rolando_lab5

February 15, 2023

1 Lab 5 - EDA with Dimensionality Reduction

1.0.1 Jackson Rolando

1.1 Part 1 - Load the Data

We'll load the data from JSON files into a Pandas data frame:

[12]:
```
import glob
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[13]:
```
objects = []
for file in glob.glob('./email_json/*.json'):
    with open(file) as f:
        objects.append(json.load(f))
```

[14]:
```
df = pd.DataFrame(objects)
df.head()
```

[14]:
```
  category                                         to_address  \
0      ham  BREAKINGNEWS Subscribers<BREAKINGNEWS-Subscrib…
1     spam                     <theorize@plg.uwaterloo.ca>
2     spam           "Theorize" <theorize@plg.uwaterloo.ca>
3     spam                warwickktwarwic@speedy.uwaterloo.ca
4      ham                          R-help@stat.math.ethz.ch

                                     from_address  \
0           BREAKING NEWS<breakingnews@foxnews.com>
1                     "cschai" <cschai@syhmco.co.kr>
2  "Aegis Capital Group LLC" <Estela.Burch@smapxs…
3      "shar Nobis" <sharNobis@autotradebuyer.co.uk>
4                          jessica.gervais@tudor.lu

                                          subject  \
0                                       FNC Alert
1                                          rtfmub
```

```
2  Invitation to fill in the vacant position of a…
3                            Terrific gains possible!
4                           [R] time serie generation

                                                 body
0  PELOSI, REID SIGN WAR-SPENDING BILL THAT INCLU…
1  \n\n\n\n\n\n\n\nwyat\nlnpmoqrkhapibcegd\n\n\n\…
2  \n\n\n\nDear    sirs,\nAegis      Capital  Gro…
3  http://s6.bilder-hosting.de/img/7LR4W.jpg\nImp…
4  \nDear all,\n\nI would like to generate a regu…
```

The columns and types are as follows, we've changed the label to a categorical variable:

[15]:
```python
df.category = df.category.astype('category')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63542 entries, 0 to 63541
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   category      63542 non-null  category
 1   to_address    63141 non-null  object
 2   from_address  63542 non-null  object
 3   subject       63410 non-null  object
 4   body          63542 non-null  object
dtypes: category(1), object(4)
memory usage: 2.0+ MB
```

1.2 Part 2 - Extract Features

Here we'll convert the message bodies to Bag of Word vectors:

[16]:
```python
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(binary=True)
feat_mat = vectorizer.fit_transform(df.body)

print(feat_mat.shape)
print(feat_mat.sum())
```

```
(63542, 300984)
6885706
```

There are 63,542 rows (nonNull entries) and 300,984 columns (different words in the entire corpus).

We'll inspect the matrix:

[17]:

```python
print(f'Length of list: {len(list(vectorizer.vocabulary_))}\n')

print('First ten entries:')
first_some_keys = list(vectorizer.vocabulary_)[:10]
for key in first_some_keys:
    print(f'{key}: {vectorizer.vocabulary_[key]}')

print()
def search_word(word):
    index = vectorizer.vocabulary_[word]
    print(f'index of {word}: {index}')
    column = feat_mat[:, index]
    print(f'column for "{word}":\n{column}')
    print(f'shape: {column.shape}')
    print(f'number of occurances: {column.sum()} out of {column.shape[0]}␣
↪emails.\n')

words_to_search = ["work", "love", "different"]
for word in words_to_search:
    search_word(word)
```

```
Length of list: 300984

First ten entries:
pelosi: 216076
reid: 235309
sign: 249548
war: 285529
spending: 254951
bill: 99386
that: 266767
includes: 164213
iraq: 167840
pullout: 227377

index of work: 289649
column for "work":
  (2, 0)        1
  (22, 0)       1
  (30, 0)       1
  (37, 0)       1
  (49, 0)       1
  (67, 0)       1
  (75, 0)       1
  (92, 0)       1
  (93, 0)       1
  (100, 0)      1
```

```
  (114, 0)      1
  (121, 0)      1
  (126, 0)      1
  (127, 0)      1
  (130, 0)      1
  (134, 0)      1
  (149, 0)      1
  (151, 0)      1
  (153, 0)      1
  (156, 0)      1
  (159, 0)      1
  (170, 0)      1
  (176, 0)      1
  (189, 0)      1
  (207, 0)      1
  :     :
  (63371, 0)    1
  (63376, 0)    1
  (63384, 0)    1
  (63385, 0)    1
  (63386, 0)    1
  (63398, 0)    1
  (63410, 0)    1
  (63421, 0)    1
  (63428, 0)    1
  (63438, 0)    1
  (63444, 0)    1
  (63461, 0)    1
  (63467, 0)    1
  (63473, 0)    1
  (63474, 0)    1
  (63476, 0)    1
  (63480, 0)    1
  (63482, 0)    1
  (63490, 0)    1
  (63504, 0)    1
  (63516, 0)    1
  (63527, 0)    1
  (63528, 0)    1
  (63532, 0)    1
  (63536, 0)    1
shape: (63542, 1)
number of occurances: 6466 out of 63542 emails.

index of love: 183355
column for "love":
  (16, 0)       1
  (22, 0)       1
```

(36, 0)	1
(54, 0)	1
(68, 0)	1
(79, 0)	1
(96, 0)	1
(107, 0)	1
(144, 0)	1
(151, 0)	1
(156, 0)	1
(188, 0)	1
(239, 0)	1
(266, 0)	1
(292, 0)	1
(297, 0)	1
(355, 0)	1
(362, 0)	1
(363, 0)	1
(421, 0)	1
(431, 0)	1
(463, 0)	1
(465, 0)	1
(467, 0)	1
(482, 0)	1
: :	
(62596, 0)	1
(62601, 0)	1
(62643, 0)	1
(62645, 0)	1
(62700, 0)	1
(62703, 0)	1
(62748, 0)	1
(62761, 0)	1
(62837, 0)	1
(62846, 0)	1
(62885, 0)	1
(62916, 0)	1
(63010, 0)	1
(63082, 0)	1
(63132, 0)	1
(63168, 0)	1
(63194, 0)	1
(63282, 0)	1
(63290, 0)	1
(63354, 0)	1
(63373, 0)	1
(63406, 0)	1
(63453, 0)	1
(63510, 0)	1

```
  (63532, 0)    1
shape: (63542, 1)
number of occurances: 2043 out of 63542 emails.

index of different: 125499
column for "different":
  (15, 0)       1
  (34, 0)       1
  (58, 0)       1
  (60, 0)       1
  (100, 0)      1
  (126, 0)      1
  (134, 0)      1
  (148, 0)      1
  (154, 0)      1
  (156, 0)      1
  (157, 0)      1
  (161, 0)      1
  (170, 0)      1
  (182, 0)      1
  (183, 0)      1
  (186, 0)      1
  (252, 0)      1
  (292, 0)      1
  (310, 0)      1
  (329, 0)      1
  (334, 0)      1
  (341, 0)      1
  (355, 0)      1
  (370, 0)      1
  (390, 0)      1
  :     :
  (63004, 0)    1
  (63024, 0)    1
  (63030, 0)    1
  (63053, 0)    1
  (63067, 0)    1
  (63071, 0)    1
  (63077, 0)    1
  (63083, 0)    1
  (63095, 0)    1
  (63100, 0)    1
  (63116, 0)    1
  (63185, 0)    1
  (63194, 0)    1
  (63217, 0)    1
  (63251, 0)    1
  (63275, 0)    1
```

```
  (63297, 0)	1
  (63436, 0)	1
  (63444, 0)	1
  (63467, 0)	1
  (63474, 0)	1
  (63480, 0)	1
  (63517, 0)	1
  (63532, 0)	1
  (63536, 0)	1
shape: (63542, 1)
number of occurances: 2893 out of 63542 emails.
```

1.3 Part 3 - Dimensionality Reduction

We'll combine several columns with high covariance:

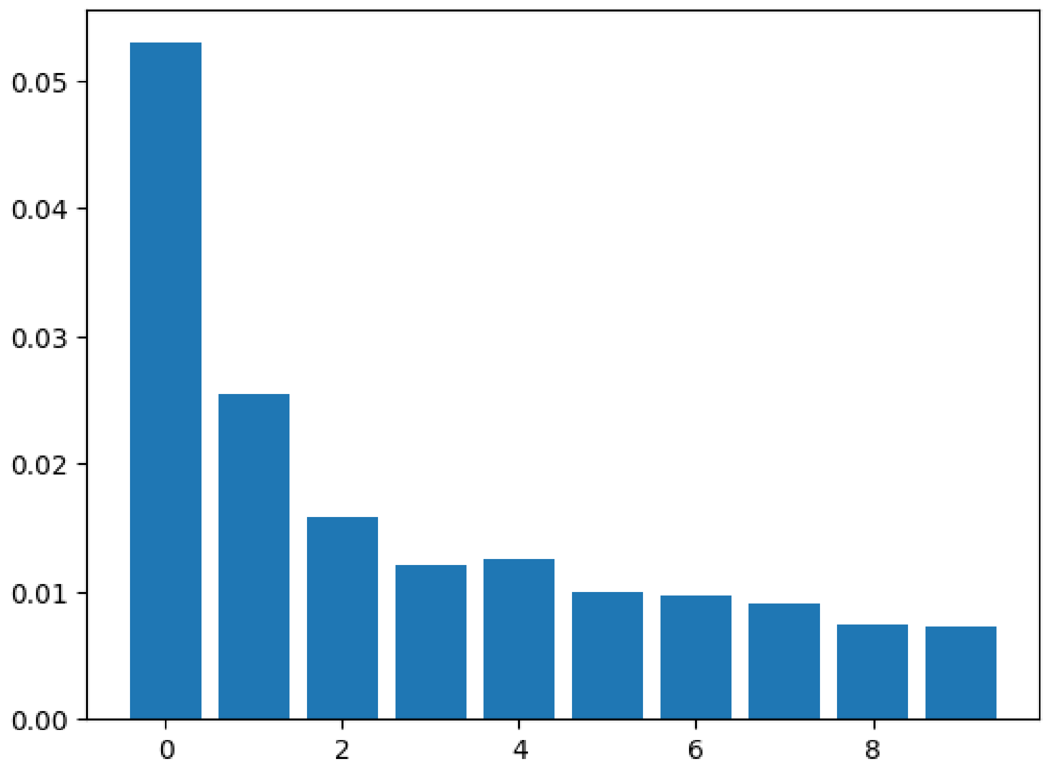
[18]:
```python
from sklearn.decomposition import TruncatedSVD

condenser = TruncatedSVD(n_components=10)
condensed_mat = condenser.fit_transform(feat_mat)
```

[19]:
```python
print(condenser.explained_variance_ratio_)
plt.bar(np.arange(condenser.explained_variance_ratio_.size), condenser.
 ↪explained_variance_ratio_)
```

```
[0.05291813 0.02548106 0.01589102 0.0120492  0.01252032 0.0099801
 0.00969514 0.00913666 0.00740042 0.00726925]
```

[19]:
```
<BarContainer object of 10 artists>
```



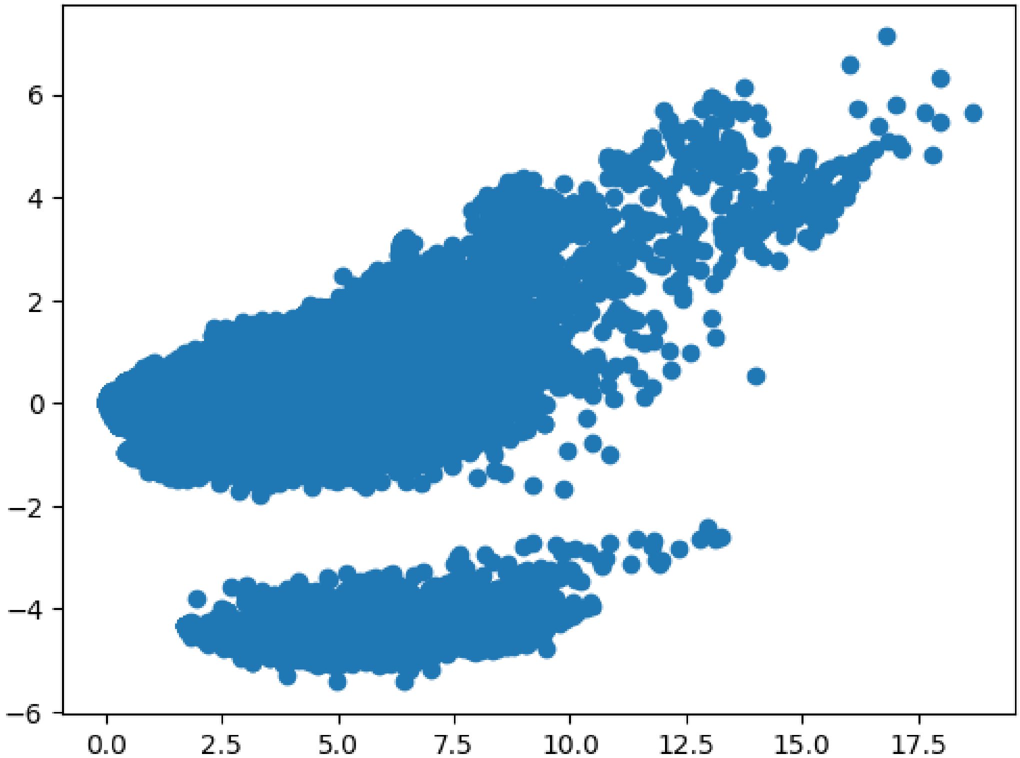
The first two components have the highest explained variance ratios, over 50%, meaning they represent most of the variance in the original data.

1.4 Part 4 - Visualization

We'll plot the components with the highest explained variance:

[30]:
```
i, j = (0, 1)
plt.scatter(condensed_mat[:, i], condensed_mat[:, j])
```

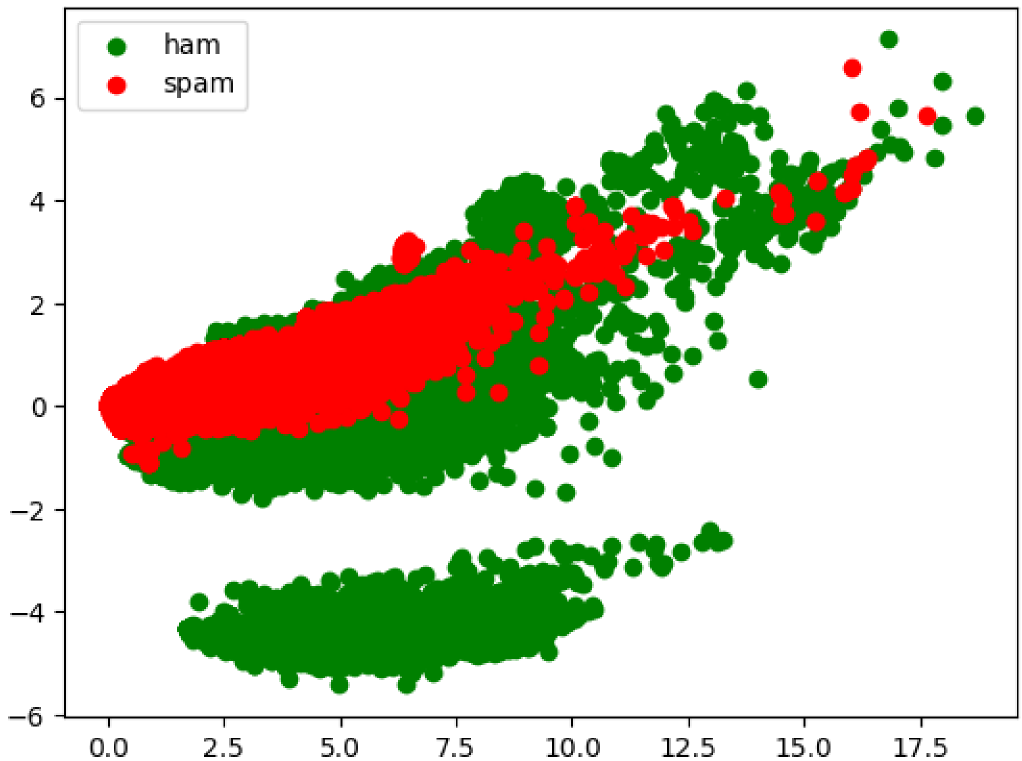
[30]: <matplotlib.collections.PathCollection at 0x7fa3534ad1f0>



Here we'll add labels and color-coding to the mix:

[31]:
```
colors = {'ham': 'green', 'spam': 'red'}

fig, ax = plt.subplots()
for label in np.unique(df["category"]):
    indices = np.where(df["category"] == label)
    ax.scatter(condensed_mat[indices, i], condensed_mat[indices, j],␣
 ↪label=label, c=colors[label])
ax.legend()
plt.show()
```



1.5 Reflection Questions:

1. Each JSON file has a sender email, a receiver email, a subject, the body of the message, and whether the message was ham or spam. The body is the content of the email, a corpus that we're encoding.
2. 63542 * 300984 * 4 = 76.5 GB: Wow, I definitely do not have this much RAM.
3. 6885706 nonzero entries * (4 bytes + 4 bytes) + 63542 rows * 4 bytes = 55.3 MB: much less memory here.
4. 100 * 6885706 nonzero entries / (63542 rows * 300984 columns) = 3.6% filled
5. A sparse matrix is not only ideal for this application, but required in order to run on any normal computer. There isn't enough memory to hold the entire filled matrix, potentially not even enough disk storage on my machine. The sparse matrix makes the matrix computable.
6. There are two main groups in the data, with very few points outside the clusters. This is generally true for the other columns as well. This makes sense, as the new matrix tries to preserve the spread of the original data. Since most of the variance is preserved with the firs column, it makes sense for it to spread wider than the next column, which still has a good amount of spread, but not as much, hence the horizontal groupings.
7. Adding in the labels, it looks like the ham messages have some key features that spam messages do not, and the spam messages probably consist mostly of very common wordings, contained in all emails.