Enron Email Top 15 Senders Prediction

Yuxin Li, Zhiqi Chen, Yuchen Liu, Zihan Li

Introduction

Dataset: Enron Email Dataset, from CALO Project

It contains 500,000 emails from about 150 users

Models we used:

SVM (Support Vector Machines)

Naive Bayes

Related Work

Which classification algorithms have the best work in the email classification area

Crawford, Kay and McCreath:

the performance difference between different classification algorithms is much smaller than the difference between different users

Matwin and Kiritchenko :

SVM is work better than Naive Bayes 10% in both highly imbalanced problems and moderately imbalanced problem. Also, have a 5% advantage prediction rate in the balanced problem

Related Work

Brutlag and Meek:

SVM is performed best, but only in the condition of a small number

TF-IDF worked best for sparse data

Segal and Kephart :

Prediction rate of TF-IDF is similar to the rate using Naıve Bayes

Related Work

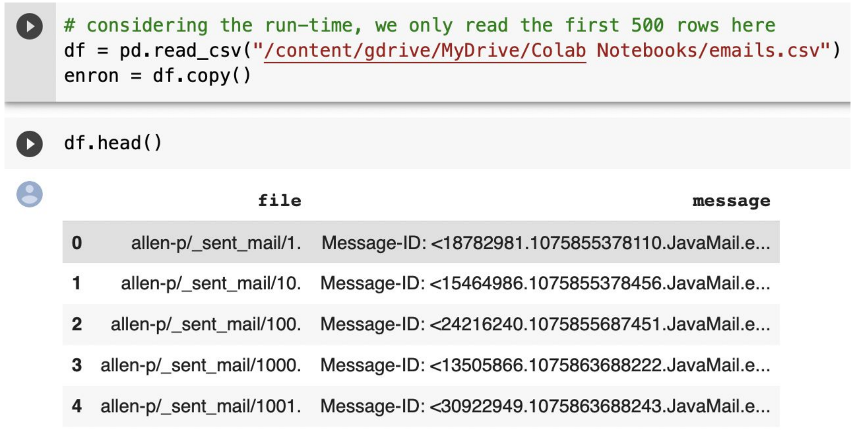
None of them proved that which of the classification algorithms are working best for a large variety of data sets

We decide to try using SVM and Naive Bayes to implement into a large variety of datasets-Enron Email Dataset as a beginner.

Data cleaning/processing & Naive Bayes classification

1. Load data
2. Top 15 senders and the #of emails they sent
3. Extract information from raw text
4. Clean the information
    a. Remove numbers and stop words in the text content
    b. Build the data frame, keep only the text content and map it with the senders
5. Separate the data into training dataset and testing dataset
6. Vectorized the data and build the model
7. Train the model using the training data, and predict the senders of the test data

1. Load data

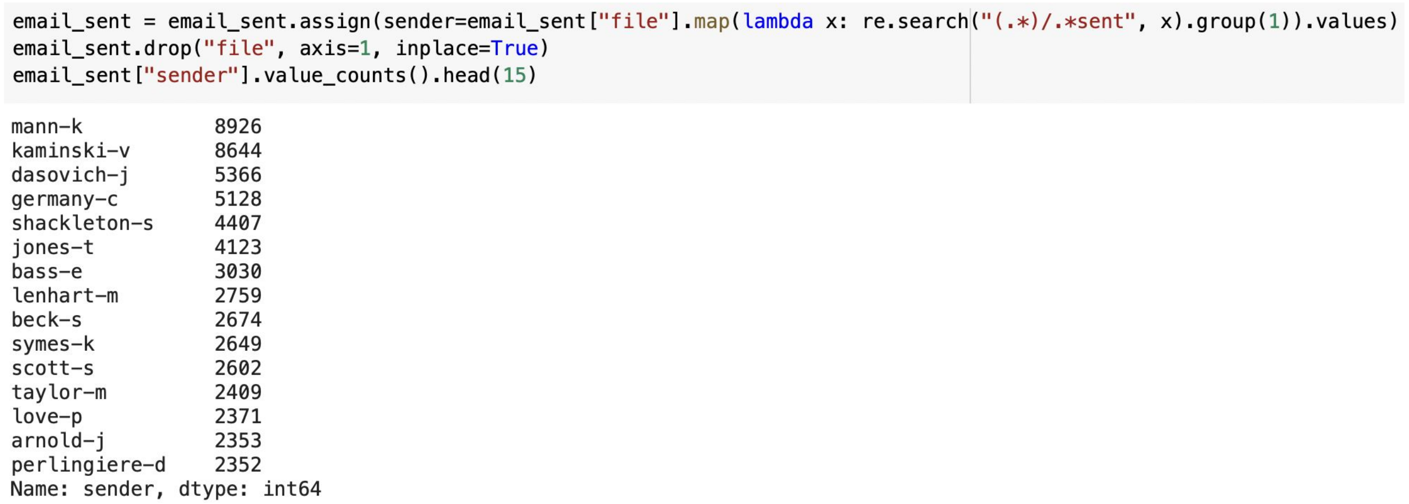


```
# considering the run-time, we only read the first 500 rows here
df = pd.read_csv("/content/gdrive/MyDrive/Colab Notebooks/emails.csv")
enron = df.copy()
```

```
df.head()
```

```
                       file                                             message
0     allen-p/_sent_mail/1.  Message-ID: <18782981.1075855378110.JavaMail.e...
1    allen-p/_sent_mail/10.  Message-ID: <15464986.1075855378456.JavaMail.e...
2   allen-p/_sent_mail/100.  Message-ID: <24216240.1075855687451.JavaMail.e...
3  allen-p/_sent_mail/1000.  Message-ID: <13505866.1075863688222.JavaMail.e...
4  allen-p/_sent_mail/1001.  Message-ID: <30922949.1075863688243.JavaMail.e...
```

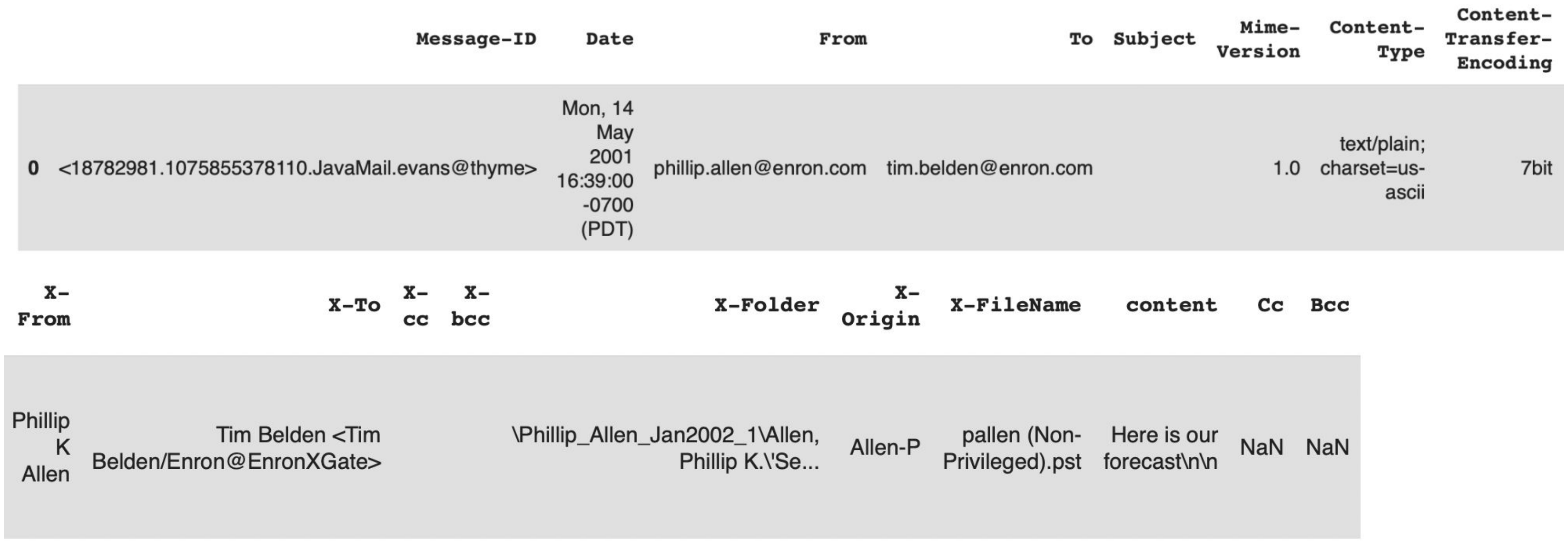
2. Top 15 senders and the #of emails they sent



```
email_sent = email_sent.assign(sender=email_sent["file"].map(lambda x: re.search("(.*)/.*sent", x).group(1)).values)
email_sent.drop("file", axis=1, inplace=True)
email_sent["sender"].value_counts().head(15)
```

```
mann-k             8926
kaminski-v         8644
dasovich-j         5366
germany-c          5128
shackleton-s       4407
jones-t            4123
bass-e             3030
lenhart-m          2759
beck-s             2674
symes-k            2649
scott-s            2602
taylor-m           2409
love-p             2371
arnold-j           2353
perlingiere-d      2352
Name: sender, dtype: int64
```

3. Extract information from raw text

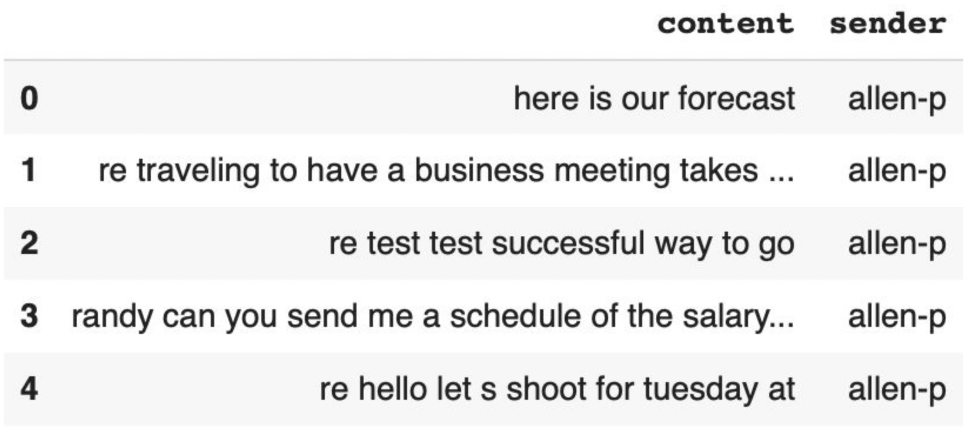


	Message-ID	Date	From	To	Subject	Mime-Version	Content-Type	Content-Transfer-Encoding
0	<18782981.1075855378110.JavaMail.evans@thyme>	Mon, 14 May 2001 16:39:00 -0700 (PDT)	phillip.allen@enron.com	tim.belden@enron.com		1.0	text/plain; charset=us-ascii	7bit

X-From	X-To	X-cc	X-bcc	X-Folder	X-Origin	X-FileName	content	Cc	Bcc
Phillip K Allen	Tim Belden <Tim Belden/Enron@EnronXGate>			\Phillip_Allen_Jan2002_1\Allen, Phillip K.\'Se...	Allen-P	pallen (Non-Privileged).pst	Here is our forecast\n\n	NaN	NaN

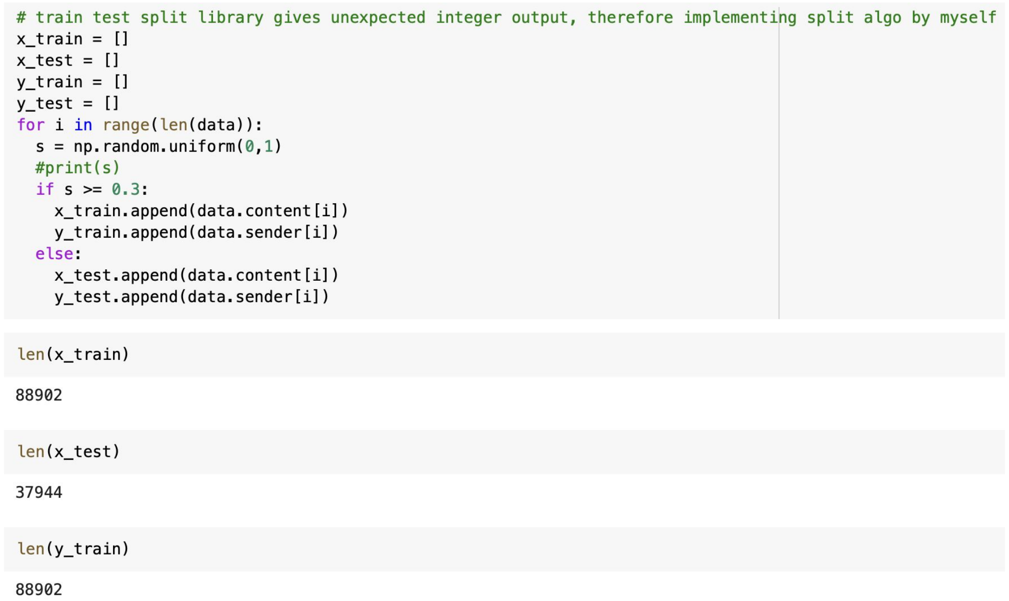
4. Clean the information

- Remove the numbers and the stop words in text content
- Build the data frame, keep only the text content and map it with the senders



	content	sender
0	here is our forecast	allen-p
1	re traveling to have a business meeting takes ...	allen-p
2	re test test successful way to go	allen-p
3	randy can you send me a schedule of the salary...	allen-p
4	re hello let s shoot for tuesday at	allen-p

5. Separate the data into training dataset and testing dataset



```
# train test split library gives unexpected integer output, therefore implementing split algo by myself
x_train = []
x_test = []
y_train = []
y_test = []
for i in range(len(data)):
  s = np.random.uniform(0,1)
  #print(s)
  if s >= 0.3:
    x_train.append(data.content[i])
    y_train.append(data.sender[i])
  else:
    x_test.append(data.content[i])
    y_test.append(data.sender[i])
```

```
len(x_train)
```

```
88902
```

```
len(x_test)
```

```
37944
```

```
len(y_train)
```

```
88902
```

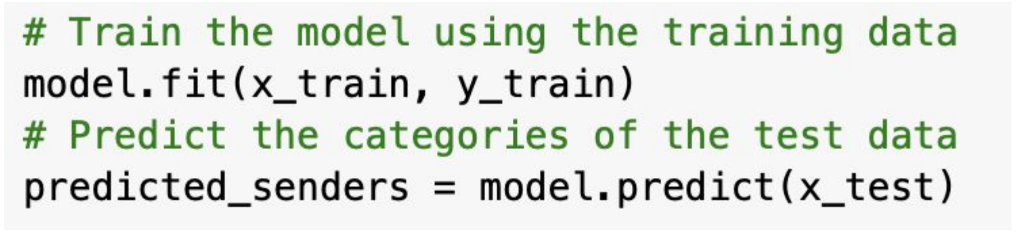
6. Vectorized the data and build the model

For example:

```
X = {__, __, __, __, __, __, __, ...          X = { 1, 0, 1, 1, 0, 0, 1, ... ──► each row represents an email

     __, __, __, __, __, __, __, ...    ⇨          0, 1, 0, 0, 1, 1, 1, ...

     ...                                           ...

     __, __, __, __, __, __, __, ...}              0, 0, 0, 1, 0, 1, 1, ...}
                                                                  │
                                                                  ▼
If this email contains the word = 1; else = 0         each column represents a word
```

```
# Build the model
model = make_pipeline(TfidfVectorizer(), MultinomialNB())
```

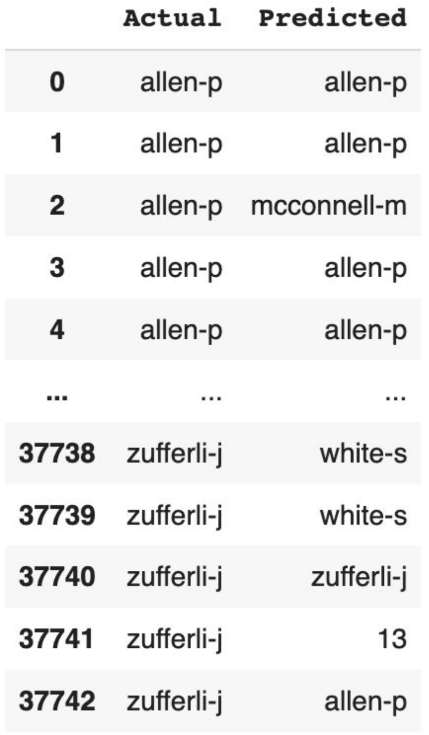
7. Train the model using the training data, and predict the senders of the test data



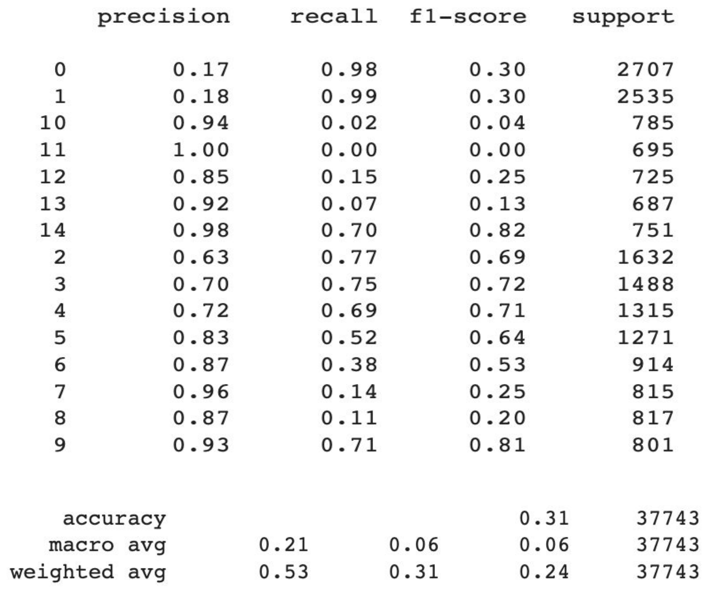
```
# Train the model using the training data
model.fit(x_train, y_train)
# Predict the categories of the test data
predicted_senders = model.predict(x_test)
```

SVM Classification

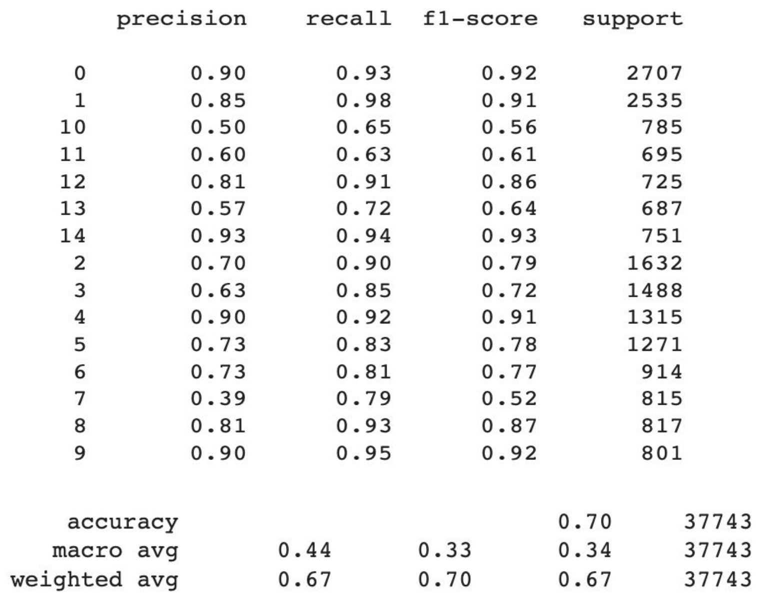
- Highly popular vector-space classification method broadly used for text categorization
- SVM (c = 0.01) demonstrates the higher accuracies compare to Naive Bayes
- We vectorized the data and performed linear dimensionality reduction
- Accuracy score: `0.701931483983785`



	Actual	Predicted
0	allen-p	allen-p
1	allen-p	allen-p
2	allen-p	mcconnell-m
3	allen-p	allen-p
4	allen-p	allen-p
...	...	...
37738	zufferli-j	white-s
37739	zufferli-j	white-s
37740	zufferli-j	zufferli-j
37741	zufferli-j	13
37742	zufferli-j	allen-p



```
             precision    recall  f1-score   support

           0       0.17      0.98      0.30      2707
           1       0.18      0.99      0.30      2535
          10       0.94      0.02      0.04       785
          11       1.00      0.00      0.00       695
          12       0.85      0.15      0.25       725
          13       0.92      0.07      0.13       687
          14       0.98      0.70      0.82       751
           2       0.63      0.77      0.69      1632
           3       0.70      0.75      0.72      1488
           4       0.72      0.69      0.71      1315
           5       0.83      0.52      0.64      1271
           6       0.87      0.38      0.53       914
           7       0.96      0.14      0.25       815
           8       0.87      0.11      0.20       817
           9       0.93      0.71      0.81       801

    accuracy                           0.31     37743
   macro avg       0.21      0.06      0.06     37743
weighted avg       0.53      0.31      0.24     37743
```

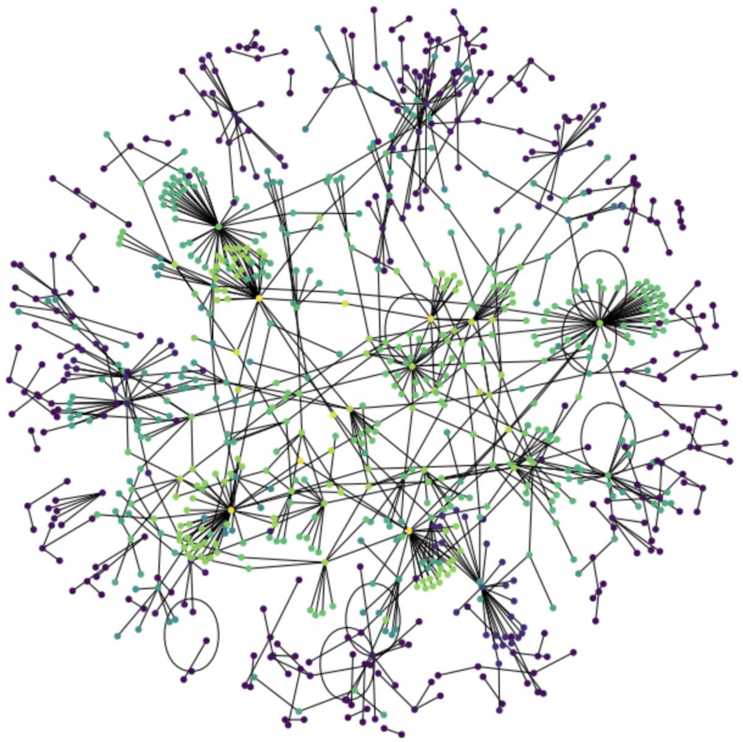


```
              precision    recall  f1-score   support

           0       0.90      0.93      0.92      2707
           1       0.85      0.98      0.91      2535
          10       0.50      0.65      0.56       785
          11       0.60      0.63      0.61       695
          12       0.81      0.91      0.86       725
          13       0.57      0.72      0.64       687
          14       0.93      0.94      0.93       751
           2       0.70      0.90      0.79      1632
           3       0.63      0.85      0.72      1488
           4       0.90      0.92      0.91      1315
           5       0.73      0.83      0.78      1271
           6       0.73      0.81      0.77       914
           7       0.39      0.79      0.52       815
           8       0.81      0.93      0.87       817
           9       0.90      0.95      0.92       801

    accuracy                           0.70     37743
   macro avg       0.44      0.33      0.34     37743
weighted avg       0.67      0.70      0.67     37743
```

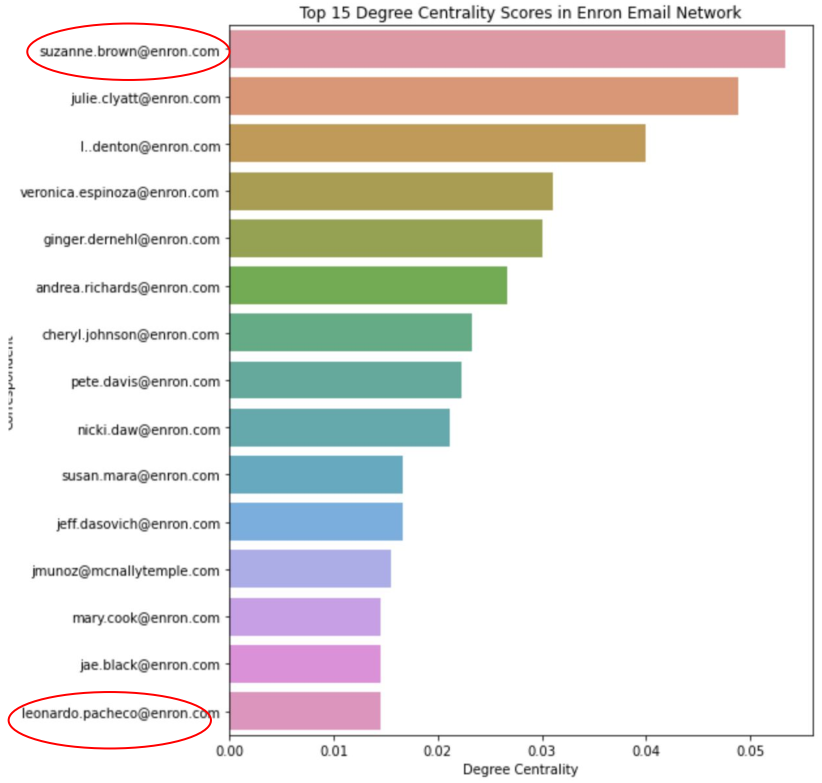
Naive Bayes **V.S.** **SVM**

Further Exploration

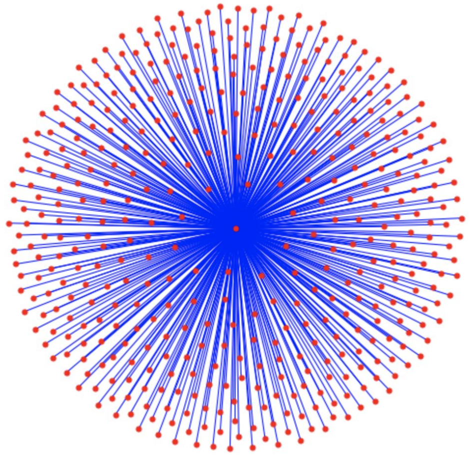
- Social network formed by emails?
- Analyze the clusters within the social network
- People who have larger centrality within the cluster has more connections
- People who are in the same cluster has higher chance of sending/receiving emails from the each other



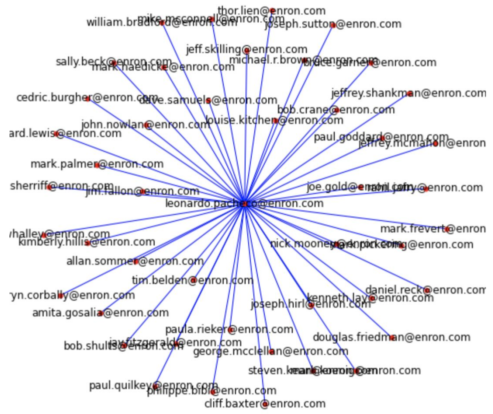
Network formed by 1000 random samples



- We can specify email address and draw the connection from that person



Suzanne



Leo

Conclusion

Accuracy:

SVM: 0.69444

Naive Bayes: 0.30766

From the accuracy rate, we can see that the SVM is higher than the Naive bayes.

In the future, we would like to use more model for this dataset, like random forest.

Our Link for coding:

https://colab.research.google.com/drive/1m0fpMGlfwQYVTpJP_qez-f2x0699UJk-?usp=sharing

https://colab.research.google.com/drive/1lCgVthAKJvxmTtFnKy73ZCnKAnZaVJZp?usp=sharing

Reference

E. Crawford, J. Kay, and E. McCreath: Automatic Induction of Rules for e-mail Classification. In ADCS2001 Proceedings of the Sixth Australasian Document Computing Symposium, pages 13-20, Coffs Harbour, NSW Australia, 2001.

E. Hung: Deduction of Procmail Recipes from Classified Emails. CMSC724 Database Management Systems, individual research project report. May, 2001

J. D. Brutlag, C. Meek: Challenges of the Email Domain for Text Classification. ICML 2000: 103-110

J. Rennie: ifile: An Application of Machine Learning to E-Mail Filtering. In Proc. KDD00 Workshop on Text Mining, Boston, 2000.

R. B. Segal and J. O. Kephart. MailCat: An Intelligent Assistant for Organizing E-Mail. In Proc. of the 3rd International Conference on Autonomous Agents, 1999.

S. Kiritchenko, S. Matwin: Email classification with co-training. In Proc. of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research, page 8, Toronto, Ontario, Canada, 2001

Q & A

Thank You!!