CS 280: Intelligent Systems (TueG-A)

Programming Assignment 1

Naive Bayes Spam Filter

Meryll Angelica J. Viernes

2010 - 42676

1. Introduction

Bayesian classification is based on Bayes' Theorem. It uses both supervised learning and statistical method for classification. Prior knowledge and observed data are used in practical learning algorithm. It uses probabilistic model which allows to capture uncertainty by determining probabilities of outcome.

Naive Bayesian text classification is commonly explored as the basis for spam filtering. Spam e-mails are identified by using Naive Bayes classifier. Since spam e-mail detection can be considered as two-category classification problem, Naive Bayesian classifier has become popular to distinguish unsolicited e-mail (called spam) and legitimate e-mail (called ham). Several modern mail clients and free software project such as Mozilla, DSPAM, SpamAssassin, SpamBayes, Bogofilter, and ASSP have implemented Bayesian spam filtering.

Naive Bayes formula is given by:

$$P(\omega|x_1, x_2, \cdots, x_d) = \frac{\prod_{i=1}^{d} P(x_i|\omega)P(\omega)}{\sum_\omega \prod_{i=1}^{d} P(x_i|\omega)P(\omega)}$$

where d is the number of unique words in the vocabulary $|V|$. This tells the probability that a mail is a spam given the words in it, written $P(\omega|x_1, x_2, \cdots, x_d)$, when we know the prior probability of words in a spam mail, written as $P(x_i|\omega)$, and the prior probabilities of spam and words. For this spam filter, the class conditional likelihood for word x_i is as follows:

$$P(x_i|\omega) = \frac{\sum_{D \in D_\omega} I(x_i \in D)}{|D_\omega|}$$

where $I(x_i \in D)$ is the indicator function, D_ω is the set of documents belonging to class in the training set and $|D_\omega|$ is the size of this set. A word that does not appear in a document class in the training data would have class conditional likelihood equal to zero. However that does not mean that that word would never appear in any document of that class in the test set. If it does, it loses all other prior probabilities and directly classify it to the other document class. To not lose count information in classes where such information is available, formula is modified by giving maximum likelihood estimate using Lambda smoothing to avoid zero probabilities.

$$P(x_i|\omega) = \frac{\left[\sum_{D \in D_\omega} I(x_i \in D)\right] + \lambda}{|D_\omega| + \lambda|V|}$$

where $|V|$ is the size of the vocabulary and lambda is some random number. The effect of lambda in the efficiency of the classifier was checked through precision and recall which are computed as follows:

$$Precision = \frac{TP}{TP + FP}$$

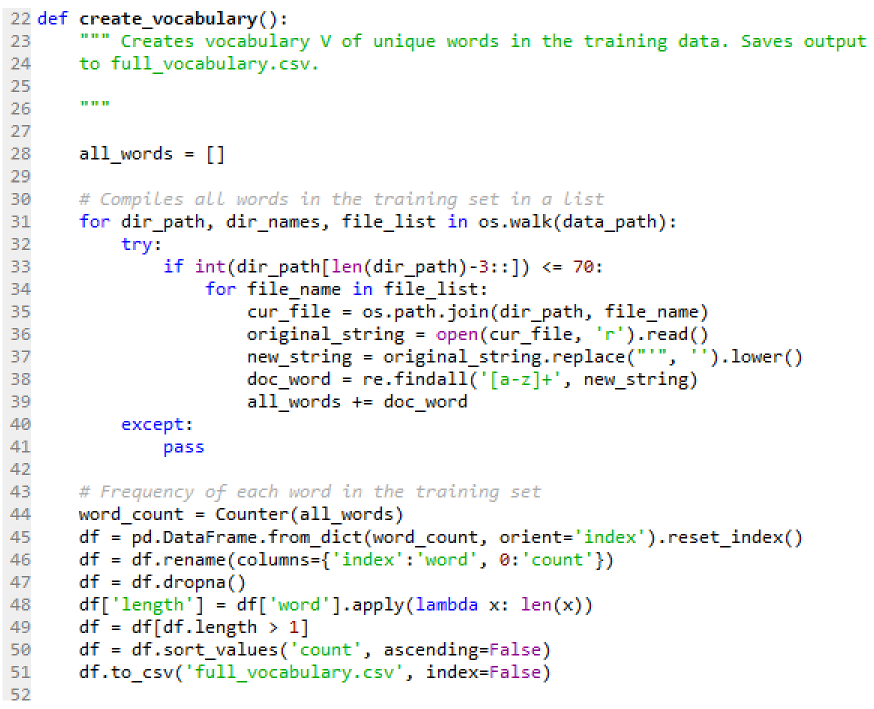
$$Recall = \frac{TP}{TP + FN}$$

where TP (true positive) is the number of spam messages classified as spam, TN (true negative) is the number of ham messages classified as ham, FP (false positive) is the number of ham messages misclassified as spam, and FN (false negative) is the number of spam messages misclassified as ham.

Another way to improve the efficiency of this classifier is to apply attribute selection process, retaining only a subset of the words. This would both speedup learning and classification process and lower memory requirement. Furthermore, the risk of overfitting the training data is reduced which leads to increase in classification performance. Possible approach is to remove infrequent and commonly frequent words. Predictions should not be based on rare observation thus infrequent words are removed. Several words such as "a, of, the" that are common for both classes are less informative but tend to dominate the classifier. Since it is likely to occur in ham as it is in spam, it does not contribute in correctly classifying messages.

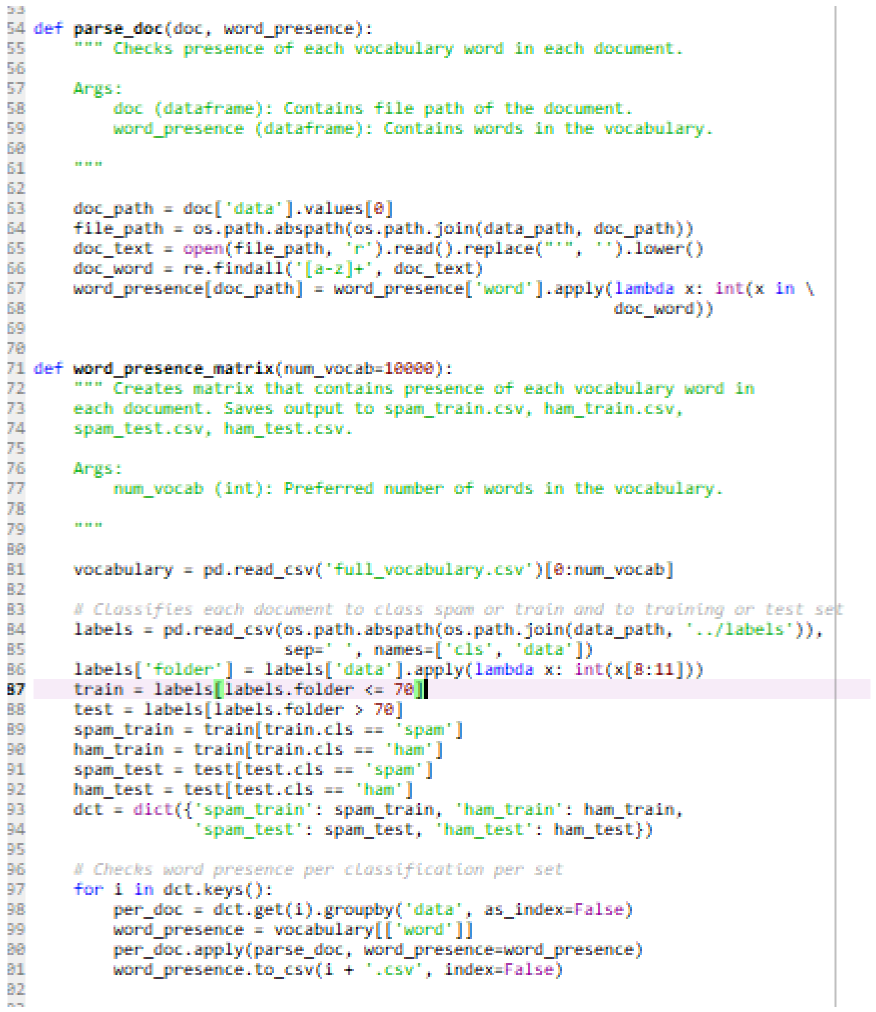
2. Methodology

Vocabulary was formed by choosing unique words in the training data. A word was defined as any sequence of alphabetic characters delimited by white space and/or punctuation. Words in all documents under training data were combined and the frequency of each word were counted. Top 10,000 words were selected; removing infrequent words in the process. Output was saved to `full_vocabulary.csv`.



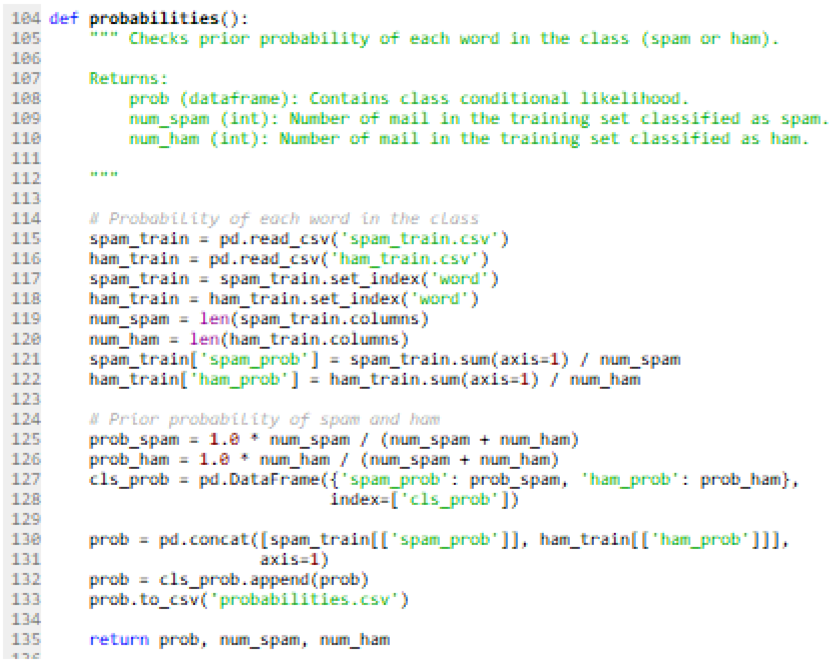
```python
def create_vocabulary():
    """ Creates vocabulary V of unique words in the training data. Saves output
    to full_vocabulary.csv.

    """

    all_words = []

    # Compiles all words in the training set in a list
    for dir_path, dir_names, file_list in os.walk(data_path):
        try:
            if int(dir_path[len(dir_path)-3::]) <= 70:
                for file_name in file_list:
                    cur_file = os.path.join(dir_path, file_name)
                    original_string = open(cur_file, 'r').read()
                    new_string = original_string.replace("'", '').lower()
                    doc_word = re.findall('[a-z]+', new_string)
                    all_words += doc_word
        except:
            pass

    # Frequency of each word in the training set
    word_count = Counter(all_words)
    df = pd.DataFrame.from_dict(word_count, orient='index').reset_index()
    df = df.rename(columns={'index':'word', 0:'count'})
    df = df.dropna()
    df['length'] = df['word'].apply(lambda x: len(x))
    df = df[df.length > 1]
    df = df.sort_values('count', ascending=False)
    df.to_csv('full_vocabulary.csv', index=False)

```

The presence of each word in the vocabulary was then checked in each document in the training data and test data creating word presence matrix with words in the vocabulary as the row index and relative file path to document in the training data and test data. Each cell is assigned 1 if the word is present in the document; else, cell is assigned 0. Word presence matrix was segregated per class (ham or spam) per data set type (training data or test data). Outputs were saved to `spam_train.csv`, `ham_train.csv`, `spam_test.csv`, `ham_test.csv`.



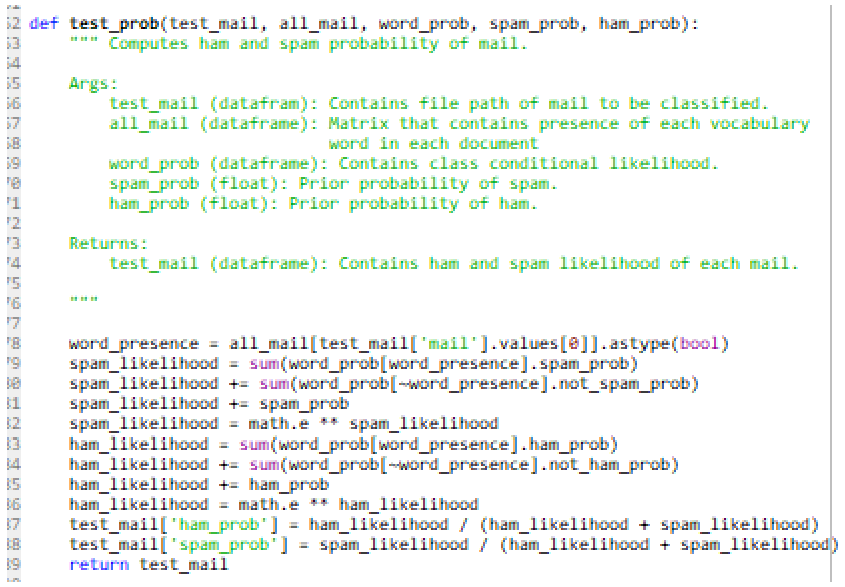
```
def parse_doc(doc, word_presence):
    """ Checks presence of each vocabulary word in each document.

    Args:
        doc (dataframe): Contains file path of the document.
        word_presence (dataframe): Contains words in the vocabulary.

    """

    doc_path = doc['data'].values[0]
    file_path = os.path.abspath(os.path.join(data_path, doc_path))
    doc_text = open(file_path, 'r').read().replace("'", '').lower()
    doc_word = re.findall('[a-z]+', doc_text)
    word_presence[doc_path] = word_presence['word'].apply(lambda x: int(x in \
                                                           doc_word))


def word_presence_matrix(num_vocab=10000):
    """ Creates matrix that contains presence of each vocabulary word in
    each document. Saves output to spam_train.csv, ham_train.csv,
    spam_test.csv, ham_test.csv.

    Args:
        num_vocab (int): Preferred number of words in the vocabulary.

    """

    vocabulary = pd.read_csv('full_vocabulary.csv')[0:num_vocab]

    # Classifies each document to class spam or train and to training or test set
    labels = pd.read_csv(os.path.abspath(os.path.join(data_path, '../labels')),
                         sep=' ', names=['cls', 'data'])
    labels['folder'] = labels['data'].apply(lambda x: int(x[8:11]))
    train = labels[labels.folder <= 70]
    test = labels[labels.folder > 70]
    spam_train = train[train.cls == 'spam']
    ham_train = train[train.cls == 'ham']
    spam_test = test[test.cls == 'spam']
    ham_test = test[test.cls == 'ham']
    dct = dict({'spam_train': spam_train, 'ham_train': ham_train,
                'spam_test': spam_test, 'ham_test': ham_test})

    # Checks word presence per classification per set
    for i in dct.keys():
        per_doc = dct.get(i).groupby('data', as_index=False)
        word_presence = vocabulary[['word']]
        per_doc.apply(parse_doc, word_presence=word_presence)
        word_presence.to_csv(i + '.csv', index=False)
```

Class conditional likelihood or probability of each word (`spam_prob, ham_prob`) found in each class in the training was computed by summing the values per row and dividing it to total number of documents (length of matrix column excluding the index) belonging in that class. Prior probabilities of spam and ham are computed by dividing the length of respective matrix column over the sum of lengths of the matrix column. Output was saved to `probabilities.csv`. Lambda smoothing was applied. Probability that a word is not found in spam documents (`not_spam_prob`) and probability that a word is not found in ham documents (`not_ham_prob`) were computed by getting the difference of spam likelihood and ham likelihood from 1, respectively. Probabilities were converted to logarithmic form to avoid loss of precision.

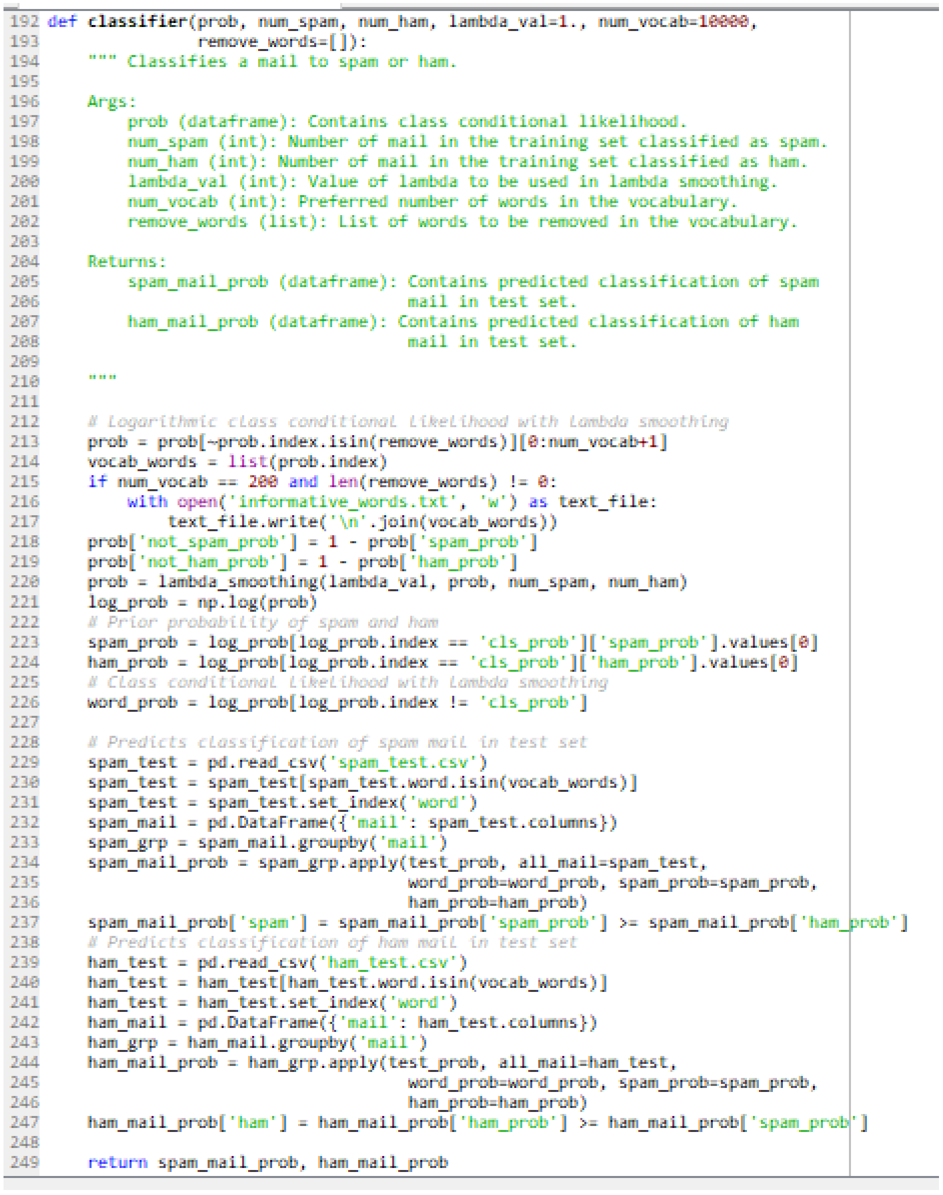


```
def probabilities():
    """ Checks prior probability of each word in the class (spam or ham).

    Returns:
        prob (dataframe): Contains class conditional likelihood.
        num_spam (int): Number of mail in the training set classified as spam.
        num_ham (int): Number of mail in the training set classified as ham.

    """

    # Probability of each word in the class
    spam_train = pd.read_csv('spam_train.csv')
    ham_train = pd.read_csv('ham_train.csv')
    spam_train = spam_train.set_index('word')
    ham_train = ham_train.set_index('word')
    num_spam = len(spam_train.columns)
    num_ham = len(ham_train.columns)
    spam_train['spam_prob'] = spam_train.sum(axis=1) / num_spam
    ham_train['ham_prob'] = ham_train.sum(axis=1) / num_ham

    # Prior probability of spam and ham
    prob_spam = 1.0 * num_spam / (num_spam + num_ham)
    prob_ham = 1.0 * num_ham / (num_spam + num_ham)
    cls_prob = pd.DataFrame({'spam_prob': prob_spam, 'ham_prob': prob_ham},
                            index=['cls_prob'])

    prob = pd.concat([spam_train[['spam_prob']], ham_train[['ham_prob']]],
                     axis=1)
    prob = cls_prob.append(prob)
    prob.to_csv('probabilities.csv')

    return prob, num_spam, num_ham
```

In classifying whether a document is spam or ham, Naïve Bayes classifier was used in computing the probability that a document is a spam. In classifying a spam test data, $P(x_i|\omega)$ of the words that are present in the test data are obtained in `spam_prob` column of the probabilities dataframe; while $P(x_i|\omega)$ of the words that are not in the test data are obtained in `not_spam_prob` column of the probabilities dataframe. Spam likelihood is computed by exponentiating the sum of these $P(x_i|\omega)$ and prior spam probability. Similarly, in classifying a ham test data, $P(x_i|\omega)$ of the words that are present in the test data are obtained in `ham_prob` column of the probabilities dataframe; while $P(x_i|\omega)$ of the words that are not in the test data are obtained in `not_ham_prob` column of the probabilities dataframe. Ham likelihood is computed by exponentiating the sum of these $P(x_i|\omega)$ and prior ham probability. Probability that a document is spam is $\frac{ham\ likelihood}{ham\ likelihood+spam\ likelihood}$ while the probability that a document is a ham is $\frac{spam\ likelihood}{ham\ likelihood+spam\ likelihood}$. Document is a classified to which has higher probability.

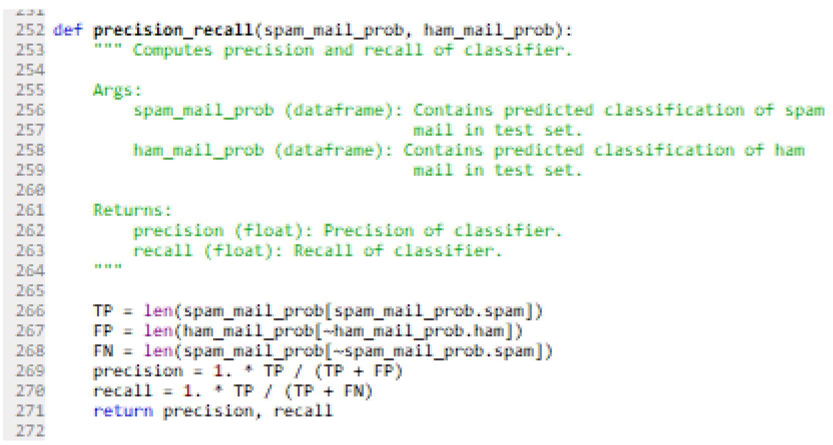


```
def test_prob(test_mail, all_mail, word_prob, spam_prob, ham_prob):
    """ Computes ham and spam probability of mail.

    Args:
        test_mail (datafram): Contains file path of mail to be classified.
        all_mail (dataframe): Matrix that contains presence of each vocabulary
                              word in each document
        word_prob (dataframe): Contains class conditional likelihood.
        spam_prob (float): Prior probability of spam.
        ham_prob (float): Prior probability of ham.

    Returns:
        test_mail (dataframe): Contains ham and spam likelihood of each mail.

    """

    word_presence = all_mail[test_mail['mail'].values[0]].astype(bool)
    spam_likelihood = sum(word_prob[word_presence].spam_prob)
    spam_likelihood += sum(word_prob[~word_presence].not_spam_prob)
    spam_likelihood += spam_prob
    spam_likelihood = math.e ** spam_likelihood
    ham_likelihood = sum(word_prob[word_presence].ham_prob)
    ham_likelihood += sum(word_prob[~word_presence].not_ham_prob)
    ham_likelihood += ham_prob
    ham_likelihood = math.e ** ham_likelihood
    test_mail['ham_prob'] = ham_likelihood / (ham_likelihood + spam_likelihood)
    test_mail['spam_prob'] = spam_likelihood / (ham_likelihood + spam_likelihood)
    return test_mail
```



```python
def classifier(prob, num_spam, num_ham, lambda_val=1., num_vocab=10000,
               remove_words=[]):
    """ Classifies a mail to spam or ham.

    Args:
        prob (dataframe): Contains class conditional likelihood.
        num_spam (int): Number of mail in the training set classified as spam.
        num_ham (int): Number of mail in the training set classified as ham.
        lambda_val (int): Value of lambda to be used in lambda smoothing.
        num_vocab (int): Preferred number of words in the vocabulary.
        remove_words (list): List of words to be removed in the vocabulary.

    Returns:
        spam_mail_prob (dataframe): Contains predicted classification of spam
                                    mail in test set.
        ham_mail_prob (dataframe): Contains predicted classification of ham
                                   mail in test set.

    """

    # Logarithmic class conditional likelihood with lambda smoothing
    prob = prob[~prob.index.isin(remove_words)][0:num_vocab+1]
    vocab_words = list(prob.index)
    if num_vocab == 200 and len(remove_words) != 0:
        with open('informative_words.txt', 'w') as text_file:
            text_file.write('\n'.join(vocab_words))
    prob['not_spam_prob'] = 1 - prob['spam_prob']
    prob['not_ham_prob'] = 1 - prob['ham_prob']
    prob = lambda_smoothing(lambda_val, prob, num_spam, num_ham)
    log_prob = np.log(prob)
    # Prior probability of spam and ham
    spam_prob = log_prob[log_prob.index == 'cls_prob']['spam_prob'].values[0]
    ham_prob = log_prob[log_prob.index == 'cls_prob']['ham_prob'].values[0]
    # Class conditional likelihood with lambda smoothing
    word_prob = log_prob[log_prob.index != 'cls_prob']

    # Predicts classification of spam mail in test set
    spam_test = pd.read_csv('spam_test.csv')
    spam_test = spam_test[spam_test.word.isin(vocab_words)]
    spam_test = spam_test.set_index('word')
    spam_mail = pd.DataFrame({'mail': spam_test.columns})
    spam_grp = spam_mail.groupby('mail')
    spam_mail_prob = spam_grp.apply(test_prob, all_mail=spam_test,
                                    word_prob=word_prob, spam_prob=spam_prob,
                                    ham_prob=ham_prob)
    spam_mail_prob['spam'] = spam_mail_prob['spam_prob'] >= spam_mail_prob['ham_prob']
    # Predicts classification of ham mail in test set
    ham_test = pd.read_csv('ham_test.csv')
    ham_test = ham_test[ham_test.word.isin(vocab_words)]
    ham_test = ham_test.set_index('word')
    ham_mail = pd.DataFrame({'mail': ham_test.columns})
    ham_grp = ham_mail.groupby('mail')
    ham_mail_prob = ham_grp.apply(test_prob, all_mail=ham_test,
                                  word_prob=word_prob, spam_prob=spam_prob,
                                  ham_prob=ham_prob)
    ham_mail_prob['ham'] = ham_mail_prob['ham_prob'] >= ham_mail_prob['spam_prob']

    return spam_mail_prob, ham_mail_prob
```

To evaluate the classifier, precision and recall was computed.



```python
def precision_recall(spam_mail_prob, ham_mail_prob):
    """ Computes precision and recall of classifier.

    Args:
        spam_mail_prob (dataframe): Contains predicted classification of spam
                                    mail in test set.
        ham_mail_prob (dataframe): Contains predicted classification of ham
                                   mail in test set.

    Returns:
        precision (float): Precision of classifier.
        recall (float): Recall of classifier.
    """

    TP = len(spam_mail_prob[spam_mail_prob.spam])
    FP = len(ham_mail_prob[~ham_mail_prob.ham])
    FN = len(spam_mail_prob[~spam_mail_prob.spam])
    precision = 1. * TP / (TP + FP)
    recall = 1. * TP / (TP + FN)
    return precision, recall
```

Effects of lambda value used in Lambda smoothing was evaluated by plotting the precision and recall with respect to lambda values. Finally, the classifier was improved by removing the commonly most frequent words and its efficiency was evaluated using precision and recall.

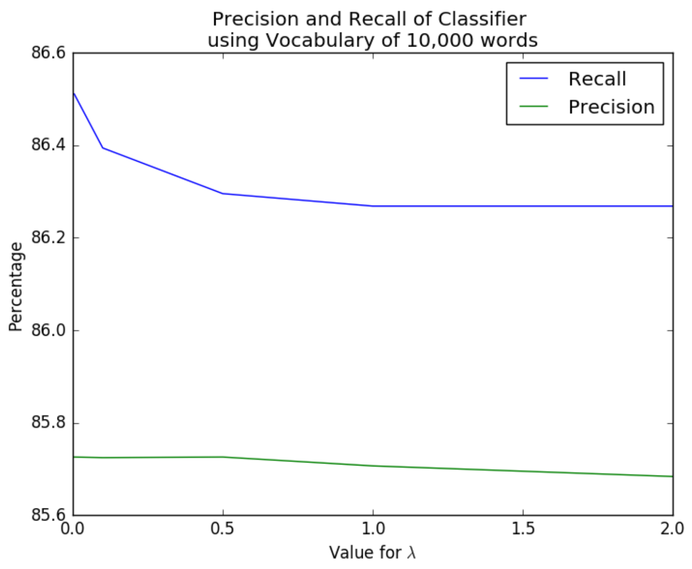
3. Results and Discussion

The following words are the frequently occurring words in the training data: `div, the, to, edu, br, font, com, from, content, and, td, mit, media, of, by, for, id, with, in, http` while the following words are the least occurring words in the training data: `tbkgty, ahoaaaamaqaakgamabaaagaaqacsaqaaraeaakwb, seeming, jwaa, polarizer, jwad, ppxtpg, fuihj, jwax, aiftuyvs, mhgxdvcxz, gttqbezgbgnqbnvgb, goryfgxwpa, iymljo, ptnbunswnbt, bjcxvbivfwe, uhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh, vxm, lautobiographie, blmbrwswausenwe`. Most of the frequently occurring words are found in the header of both spam and ham training data while the least occurring words appeared only once which is an indicator that the classifier can be improved by removing the most frequent and infrequent words.

Expectedly, the words with highest probabilities are those belonging to most frequent words. These words are less informative since they have high probability in both classes.

WORD	ham_prob	spam_prob
by	1.000000	1.000000
received	1.000000	1.000000
from	0.999734	1.000000
to	0.999601	0.998984
date	0.999468	0.866589
id	0.999468	1.000000
message	0.999335	0.805836
edu	0.996012	0.996008
with	0.994550	1.000000
subject	0.987239	1.000000

In the training data, spam has prior probability of 64.68% while ham has prior probability of 35.32%. The precision of the classifier ranges from 85.68% to 85.73% while the recall of the classifier ranges from 86.27% to 86.51%. Note that as lambda decreases, precision and recall increases. Lambda smoothing gives artificial probability to words not occurring in one class of the training data, word of probability zero appearing in certain class. As you increase lambda, the gap between the original probability zero and the smoothed probability increases. Increasing its probability, artificially increases its importance; thus, lower lambda illustrates better precision and recall.



To improve the classifier, the following commonly frequent words were removed: `the, to, edu, com, from, content, and, mit, media, by, for, id, with, in, received, is, type, handyboard, text, this, message, charset, aleve, mime, subject, date, esmtp, version, plain, mailerthe, to, edu, com, from, content, and, mit, media, by, for, id, with, in, received, is, type, handyboard, text, this, message, charset, aleve, mime, subject, date, esmtp, version, plain, mailer`. These are usually stop words or part of a header of mail.

The following are the top 200 words from the vocabulary after removing the commonly frequent words: `div, br, font, td, of, http, span, strong, size, ee, nbsp, you, px, color, style, html, class, tr, border, ed, net, it, width, that, table, on, be, face, transfer, encoding, us, href, your, at, have, are, right, bit, left, cc, ea, eb, body, as, info, www, float, or, align, ec, iso, washington, we, microsoft, smtp, not, ff, img, my, center, jp, will, our, price, padding, if, head, mail, ef, top, pt, ascii, reply, jan, but, ce, more, can, all, wed, edt, est, thu, arial, src, meta, may, mon, tue, priority, fri, nextpart, windows, height, an, ca, darial, en, title, no, product, fb, format, pine, normal, was, sun, has, mso, re, arizona, its, psu, verdana, out, mimeole, pc, one, cb, cse, equiv, new, bgcolor, feb, any, me, list, uw, mar, de, weight, sat, cac, li, adobe, boundary, psy, apr, do, psych, lists, about, so, sender, here, cf, localhost, fc, part, would, family, multipart, get, unknown, cd, there, up, none, ba, alternative, aa, ml, co, board, sp, what, bd, they, lugnet, images, quoted, printable, now, bf, white, rating, some, target, cellspacing, ad, when, like, ms, add, cs, dragon, blank, help, name, using, use, nov, he, gif, multi, margin, bb, cellpadding, only, uk`.

Using these informative only as part of the vocabulary increases the precision of the classifier from 85.73% to 89.00%. However, it decreases the recall from 86.51% to 68.99%. This is due to increase in false negative or spam messages misclassified as ham. Upon inspecting of the informative words, number of words that have higher spam probability than ham probability are significantly high than number of words that have higher ham probability than spam probability in the list of informative words. This increases the spam likelihood which in turn increases the tendency of the classifier to classify the document as spam. Similarly, precision increased due to the decrease in false positive or ham messages misclassified as spam since there are more prior knowledge in spam likelihood. Since misclassifying a legitimate message is generally much worse than misclassifying a spam, we can say that classifier improved since misclassification of ham messages decreased.

4. Conclusion and Recommendation

Attribute selection significantly affects the precision and recall of the classifier. Particularly, removing commonly frequent and infrequent words improves the classifier by decreasing false positive or ham messages misclassified as spam. Lower lambda have better precision and recall since the artificial or smoothed probability used to alternate the probability is near the original zero probability.

To further improve the classifier for spam or ham mail, the following are recommended: (i) to consider using whole email address of sender as one word since most spam mail are sent by few senders, (ii) since term frequency-inverse document frequency (TFID) was used in selecting top words to include in the vocabulary, it is recommended to use TFID in computing for the probability to reflect how important a word is in the document.

References

Johan Hovold, Naive Bayes Spam Filtering using Word Position Attributes, In *Conference on Email and Anti-Spam*, Stanford University, 2005