schulzdLab5

January 25, 2021

1 Lab 5 - EDA with Dimensionality Reduction

David Schulz

1.1 Introduction

High dimensional data sets have too many variables for us to analyze each variable individually. We need to turn to more sophisticated techniques such dimensionality reduction and clustering. In this lab, we are going to analyze 63,542 emails. We will convert the raw text into a feature matrix using a "bag of words" model. Each column of the feature matrix corresponds to one word, each row corresponds to one email, and the entry stores the number of times that word was found in that email. We will perform dimensionality reduction using the Truncated SVD method, cluster the emails, and compare the "inherent" structure to the given class labels.

1.2 Part I: Load the Data

[1]:
```
import pandas as pd
from glob import glob
import json

data = []

files = glob('email_json/*.json', recursive=True)

for single_file in files:
    with open(single_file, 'r') as f:
        json_file = json.load(f)
        data.append({
            'category': json_file['category'],
            'to_address': json_file['to_address'],
            'from_address': json_file['from_address'],
            'subject': json_file['subject'],
            'body': json_file['body']
        })
```

[2]:
```
data = pd.DataFrame.from_dict(data)
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63542 entries, 0 to 63541
Data columns (total 5 columns):
body            63542 non-null object
category        63542 non-null object
from_address    63542 non-null object
subject         63410 non-null object
to_address      63141 non-null object
dtypes: object(5)
memory usage: 2.4+ MB
None
```

1.3 Part II: Extract Features

[3]:
```
from sklearn.feature_extraction.text import CountVectorizer

vect = CountVectorizer(binary=True)
X = vect.fit_transform(data['body'])
x = X.toarray()
```

[4]:
```
import numpy as np

print("Rows:", x.shape[0], " Columns:", x.shape[1])
print("Nonzero Entries:", np.count_nonzero(x))
print("Number of Vocabulary Entries:", len(vect.vocabulary_))
```

```
Rows: 63542  Columns: 300984
Nonzero Entries: 6885706
Number of Vocabulary Entries: 300984
```

[5]:
```
cols = vect.get_feature_names()
work = cols.index("work")
love = cols.index("love")
different = cols.index("different")

print("Work Column:", x[:, work])
print("Love Column:", x[:, love])
print("Different Column:", x[:, different])
```

```
Work Column: [0 0 0 … 0 0 0]
Love Column: [1 0 0 … 0 0 0]
Different Column: [1 0 0 … 0 0 0]
```

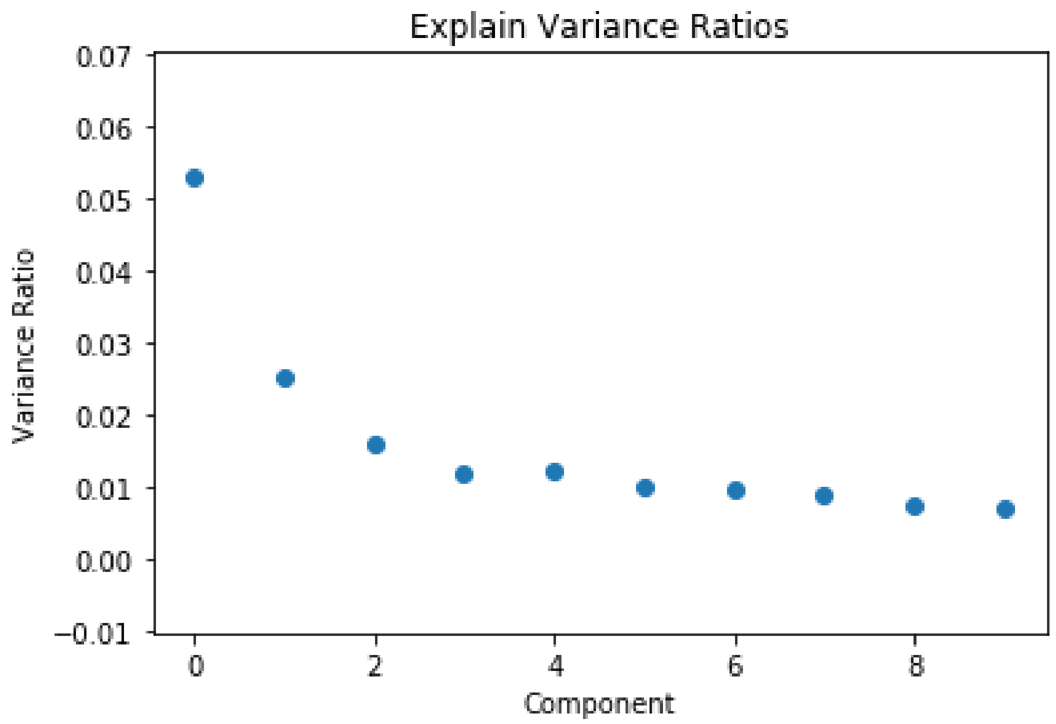
1.4 Part III: Dimensionality Reduction

[13]:
```python
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=10)

svd_fit = svd.fit_transform(X)

variances = svd.explained_variance_ratio_

print(variances)
```

```
[0.05291813 0.02548106 0.01589103 0.01204933 0.01252034 0.00997993
 0.00969524 0.00913673 0.00740018 0.00726906]
```

[14]:
```python
import matplotlib.pyplot as plt

plt.title("Explain Variance Ratios")
plt.xlabel("Component")
plt.ylabel("Variance Ratio")
plt.scatter(range(10), variances)
```

[14]:
```
<matplotlib.collections.PathCollection at 0x7f0a1a914c90>
```

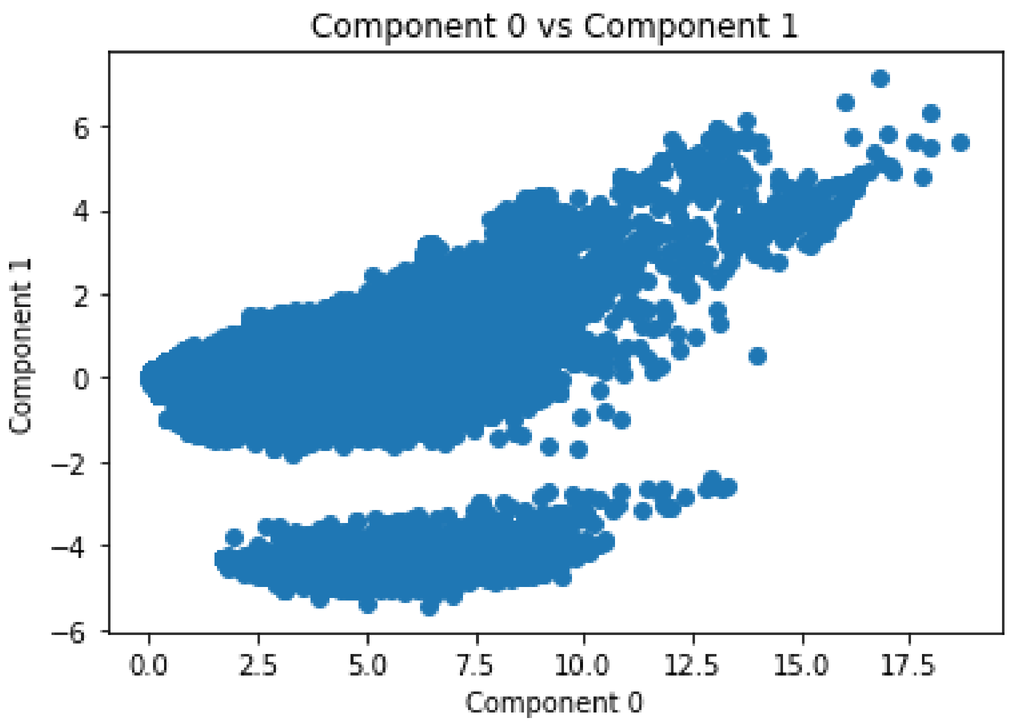


The two components with the highest explained variance ratios are 0 and 1.

1.5 Part IV: Visualization

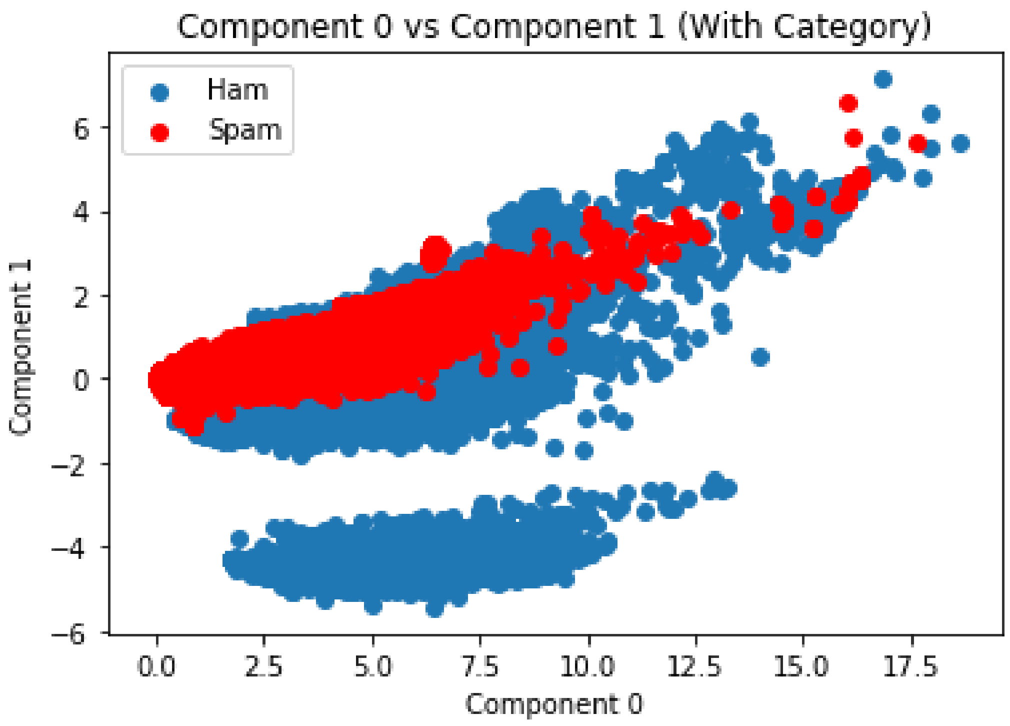
[17]:
```
plt.title("Component 0 vs Component 1")
plt.xlabel("Component 0")
plt.ylabel("Component 1")
plt.scatter(svd_fit[:, 0], svd_fit[:, 1])
```

[17]:
```
<matplotlib.collections.PathCollection at 0x7f0a1a2565d0>
```



[22]:
```
plt.title("Component 0 vs Component 1 (With Category)")
plt.xlabel("Component 0")
plt.ylabel("Component 1")

ham = svd_fit[data['category'] == 'ham']
spam = svd_fit[data['category'] == 'spam']

plt.scatter(ham[:, 0], ham[:, 1], label="Ham")
plt.scatter(spam[:, 0], spam[:, 1], color="red", label="Spam")
plt.legend()
```

[22]:
```
<matplotlib.legend.Legend at 0x7f0a1afa39d0>
```



1.6 Reflection Questions

1. Use a text editor to look at one of the JSON files. Describe the JSON file. What are the keys and values of the JSON document? What do you think they correspond to?
   - It's a comma-separated dictionary/map. The keys are the strings before the colons and the values are the strings after.
2. Assume that 32-bit (4-byte) floating point values are used to store the counts. Calculate the memory usage of a dense matrix with those dimensions.
   - 63,542 * 300,984 * 4 = 76,500,501,312 bytes
3. The vectorizer returns a sparse matrix in compressed row format (CSR). Assume that the sparse matrix uses one 32-bit (4-byte) floating point number and one 32-bit (4-byte) integer for each nonzero entry and one 32-bit (4-byte) integer for each row. Calculate the memory usage for the sparse matrix.
   - (4 * 63,542) + (8 * 6,885,706) = 254,168 + 55,085,648 = 55,339,816 bytes
4. Calculate the sparsity ratio (100 * number of nonzero entries divided by maximum possible entries).
   - (100 * 6,885,706) / 19,125,125,328 = 0.036
5. Based on your analysis, do you think that the sparse matrix is better suited for this situation? If so, why?
   - Yes because the sparse matrix is severely more resource efficient.
6. What do you notice when looking at the first scatter plot? Why do you think this pattern appears?

    - The points are very clearly separated into two clusters. I assume one cluster represents the spam emails while the other one represents the ham emails.
7. In the second plot, how would you describe the relationship between the pattern you observe and the ham and spam messages?
    - The bottom cluster probably represents the emails where the model was much more certain they were ham, likely because of specific words that were almost never seen in spam emails, while the rest are much less certain and sometimes undeterministic.

1.7 Conclusion

When dealing with high dimensional data sets, dimensionality reduction techniques such as SVD can be used to create clusters and more efficiently analyze and compare inherent structures with their given class labels.