
'start': 1121,
'report': 1109,
'thing': 1106,
'doesnt': 1097,
'unicode': 1097,
'question': 1097,
'sender': 1096,
'international': 1095,
'business': 1092,
'works': 1089,
'replyto': 1087,
'httphyakushowcomtada': 1072,
'small': 1070,
'files': 1061,
'support': 1060,
'digital': 1044,
'windowtext': 1044,
```

```
'working': 1041,
'serial': 1038,
'check': 1036,
'space': 1034,
'yew': 1030,
'tue': 1027,
'mon': 1025,
'expansion': 1024,
'cant': 1021,
'sep': 1018,
'input': 1011,
'style': 1009,
'span': 1009,
'currently': 1007,
'contact': 1005,
'department': 1004,
'original': 1000,
'account': 998,
'science': 997,
'point': 995,
'aug': 991,
'plan': 989,
'video': 989,
'colorred': 984,
'jul': 982,
'online': 978,
'interested': 975,
'receive': 968,
'change': 965,
'txtadd': 965,
'second': 959,
'infinex': 956,
'running': 952,
'sizefonttd': 948,
'design': 942,
'download': 940,
'website': 937,
'apple': 935,
'charsetiso': 931,
'error': 930,
'fri': 930,
'short': 929,
'dragon': 926,
'charsetusascii': 923,
'px': 922,
'output': 921,
'arial': 915,
'approved': 914,
'place': 914,
'side': 911,
'technologies': 909,
'claims': 904,
'properties': 903,
'messageid': 903,
'year': 903,
'low': 894,
```

'investors': 890,
'fred': 890,
'commands': 886,
'sensor': 877,
'students': 873,
'times': 864,
'golden': 864,
'access': 863,
'light': 863,
'phone': 861,
'dear': 861,
'release': 859,
'timeout': 859,
'companys': 853,
'option': 852,
'services': 850,
'aligncenter': 850,
'group': 842,
'including': 842,
'nan': 841,
'battery': 840,
'esmtp': 840,
'hope': 838,
'note': 838,
'facearial': 836,
'course': 833,
'questions': 831,
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa': 829,
'crustlvimsedu': 826,
'three': 823,
'mm': 823,
'fine': 819,
'systems': 818,
'form': 816,
'test': 816,
'ir': 814,
'fast': 812,
'internet': 812,
'application': 808,
'hello': 807,
'plain': 802,
'include': 801,
'product': 800,
'st': 799,
'turn': 796,
'watch': 795,
'servo': 794,
'kind': 792,
'standard': 789,
'pleased': 787,
'case': 787,
'format': 785,
'apr': 785,
'future': 783,
'idea': 780,
'handyboardmediamitedu': 776,

```
'pin': 775,
'pc': 771,
'mode': 771,
'pdt': 770,
'memory': 770,
'chip': 770,
'hours': 768,
'common': 768,
'pine': 768,
'corporation': 763,
'write': 756,
'women': 751,
'sonar': 748,
'opportunities': 745,
'starting': 743,
'httpequivcontenttype': 738,
'prices': 736,
'limited': 734,
'effects': 734,
'class': 733,
'statements': 733,
'committee': 733,
'widthwindowstd': 731,
'voltage': 730,
'img': 729,
'complete': 728,
'edt': 728,
'contenttexthtml': 725,
'friday': 723,
'valigntop': 716,
'store': 715,
'companies': 713,
'buy': 712,
'lot': 712,
'prospect': 712,
'cs': 710,
'process': 707,
'archives': 706,
'sun': 706,
'press': 705,
'example': 703,
'sansserif': 703,
'production': 701,
'share': 700,
'type': 699,
'agreement': 696,
'simply': 696,
'lcd': 696,
'joint': 695,
'center': 695,
'john': 694,
'bldkb': 693,
'analog': 693,
'longer': 691,
'large': 688,
'seismic': 688,
```

```
'link': 685,
'visit': 680,
'tel': 676,
'book': 669,
'shares': 664,
'area': 664,
'ventures': 662,
'ago': 660,
'sensors': 659,
'mar': 658,
'source': 656,
'field': 655,
'answer': 655,
'rolex': 654,
'feel': 654,
'private': 654,
'issue': 654,
'usa': 650,
'hc': 645,
'investment': 644,
'allow': 643,
'photoshop': 639,
'delay': 639,
'opportunity': 638,
'launch': 637,
'word': 636,
'click': 634,
'nov': 633,
'update': 632,
'int': 632,
'simple': 631,
'april': 628,
'drive': 626,
'solution': 626,
'announces': 625,
'building': 625,
'engineering': 625,
'main': 624,
'mailing': 623,
'shipping': 623,
'advance': 622,
'record': 621,
'san': 621,
'httplovematchbzpc': 620,
'general': 618,
'matter': 618,
'sincerely': 618,
'student': 618,
'third': 616,
'return': 615,
'shareholder': 612,
'programs': 612,
'effective': 611,
'fact': 611,
'monday': 611,
'wrong': 609,
```

```
'products': 609,
'thought': 608,
'connected': 608,
'characters': 608,
'discussion': 607,
'natural': 606,
'left': 606,
'school': 604,
'canyon': 603,
'true': 602,
'men': 598,
'worlds': 597,
'lines': 597,
'interest': 596,
'north': 594,
'texas': 594,
'trade': 593,
'collection': 593,
'ill': 593,
'easy': 593,
'cc': 592,
'mineral': 590,
'hard': 589,
'history': 589,
'led': 589,
'function': 588,
'extension': 588,
'smtp': 587,
'directly': 587,
'connect': 587,
'youll': 585,
'build': 584,
'jan': 584,
'edition': 582,
'management': 582,
'httpwwwvimsedujeffarchivehtm': 581,
'reading': 580,
'dr': 579,
'details': 579,
'signal': 579,
'starshipdesignlistsuoregonedu': 579,
'numbers': 578,
'acrobat': 576,
'swath': 574,
'started': 573,
'pfont': 572,
'normal': 570,
'device': 570,
'display': 570,
'alignleft': 566,
'offers': 565,
'dec': 565,
'mining': 564,
'conference': 563,
'weight': 562,
'processing': 562,
```

```
'character': 561,
'split': 561,
'public': 561,
'top': 559,
'heighttd': 559,
'deal': 559,
'interface': 558,
'engaged': 557,
'box': 557,
'independent': 557,
'increase': 557,
'weeks': 555,
'machine': 555,
'degree': 555,
'speed': 552,
'widthd': 550,
'dmdx': 550,
'huge': 549,
'position': 549,
'manager': 549,
'choice': 548,
'expect': 547,
'developing': 547,
'fontsize': 547,
'didnt': 543,
'premiere': 542,
'fully': 542,
'reply': 541,
'pretty': 540,
'circuit': 540,
'close': 539,
'play': 537,
'la': 537,
'beeddddeeb': 536,
'library': 535,
'west': 535,
'lose': 535,
'sell': 535,
'making': 534,
'move': 533,
'president': 533,
'minerals': 532,
'widtha': 532,
'starshipdesign': 532,
'open': 528,
'remember': 527,
'credit': 527,
'quality': 526,
'watches': 525,
'response': 524,
'total': 523,
'mike': 523,
'language': 522,
'paper': 521,
'meeting': 521,
'operations': 519,
```

```
'june': 519,
'located': 518,
'turned': 516,
'server': 515,
'te': 515,
'havent': 513,
'sharp': 513,
'local': 511,
'resources': 511,
'producer': 510,
'american': 508,
'servos': 508,
'cost': 507,
'cpu': 505,
'required': 505,
'luck': 505,
'correct': 505,
'communication': 503,
'gmt': 503,
'range': 500,
'supply': 500,
'classtext': 500,
'load': 499,
'held': 498,
'water': 498,
'called': 497,
'needed': 496,
'result': 496,
'college': 495,
'island': 494,
'pa': 494,
'major': 492,
'loss': 490,
'shareholders': 489,
'reference': 488,
'applications': 487,
'announced': 486,
'early': 486,
'country': 485,
'customer': 485,
'issues': 485,
'network': 484,
'mime': 480,
'advice': 480,
'consider': 477,
'expected': 476,
'directors': 476,
'request': 476,
'driver': 476,
'cwtd': 476,
'risk': 475,
'returnpath': 475,
'parts': 475,
'appreciated': 472,
'america': 472,
'cash': 471,
```

```
'copy': 471,
'venture': 471,
'dc': 471,
'cialis': 471,
'happy': 469,
'focus': 469,
'level': 469,
'highly': 468,
'ports': 468,
'strong': 467,
'marketing': 466,
'interesting': 464,
'unicodeunicodeorg': 464,
'south': 461,
'ability': 461,
'pay': 461,
'sat': 460,
'multiple': 460,
'greko': 460,
'suggestions': 458,
'members': 457,
'user': 457,
'phd': 457,
'director': 456,
'tuesday': 456,
'media': 456,
'valigndtop': 456,
'tenure': 455,
'fontfamily': 455,
'industry': 454,
'stuff': 454,
'understand': 454,
'verified': 454,
'wondering': 454,
'lego': 454,
'sector': 451,
'knowledge': 450,
'growth': 450,
'antonio': 450,
'soft': 450,
'securities': 447,
'ascii': 447,
'bad': 446,
'coming': 446,
'bjegsb': 446,
'written': 445,
'august': 445,
'drilling': 444,
'ross': 444,
'encore': 443,
'programming': 443,
'books': 442,
'month': 442,
'unit': 442,
'interactive': 442,
'bestsellers': 441,
```

```
'mhz': 441,
'boards': 441,
'names': 440,
'loan': 440,
'ownerstarshipdesignlistsuoregonedu': 440,
'darkwinguoregonedu': 440,
'dvd': 438,
'unicodeorg': 438,
'card': 437,
'screen': 437,
'inreplyto': 436,
'bank': 434,
'bfonttd': 434,
'key': 432,
'audition': 432,
'hardware': 432,
'series': 432,
'jonathan': 432,
'person': 431,
'command': 431,
'july': 430,
'pins': 430,
'nowrap': 430,
'changed': 429,
'friend': 429,
'changes': 428,
'clixme': 426,
'volume': 425,
'job': 424,
'term': 424,
'classdmsonormalif': 424,
'supportemptyparasnbspendifspan': 424,
'styledfontsizeptmsobidifontsizeptopopspan': 424,
'ideas': 423,
'talk': 423,
'aa': 423,
'exciting': 422,
'green': 421,
'love': 421,
'martin': 419,
'papers': 419,
'pcode': 418,
'delivery': 417,
'rate': 417,
'trace': 416,
'institute': 416,
'val': 415,
'isnt': 414,
'unique': 414,
'est': 414,
'sex': 412,
'room': 411,
'difficult': 411,
'orgasms': 411,
'national': 411,
'texada': 410,
```

```
'nevada': 409,
'msfd': 409,
'allows': 405,
'team': 405,
'franklin': 405,
'global': 405,
'manual': 404,
'cr': 404,
'unix': 403,
'tbody': 403,
'xmailer': 403,
'charge': 402,
'formula': 402,
'charsetwindows': 401,
'man': 400,
'wont': 400,
'couple': 400,
'continue': 400,
'writes': 399,
'trading': 398,
'create': 398,
'aligncenterbfont': 397,
'acres': 396,
'single': 396,
'graduate': 395,
'methods': 395,
'gt': 395,
'search': 394,
'cable': 393,
'register': 392,
'users': 392,
'higher': 391,
'colorwhite': 391,
'exactly': 387,
'colorgraybadobe': 387,
'texthtml': 386,
'geological': 386,
'health': 386,
'rich': 385,
'enjoy': 385,
'writing': 385,
'red': 385,
'ready': 383,
'inform': 382,
'march': 381,
'reason': 381,
'degrees': 381,
'bmpinterrogdelay': 379,
'david': 376,
'remove': 376,
'cgdc': 376,
'jp': 376,
'frank': 375,
'comments': 375,
'ii': 374,
'bulk': 374,
```

```
'licensed': 372,
'enhancing': 371,
'values': 371,
'item': 370,
'nice': 370,
'en': 370,
'penis': 369,
'initial': 369,
'minutes': 369,
'sound': 369,
'addresses': 368,
'letter': 368,
'included': 368,
'tests': 368,
'fontp': 368,
'feb': 366,
'chips': 366,
'jun': 366,
'asked': 365,
'hot': 365,
'max': 365,
'outputs': 365,
'confirm': 364,
'li': 364,
'pounds': 364,
'ground': 363,
'purchase': 363,
'species': 362,
'tomorrow': 361,
'techniques': 361,
'successful': 361,
'worked': 361,
'url': 361,
'routines': 361,
'switch': 360,
'headquartered': 360,
'house': 360,
'professor': 359,
'personal': 358,
'designed': 357,
'winning': 357,
'inside': 357,
'eye': 356,
'worldwide': 355,
'mobile': 355,
'funds': 355,
'climb': 353,
'security': 353,
'break': 353,
'connection': 352,
'lottery': 352,
'roman': 351,
'linux': 351,
'foreign': 350,
'bbb': 350,
'undertaken': 350,
```

```
'advanced': 349,
'radio': 349,
'completed': 348,
'amount': 348,
'returns': 348,
'void': 348,
'base': 348,
'exchange': 348,
'specific': 348,
'geophysical': 347,
'chance': 347,
'precedence': 347,
'care': 345,
'extra': 345,
'multipart': 344,
'directory': 344,
'oct': 344,
'chinas': 343,
'spi': 343,
'generate': 342,
'resource': 341,
'method': 340,
'events': 340,
'resistor': 340,
'classzfonttd': 340,
'profile': 339,
'brfont': 338,
'ctxe': 338,
'moving': 338,
'registration': 338,
'engine': 338,
'reported': 337,
'electronics': 337,
'model': 336,
'helps': 336,
'built': 336,
'phase': 336,
'java': 336,
'youve': 335,
'statement': 335,
'ma': 335,
'las': 334,
'developments': 334,
'markets': 334,
'visa': 334,
'technical': 333,
'secure': 333,
'managed': 333,
'appreciate': 333,
'bytes': 333,
'polaroid': 333,
'aggressive': 332,
'realize': 332,
'terms': 332,
'material': 331,
'ram': 331,
```

```
'assembly': 331,
'confidentiality': 330,
'largest': 330,
'mit': 330,
'todays': 330,
'face': 329,
'wide': 329,
'respect': 329,
'worldclass': 328,
'guess': 328,
'groups': 327,
'draw': 326,
'reset': 326,
'pulse': 326,
'takes': 325,
'fontbtd': 325,
'legal': 325,
'category': 325,
'article': 325,
'vegas': 324,
'finally': 324,
'avoid': 323,
'voice': 323,
'meaning': 323,
'setting': 322,
'bilbo': 321,
'functions': 321,
'verde': 321,
'divfont': 321,
'operating': 320,
'black': 320,
'bbhttplovematchbzpc': 320,
'bbblmjb': 320,
'stepper': 320,
'herea': 319,
'andor': 318,
'gapj': 318,
'ph': 317,
'pick': 317,
'robotics': 317,
'httpequivcontentlanguage': 316,
'claim': 316,
'reserves': 316,
'wanted': 315,
'materials': 315,
'le': 315,
'distance': 315,
'batteries': 315,
'bzb': 315,
'publiclistsapplecom': 315,
'respond': 314,
'hereabrbr': 313,
'mind': 313,
'advantage': 313,
'study': 313,
'canadian': 313,
```

```
'analysis': 313,
'sort': 312,
'tremendous': 312,
'dated': 312,
'styledmsospacerun': 312,
'michael': 311,
'fall': 311,
'providence': 311,
'community': 311,
'rat': 311,
'win': 310,
'months': 310,
'step': 310,
'clear': 310,
'tabs': 310,
'marine': 310,
'english': 309,
'opt': 309,
'hand': 309,
'environment': 309,
'rest': 309,
'fixed': 309,
'sexual': 309,
'sciences': 309,
'follow': 308,
'lost': 308,
'sperm': 308,
'external': 308,
'stages': 307,
'heard': 307,
'tv': 307,
'wait': 306,
'attention': 306,
'clock': 306,
'mention': 305,
'notice': 304,
'dynamic': 304,
'hightech': 304,
'includes': 304,
'colspan': 304,
'contentenus': 303,
'relief': 303,
'food': 303,
'customers': 303,
'formatflowed': 302,
'structure': 302,
'worry': 302,
'feminist': 302,
'ps': 301,
'financial': 301,
'direct': 301,
'wouldnt': 300,
'figure': 300,
'final': 300,
'routine': 300,
'gpd': 300,
```

```
 'region': 299,
 'view': 299,
 'appear': 299,
 'bill': 299,
 'lower': 299,
 'downloaded': 299,
 'biology': 298,
 'session': 298,
 'social': 298,
 'cut': 298,
 'inputs': 298,
 'paddingin': 298,
 'provided': 297,
 'thursday': 297,
 'facearialverdana': 297,
 'targetblankmore': 297,
 'tablebrbr': 297,
 'websitebr': 297,
 'targetblanksee': 297,
 'prize': 297,
 'staff': 297,
 'estate': 296,
 'car': 296,
 'completion': 296,
 'monthly': 296,
 'ptheightin': 296,
 'interrupt': 295,
 'night': 295,
 'wall': 295,
 'electronic': 295,
 'difference': 294,
 'review': 294,
 'watsuncccolumbiaedu': 294,
 'annual': 293,
 'cheers': 293,
 'wire': 293,
 'wells': 292,
 'da': 292,
 'projects': 292,
 'australia': 291,
 'solid': 291,
 'dollars': 291,
 'references': 291,
 'separator': 291,
 'accelerate': 290,
 'happened': 290,
 'vm': 290,
 'diet': 289,
 ...}
```

Creating the feature matrices

In [17]:
```
# Feature matrix for the spam training set
feature_matrix_spam = np.zeros((len(training_spam_df), 10000))
list_most_common_words = list(most_common_words.keys())
```

```
for index in range(len(training_spam_df)):
    for word in str(training_spam_df.iloc[index]['email_message']).split():
        if word in most_common_words:
            feature_matrix_spam[index][list_most_common_words.index(word)] = 1

feature_matrix_spam
```

Out[17]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [18]:
```
# Feature matrix for the ham training set
feature_matrix_ham = np.zeros((len(training_ham_df), 10000))

for index in range(len(training_ham_df)):
    for word in str(training_ham_df.iloc[index]['email_message']).split():
        if word in most_common_words:
            feature_matrix_ham[index][list_most_common_words.index(word)] = 1

feature_matrix_ham
```

Out[18]:
```
array([[0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Computing the Priors

In [20]:
```
# Compute the probabilities for spam and ham

# Number of ham emails in the training set
n_ham = len(training_ham_df)
# Number of spam emails in the training set
n_spam = len(training_spam_df)
# Total number of emails
n_doc = len(training_df)

# Formulas
p_ham = n_ham / n_doc
p_spam = n_spam / n_doc

# Print the probabilities of ham and spam
print(f"P(c = ham) = {p_ham}")
print(f"P(c = spam) = {p_spam}")
```

```
P(c = ham) = 0.3531924882629108
P(c = spam) = 0.6468075117370892
```

Computing the Likelihood of each word

In [22]:
```
def laplace_smoothing(feature_matrix_spam, feature_matrix_ham, smoothing_param = 1.
    # Initialize the probability of each word given spam and ham
    word_probabilities_spam = np.zeros(len(most_common_words))
    word_probabilities_ham = np.zeros(len(most_common_words))

    # Count occurrences of words in spam and ham emails
    word_count_spam = np.sum(feature_matrix_spam, axis=0)
    word_count_ham = np.sum(feature_matrix_ham, axis=0)

    # Total number of words in spam and ham
    total_words_spam = np.sum(word_count_spam)
    total_words_ham = np.sum(word_count_ham)

    # Initialize the number of classes
    num_classes = 2

    # Calculate the likelihood of each word with Laplace smoothing
    for i in range(len(most_common_words)):
        word_probabilities_spam[i] = (word_count_spam[i] + smoothing_param) / (tota
        word_probabilities_ham[i] = (word_count_ham[i] + smoothing_param) / (total_

    return word_probabilities_spam, word_probabilities_ham

# Calculate the likelihood of words given spam and ham using Laplace smoothing
likelihood_spam, likelihood_ham = laplace_smoothing(feature_matrix_spam, feature_ma

# Print the likelihood of each word given spam and ham after applying Laplace smoot
print("Likelihood of each word given spam (after applying Laplace smoothing):")
print(likelihood_spam)
print("Likelihood of each word given ham (after applying Laplace smoothing):")
print(likelihood_ham)
```

```
Likelihood of each word given spam (after applying Laplace smoothing):
[2.24136692e-03 1.55638004e-03 4.94313486e-03 ... 4.71630314e-05
 4.71630314e-05 4.71630314e-05]
Likelihood of each word given ham (after applying Laplace smoothing):
[4.97431320e-05 3.86891027e-05 6.36988440e-03 ... 2.76350733e-06
 2.76350733e-06 2.76350733e-06]
```

Classifying the emails

In [24]:
```
# Function for classifying emails as ham (0) or spam (1)
def classify_email(email, likelihood_ham, likelihood_spam, p_ham, p_spam):
    # Initialize the log probabilities for ham and spam
    log_probability_ham = 0
    log_probability_spam = 0

    # Split the email into words
    words = str(email).split()

    # Compute the log probabilities for each word in the email
    for word in words:
        if word in most_common_words:
            log_probability_ham += np.log(likelihood_ham[list_most_common_words.ind
            log_probability_spam += np.log(likelihood_spam[list_most_common_words.i
```

```
    # Add the log probabilities of the prior probabilities for ham and spam
    log_probability_ham += np.log(p_ham)
    log_probability_spam += np.log(p_spam)

    # Return 0 if the email is classified as ham; otherwise, return 1 for spam
    return 0 if log_probability_ham > log_probability_spam else 1
```

In [25]:
```
# Classify the training emails and store the results in a new column 'predicted'
training_df_predicted = training_df.copy()
training_df_predicted.loc[:, 'predicted'] = training_df_predicted['email_message'].
    lambda email: classify_email(email, likelihood_ham, likelihood_spam, p_ham, p_s
)
training_df_predicted
```

Out[25]:

	folder	file	email_message	classification	predicted
0	0	0	mailing list queried weeks ago running set arc...	0	0
1	0	1	luxury watches buy rolex rolex cartier bvlgari...	1	1
2	0	2	academic qualifications prestigious nonacc red...	1	1
3	0	3	greetings verify subscription planfans list ch...	0	0
4	0	4	chauncey conferred luscious continued tonsillitis	1	0
...	...	...	...	...	...
21295	70	295	btijclnab binpqnejgmb httpgethighbizez bldb xi...	1	1
21296	70	296	special offer adobe video collection adobe pre...	1	1
21297	70	297	doctype html public wcdtd html transitionalen ...	1	1
21298	70	298	mounted isu infrared demodulator hb realised r...	0	0
21299	70	299	httptmqmctoverpacenet suffering pain depressio...	1	1

21300 rows × 5 columns

In [26]:
```
# Count the number of correctly classified emails
correct_count = 0

# Iterate over each email in the training DataFrame
for index, row in training_df_predicted.iterrows():
    # Check if the predicted classification matches the actual classification
    if float(row['classification']) == float(row['predicted']):
        correct_count += 1
```

```
# Total number of emails
total_emails = len(training_df_predicted)

# Print the results
print(
    f"Out of {total_emails} emails, {correct_count} were classified correctly. "
    f"This represents an accuracy of {correct_count / total_emails * 100:.2f}%."
)
```

```
Out of 21300 emails, 20324 were classified correctly. This represents an accuracy of
95.42%.
```

Testing the Classifier

In [28]:
```
# Classify the testing emails and store the results in a new column 'predicted'
testing_df_predicted = testing_df.copy()
testing_df_predicted.loc[:, 'predicted'] = testing_df_predicted['email_message'].ap
    lambda email: classify_email(email, likelihood_ham, likelihood_spam, p_ham, p_s
).values
testing_df_predicted
```

Out[28]:

	folder	file	email_message	classification	predicted
21300	71	0	hesitantly derive perverse satisfaction clodho...	1	1
21301	71	1	things perform experiment display will remain ...	0	0
21302	71	2	best offer month viggra ci ialis vaiium xa naa...	1	1
21303	71	3	de ar wne cr doesnt matter ow real st mmed ia ...	1	1
21304	71	4	special offer adobe video collection adobe pre...	1	1
...	...	...	...	...	...
37817	126	17	great news expec ted infinex ventures infx pri...	1	1
37818	126	18	oil sector going crazy weekly gift kkpt thing ...	1	1
37819	126	19	httpvdtobjdocscaninfo suffering pain depressio...	1	1
37820	126	20	prosperous future increased money earning powe...	1	1
37821	126	21	moat coverall cytochemistry planeload salk	1	1

16522 rows × 5 columns

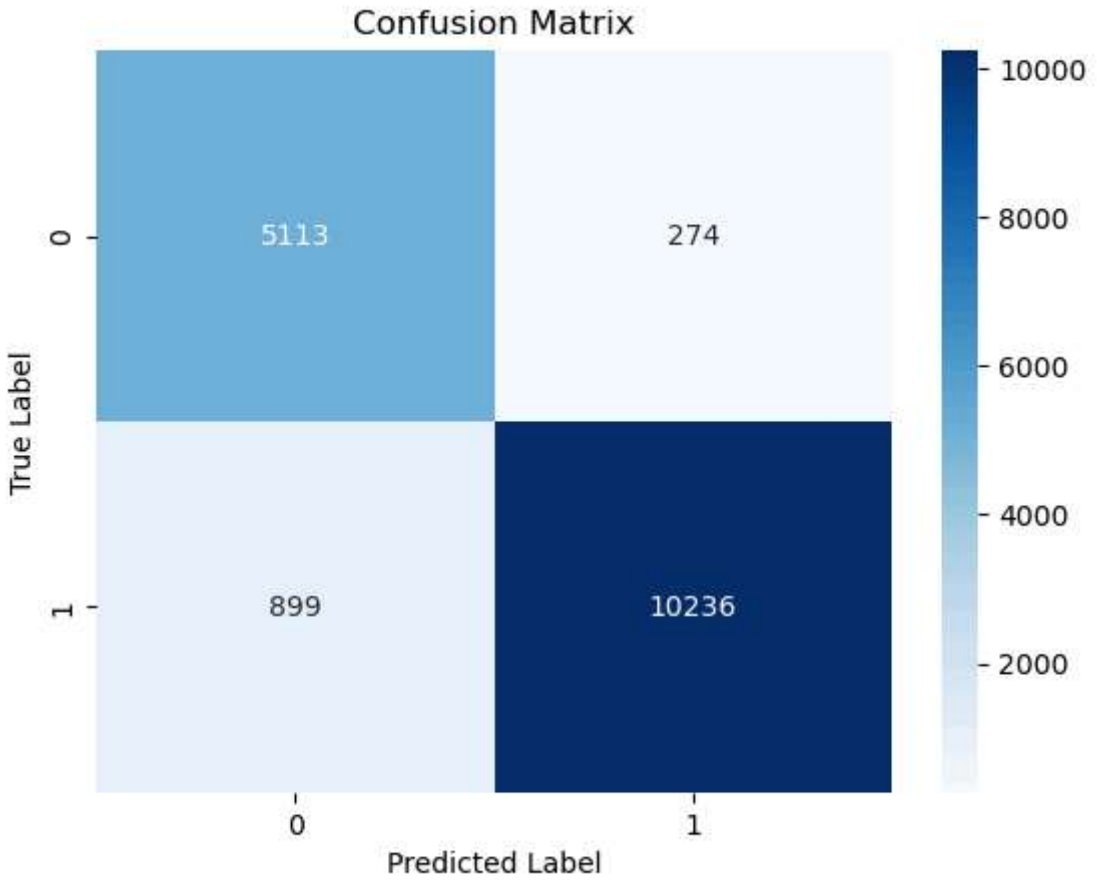
In [29]:
```
# Count the number of correctly classified emails
correct_count = 0

# Iterate over each email in the training DataFrame
```

```
for index, row in testing_df_predicted.iterrows():
    # Check if the predicted classification matches the actual classification
    if float(row['classification']) == float(row['predicted']):
        correct_count += 1

# Total number of emails
total_emails = len(testing_df_predicted)

# Print the results
print(
    f"Out of {total_emails} emails, {correct_count} were classified correctly. "
    f"This represents an accuracy of {correct_count / total_emails * 100:.2f}%."
)
```

```
Out of 16522 emails, 15349 were classified correctly. This represents an accuracy of
92.90%.
```

Performance Evaluation

In [31]:
```
# Extract the actual and predicted classifications
true_labels = np.array(testing_df_predicted['classification'])
predicted_labels = np.array(testing_df_predicted['predicted'])

# Generate the confusion matrix using true and predicted labels
confusion_matrix = metrics.confusion_matrix(true_labels, predicted_labels, labels=[

# Create a heatmap to visualize the confusion matrix
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap="Blues")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Extract values from the confusion matrix
false_positives = confusion_matrix[0][1]  # Predicted spam but is actually ham
false_negatives = confusion_matrix[1][0]  # Predicted ham but is actually spam
true_positives = confusion_matrix[1][1]   # Correctly predicted spam
true_negatives = confusion_matrix[0][0]   # Correctly predicted ham

# Print the results of the confusion matrix
print(f"False Positive Count (FP): {false_positives}")
print(f"False Negative Count (FN): {false_negatives}")
print(f"True Positive Count (TP): {true_positives}")
print(f"True Negative Count (TN): {true_negatives}")
```




```
False Positive Count (FP): 274
False Negative Count (FN): 899
True Positive Count (TP): 10236
True Negative Count (TN): 5113
```

In [32]:
```python
# Calculate the percentage of correctly identified spam and ham emails
accuracy = accuracy_score(true_labels, predicted_labels)
# Calculate the percentage of actual spam emails that are correctly classified as s
recall = recall_score(true_labels, predicted_labels)
# Calculate the percentage of emails predicted as spam that are actually spam
precision = precision_score(true_labels, predicted_labels)

# Print the results
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")
print(f"Precision: {precision * 100:.2f}%")
```

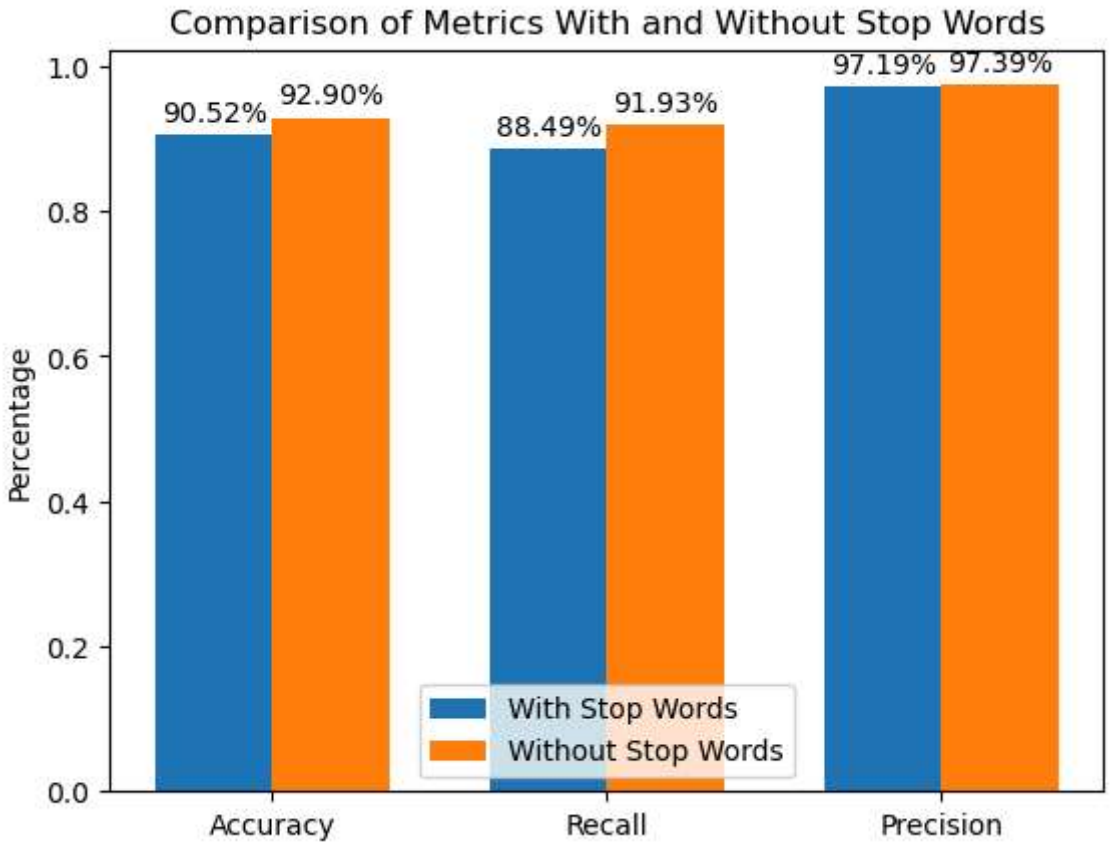
```
Accuracy: 92.90%
Recall: 91.93%
Precision: 97.39%
```

Results and Discussion

1. What is the effect of removing stop words in terms of precision, recall, and accuracy? Show a plot or a table of these results.

In [35]:

```
# Store the precision, recall, and accuracy without removing stop words
# These values are obtained by running the code above while commenting out the stop
accuracy_with_stopwords = 0.9051567606827261
recall_with_stopwords = 0.8848675348001797
precision_with_stopwords = 0.9718879463405011

# Prepare data for plotting
metrics = ['Accuracy', 'Recall', 'Precision']
with_stopwords = [accuracy_with_stopwords, recall_with_stopwords, precision_with_st
without_stopwords = [accuracy, recall, precision]

# Create the bar plot
x = np.arange(len(metrics))
width = 0.35
fig, ax = plt.subplots()
bars1 = ax.bar(x - width/2, with_stopwords, width, label='With Stop Words')
bars2 = ax.bar(x + width/2, without_stopwords, width, label='Without Stop Words')
ax.set_ylabel('Percentage')
ax.set_title('Comparison of Metrics With and Without Stop Words')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()

# Function to add value labels on top of the bars
def add_value_labels(bars):
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height * 100:.2f}%',
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom')

# Add value labels on top of the bars
add_value_labels(bars1)
add_value_labels(bars2)

plt.show()
```



Based on the bar graph above, the accuracy when stop words are not removed is 90.52%, while it is 92.90% when they are removed. Recall is 88.49% when stop words are not removed, compared to 91.93% when they are removed. Precision is 97.19% when stop words are not removed, while it is 97.39% when they are removed. This implies that removing the stop words results in improved model performance across all metrics which indicates a more effective classification of spam and ham emails.

2. Experiment on the number of words used for training. Filter the dictionary to include only words occurring more than k times (1000 words, then k > 100, and k = 50 times). For example, the word "offer" appears 150 times, that means that it will be included in the dictionary.

In [38]:
```
def filter_words(word_count, k):
    return {word: count for word, count in word_count.items() if count > k}

def create_feature_matrix(df, most_common_words):
    feature_matrix = np.zeros((len(df), len(most_common_words)))
    list_most_common_words = list(most_common_words.keys())

    for index in range(len(df)):
        for word in str(df.iloc[index]['email_message']).split():
            if word in most_common_words:
                feature_matrix[index][list_most_common_words.index(word)] = 1

    return feature_matrix
```

```
def laplace_smoothing2(feature_matrix_spam, feature_matrix_ham):
    # Initialize the probability of each word given spam and ham
    word_probabilities_spam = np.zeros(feature_matrix_spam.shape[1])
    word_probabilities_ham = np.zeros(feature_matrix_ham.shape[1])

    # Count occurrences of words in spam and ham emails
    word_count_spam = np.sum(feature_matrix_spam, axis=0)
    word_count_ham = np.sum(feature_matrix_ham, axis=0)

    # Total number of words in spam and ham
    total_words_spam = np.sum(word_count_spam)
    total_words_ham = np.sum(word_count_ham)

    # Initialize the Laplace smoothing parameter and the number of classes
    smoothing_param = 1
    num_classes = 2

    # Calculate the likelihood of each word with Laplace smoothing
    for i in range(len(word_probabilities_spam)):
        word_probabilities_spam[i] = (word_count_spam[i] + smoothing_param) / (tota
        word_probabilities_ham[i] = (word_count_ham[i] + smoothing_param) / (total_

    return word_probabilities_spam, word_probabilities_ham

def classify_email2(email, likelihood_ham, likelihood_spam, p_ham, p_spam, most_com
    log_probability_ham = 0
    log_probability_spam = 0

    words = str(email).split()

    for word in words:
        if word in most_common_words:
            log_probability_ham += np.log(likelihood_ham[list(most_common_words.key
            log_probability_spam += np.log(likelihood_spam[list(most_common_words.k

    log_probability_ham += np.log(p_ham)
    log_probability_spam += np.log(p_spam)

    return 0 if log_probability_ham > log_probability_spam else 1

k_values = [1000, 500, 100, 50]
results = {}

for k in k_values:
    filtered_words = filter_words(most_common_words, k)

    # Create feature matrices for spam and ham
    feature_matrix_spam2 = create_feature_matrix(training_spam_df, filtered_words)
    feature_matrix_ham2 = create_feature_matrix(training_ham_df, filtered_words)

    # Calculate the likelihood of words using Laplace smoothing
    likelihood_spam, likelihood_ham = laplace_smoothing2(feature_matrix_spam2, feat

    # Classify the testing emails and store the results in a new column 'predicted'
    testing_df = testing_df.copy()
```

```
    testing_df.loc[:, 'predicted'] = testing_df['email_message'].apply(
        lambda email: classify_email2(email, likelihood_ham, likelihood_spam, p_ham
    )

    # Extract the actual and predicted classifications
    true_labels = np.array(testing_df['classification'])
    predicted_labels = np.array(testing_df['predicted'])

    # Calculate accuracy, recall, and precision
    accuracy = accuracy_score(true_labels, predicted_labels)
    recall = recall_score(true_labels, predicted_labels)
    precision = precision_score(true_labels, predicted_labels)

    # Store results for the current k
    results[k] = {
        'Accuracy': accuracy * 100,
        'Recall': recall * 100,
        'Precision': precision * 100
    }

# Create a DataFrame for the results
results_df = pd.DataFrame.from_dict(results, orient='index')
results_df.index.name = 'k Value'
results_df.reset_index(inplace=True)
results_df
```

Out[38]:

	k Value	Accuracy	Recall	Precision
0	1000	90.176734	92.950157	92.509832
1	500	92.204334	93.426134	94.926544
2	100	92.773272	92.303547	96.825247
3	50	92.906428	92.213740	97.115294

Based on the values, it can be concluded that as the number of words in the dictionary is filtered to include only those occurring more than k times, a lower value of k typically results in improved metrics for the model.

3. Discuss the results of the different parameters used for Lambda smoothing. Test it on 5 varying values of the λ (e.g. λ = 2.0, 1.0, 0.5, 0.1, 0.005), Evaluate performance metrics for each.

In [41]:

```
# Function to run classification and evaluate metrics for a given λ
def evaluate_lambda(lmbda):
    likelihood_spam, likelihood_ham = laplace_smoothing(feature_matrix_spam, featur

    testing_df_copy = testing_df.copy()
    testing_df_copy.loc[:, 'predicted'] = testing_df_copy['email_message'].apply(
        lambda email: classify_email(email, likelihood_ham, likelihood_spam, p_ham,
    ).values
```

```
    true_labels = np.array(testing_df_copy['classification'])
    predicted_labels = np.array(testing_df_copy['predicted'])

    accuracy = accuracy_score(true_labels, predicted_labels)
    recall = recall_score(true_labels, predicted_labels)
    precision = precision_score(true_labels, predicted_labels)

    return accuracy, recall, precision

# List of λ values to test
lambda_values = [2.0, 1.0, 0.5, 0.1, 0.005]

# DataFrame to store the results
results_list = []

# Evaluate for each λ
for lmbda in lambda_values:
    accuracy, recall, precision = evaluate_lambda(lmbda)
    results_list.append({
        'Lambda': lmbda,
        'Accuracy': accuracy,
        'Recall': recall,
        'Precision': precision
    })

# Convert list of results to DataFrame
results_df = pd.DataFrame(results_list)

# Display the results
results_df
```

Out[41]:

	Lambda	Accuracy	Recall	Precision
0	2.000	0.928338	0.918455	0.973722
1	1.000	0.929004	0.919264	0.973930
2	0.500	0.929851	0.920341	0.974144
3	0.100	0.931727	0.922676	0.974670
4	0.005	0.931667	0.921419	0.975842

Based on the results, it can be concluded that as the value of λ decreases, the performance metrics (accuracy, recall, and precision) show a slight improvement. Although the changes are not drastic, the overall trend suggests that lower values of λ lead to better performance.

4. What are your recommendations to further improve the model?

To further improve the model, it is recommended to remove HTML tags such as `<td>`, `<tr>`, and `<br>` because they are included in the top five most common words, even though they are not particularly useful.