Assignment 2

Johnnie Oldfield

October 2019

1 Decisions

To start the assignment I made three dictionaries; ham, spam, and tokens. ham and spam will keep track of a words occurrence for each class. Tokens will just be every *unfiltered* token (or 'word') from each every email. Important note: all three dictionaries will only be using emails only from the training set. The training and testing sets will using an 80-20 split. While there is an argument to be had over the right split, 80-20 is common and considered fair.

Listing 1: Train/Test Split

```
# Get rows, shuffle, split into training and test sets
rows = df.iloc[:, 0:2].values
np.random.shuffle(rows)
rows = pd.DataFrame.from_records(rows)

# 80-20 spilt
size = int(len(rows) * .8)
training_set = rows[:size]
test_set = rows[size:]
```

I decided to shuffle the data set so that the tests are more representative of the distribution of the data. Otherwise I can only get one test for each limit I set on the important vocabulary. The next step is to add the words from the emails in the training set to the dictionaries.

Listing 2: Function to Clean Emails

```
# clean emails using regular expressions and stopwords
def clean_email(email):
  stopwords = ['a', "the", "is", "we", "are", "that", 'i',
               "on", "an", "this", "so", "do", "to", "and",
               "for", "of", "will", "or", "then", "from",
               "no", "you", "no", "so", "if", "be", "with",
               "has", "they"]
  words = re.sub("[^\w]", " ",  email).split()
  clean_email = []
  for word in words:
    if word not in stopwords:
      clean_email.append(word.lower())
  return clean_email
```

I made a function to clean the emails using regular expressions and decided that I only cared about alphanumeric words. While there were several strings with accents located in the emails (primarily spam) removing them did not affect much with the classifications; not to say that they can not be used to identify spam. The stopwords array was used to flag key words present in both ham and spam that I believed would not be useful in identifying class of email. The list can go on as there are many stop words in emails. I decided to lower case all tokens because the words themselves are more important than the case and it would be easier to count. However, uppercase words are more present in spam emails and thus could help in identifying them.

Listing 3: Defining Important Words with Occurrence

```
vocab = tokens.copy()
for word in tokens:
  if (tokens[word] <= 50):
    vocab.pop(word, None)
ham  = vocab.copy()
spam = vocab.copy()
```

I decided to limit my words based on their overall occurrences regardless if it was ham or spam. I played around with the tests which relied on the number of important words. The dictionary size would vary based on the occurrence I set. See Section 2 (Evaluation) for results.

For training I made two more dictionaries, ham_probs and spam_probs, to hold the probabilities of words in vocab being ham or spam.

Listing 4: Testing Function

```
def test(words):
  hprob = 1
  sprob = 1
  result = 0
  for word in words:
    if word in vocab:
      if ham_probs[word] != 0.0:
        hprob *= ham_probs[word]
      if spam_probs[word] != 0.0:
        sprob *= spam_probs[word]
  hprob *= ham_prob
  sprob *= spam_prob
  if hprob > sprob:
    result = 1
  else:
    result = -1
  print(hprob, sprob)
  return result
```

I made a function that follows the MAP (Maximum a posteriori) estimation to test each email in the testing set. MAP estimation intuitively made more sense to me which is why I decided to use it over the log method recommended in the assignment. Words that do not occur at all in their respective dictionary are ignored. The result is returned and stored in a result vector to be compared to the ground truth.

Listing 5: Confusion Matrix

```
truth = np.array(test_rows[0], dtype=int)
classes = len(np.unique(truth))
conmat = np.zeros((classes, classes), dtype=float)
for i in range(len(truth)):
  t = 0
  r = 0
  if truth[i] == -1:
    t = 1
  if res[i] == -1:
    r = 1
  conmat[t][r] += 1
```

2 Evaluation

The figures you see below are not the only tests I ran. For each part I ran 20+ tests to get a feel how the limit I set affected the model's performance at classifying emails as ham or spam. I chose 4 that helps represents the effects of the limits I placed.

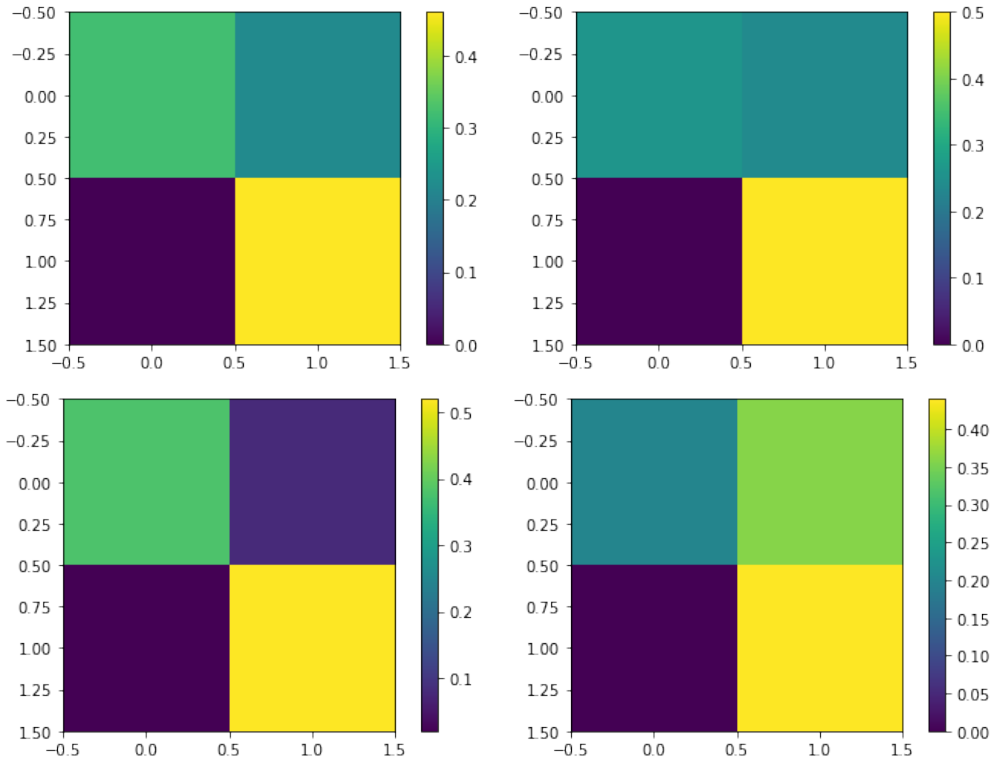

Figure 1: Around 100 words in dictionary. Occurrence 50+.

Figure 1 shows as few tests with only 100 words in the dictionary of important words (those that occur 50 or more times). Most tests average around with a 63% accuracy with some approaching 70%. On occasion the model can get lucky with it's limited dictionary and guess very well. Achieving accuracy in the 90% range.

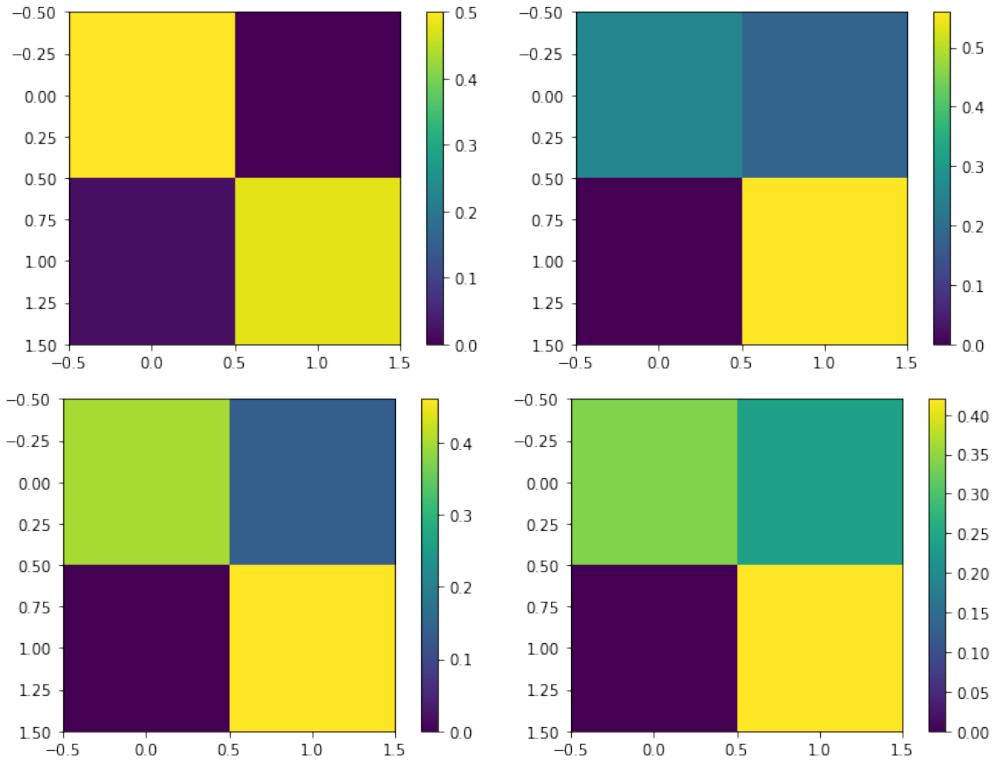

Figure 2: Around 500 words in dictionary. Occurrence 15+.

With 500 or so words in the dictionary the model improves its ability to classify emails. Errors are greatly reduced. The model consistently gets an accuracy in the 80% range and the 90% range becoming more common. Some tests fall short to the 70% range but tests below that are a rare case. I found this testing range interesting, see section What I Learned for more on that.

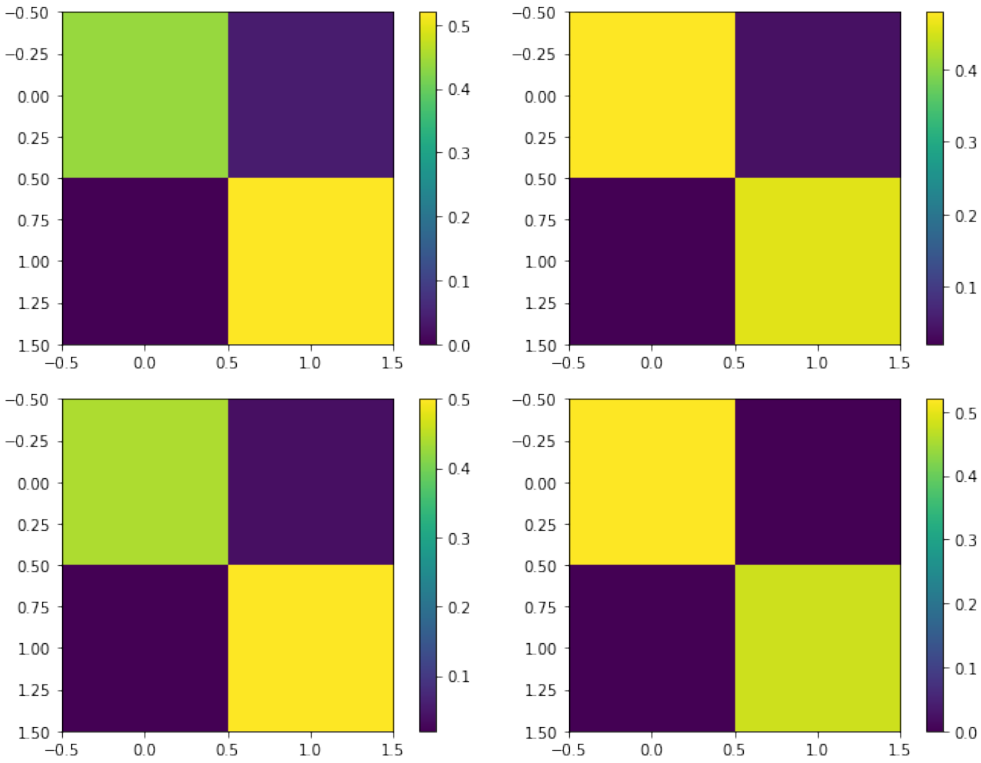

Figure 3: 3000+ words in dictionary. Occurrence 1+.

With little to no limitation (other than the stop words)the accuracy is very good. Mostly staying above 90% and sometimes classifying every email in the test set correctly. One thing I did notice during the testing that false negatives never really appear in the confusion matrix. So this model is very good a classifying spam emails. However, it can struggle to classify ham properly when the dictionary is reduced.

3 What I Learned

When looking into how training and testing should be split I learned that for large amounts of data the exact split is not very import. As long as training is getting more than test. I noticed that 80-20 was common not only for splitting data but for other things; such as population and wealth, sports, and economics [1]. Not necessarily directly related to the problem but something I found interesting.

When it came to making the confusion matrix I thought it would be a littler harder to do. In Dr.Xu's artificial intelligence class we are using the sklearn library for our assignments and labs so I did not know how to make my own confusion matrix. However, making one wasn't as bad as I thought and intuitively made sense how to create it. [2]

I was surprised by how well the model functioned with a limited dictionary. I assumed it would need at least 1000 words to preform well. But it only needed half of that to reach an accuracy in the 80% range. Which is not that bad. Although, to reach a more desirable outcome(90+%) a dictionary of over 3000 words is necessary for this model.

References

[1] `https://en.wikipedia.org/wiki/Pareto_principle`.

[2] `https://en.wikipedia.org/wiki/Confusion_matrix`.