# Identifying Themes from Technical Paper Abstracts

**Prince Joseph Erneszer Javier**
MSc Data Science
Asian Institute of Management

## Executive Summary

In this analysis, we identified common themes or topics in academic papers based on paper abstract characteristics. The dataset, containing 500 papers abstracts, titles, and corresponding authors, was prepared by the Access lab from Asian Institute of Management, Philippines. The problem being tackled is identifying the common topics or themes in the corpus of abstracts based on the words used (features). The abstracts were analyzed using two theme-extraction methods: k Means Clustering and Latent Dirichlet Allocation(LDA).

TF-IDF bag of words vectorization with Euclidean normalization was applied on the corpus of abstracts. Words that occur in less than 10% of the documents were dropped, as well as those that occur in more than 50% of the documents. This was done because the rarest and most common words do not contribute to new information about the themes in the abstracts. For similar reasons, common english words were also removed from the abstracts.

Principal Component Analysis (PCA) was used to inspect visually the clusters generated by k Means and LDA on the abstracts. The abstract TF-IDF vectors were projected on the first two principal components (2D plot) and first three principal components (3D plot).

k Means clustering was able to form clear boundaries between clusters for two clusters or three clusters. Beyond this, overlaps between clusters form. This means that two or three clusters is preferrable to distinguish the groups clearly. When three clusters were used for LDA however, two clusters overlap. For LDA, two clusters was found to clearly separate abstracts. The table below shows the top words in each theme for each clustering method.

| Method | Clusters | Themes |
|---|---|---|
| k Means | 2 | model, data, method |
| | | learning, data, neural |
| | 3 | time, function, problems |
| | | model, data, learning |
| | | structure, nodes, model |
| LDA | 2 | network, networks, learning |
| | | problem, time, function |

Although k Means was able to generate two or three separate clusters, as projected on the principal components, the clusters are more difficult to interpret and some top words overlap. For k=3 we can infer that one cluster is about time-related problems, but the other two clusters are more vague. LDA, on the other hand was able to identify networks as a clear theme, and the other is about time, similar to what k means identified.

k Means and LDA can be used to cluster technical paper abstracts. It is recommended to use LDA since it was able to show distinctly two separate themes namely, 1) time, problem, function and 2) network, networks, learning.

## Data Description

The dataset was prepared by data scientists from The Asian Institute of Management Access Lab. The dataset contains information about 500 research papers. The information are:

1. title - title of the paper
2. abstract - abstract of the paper
3. date created - date the paper was created
4. authors - authors of the paper

## Python Packages

`Numpy` gives a wide array of functionality when dealing with arrays (pun intended). `Matplotlib` allows us to make charts easily. `Pandas` enables us to transform data into tables that are more understandable and transformable. `Counter` enables us to quickly count unique elements in a sequence like a dictionary.

Six packages were imported from `Scikit-learn`, which contains many machine learning algorithms. `ENGLISH_STOP_WORDS` imports words that are most commonly used in English like "the", "of", and "for", so they can be removed from the analysis as they don't add useful information. `CountVectorizer` converts a text into a Bag-of-Words vector with values corresponding to the counts of each word. `TfidfVectorizer` works the same way except that the values in the bag of words vectors are in a way "regularized", reducing values of least important words and increasing values of more important words.

`PCA`, referring to principal component analysis, is a package that allows quick computation of principal components, which in the simplest of terms is like viewing a camera at an object and finding the angle that best identifies the object. `KMeans` is used to perform clustering based on how similar the elements or vectors are. Every element in KMeans is clustered into one unique cluster. `LatentDirichletAllocation`, meanwhile, can be used to cluster bag of words vectors based on how frequently words appear together, while at the same time acknowledging that each vector or document can have multiple topics or clusters (in contrast to k Means).

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        %matplotlib inline
        import pandas as pd
        from collections import Counter

        from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.feature_extraction.text import TfidfVectorizer

        from sklearn.decomposition import PCA
        from sklearn.cluster import KMeans
        from sklearn.decomposition import LatentDirichletAllocation
```

## Exploratory Data Analysis

### Importing Data

The dataset is saved in a comma-separated values (csv) file `arxiv-collection.csv`. The dataset was loaded in a pandas dataframe, `_df`, which organizes the data in table form.

The dataset contains 5 columns: `Unnamed: 0`, `title`, `abstract`, `created`, and `authors`. Upon further inspection, we see that `Unnamed: 0` is just the index of the row, and so was

In [2]:
```
_df = pd.read_csv('arxiv-collection.csv')
_df.reset_index()
print("No. of features/columns in the dataset: {}".format(_df.shape[1]))
print("No. of samples/rows in the dataset: {}".format(_df.shape[0]))
print("Columns: {}".format(_df.columns))
```

```
No. of features/columns in the dataset: 5
No. of samples/rows in the dataset: 500
Columns: Index(['Unnamed: 0', 'title', 'abstract', 'created', 'authors'], dtype='object')
```

In [3]:
```
print("Top 3 rows in Unnamed: 0:\n{}".format(_df['Unnamed: 0'].head(3)))
print("Top 3 rows in Unnamed: 0:\n{}".format(_df['Unnamed: 0'].tail(3)))
```

```
Top 3 rows in Unnamed: 0:
0    0
1    1
2    2
Name: Unnamed: 0, dtype: int64
Top 3 rows in Unnamed: 0:
497    497
498    498
499    499
Name: Unnamed: 0, dtype: int64
```

### Cleaning the Data

There are no null values in the dataset.

In [4]:
```
_df.info()

print()
a = np.sum([_df.isnull().any()])

print("No. of columns with null: {}".format(a))
print("===========")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 5 columns):
Unnamed: 0    500 non-null int64
title         500 non-null object
abstract      500 non-null object
created       500 non-null object
authors       500 non-null object
dtypes: int64(1), object(4)
memory usage: 19.6+ KB

No. of columns with null: 0
===========
```

`Unnamed: 0` only contains indices of the rows in the dataset. We dropped this column and save the new dataset in `df`.

In [5]:
```
df = _df.drop('Unnamed: 0', axis=1)
print("No. of features/columns in the dataset: {}".format(df.shape[1]))
print("No. of samples/rows in the dataset: {}".format(df.shape[0]))
print("Columns: {}".format(df.columns))
```

```
No. of features/columns in the dataset: 4
No. of samples/rows in the dataset: 500
Columns: Index(['title', 'abstract', 'created', 'authors'], dtype='object')
```

There are only 490 unique abstracts in the dataset, and redundant abstracts need to be removed. We saved the new corpus of abstracts into a list `abstracts`.

In [6]:
```
print("Number of unique abstracts in dataset:", len(set(df.abstract)))
abstracts = list(set(df.abstract))
```

```
Number of unique abstracts in dataset: 490
```

Similarly, there are only 490 unique abstracts in the dataset, and redundant abstracts need to be removed. We saved the new corpus of abstracts into a list `titles`.

In [7]:
```
print("Number of unique titles in dataset:", len(set(df.title)))
titles = list(set(df.title))
```

```
Number of unique titles in dataset: 490
```

### Exploring the Data

We convert the titles and the abstracts into bag-of-words (BoW) vector representation saved to `bow_title`. When converting to BoW, we remove commonly used English words like "and", "or", and "the" by applying `stop_words=`"english". The corpus of titles contain 1754 words and 490 titles.

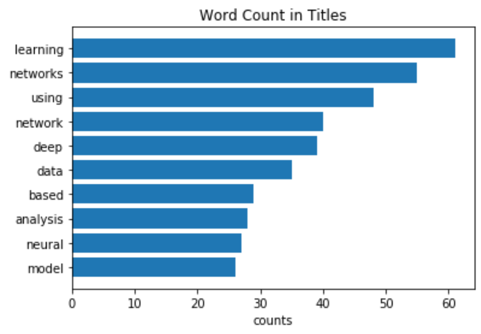
In [8]:
```
# Converting titles to BoW vector representation
w = titles
vect_title = CountVectorizer(stop_words='english').fit(w)
bow_title = vect_title.transform(w)
print("No. of words considered in corpus:", len(vect_title.vocabulary_))
print("No. of documents converted to BoW vectos:",
      (bow_title.toarray().shape[0]))
print("Length of each BoW vector:", (bow_title.toarray().shape[1]))
```

```
No. of words considered in corpus: 1754
No. of documents converted to BoW vectos: 490
Length of each BoW vector: 1754
```

There are 490 titles in the dataset. Below are the top ten most common words in the title identified after preprocessing the data.

In [9]:
```
counts = np.array((bow_title.sum(axis=0)).tolist()[0])
inds = np.argsort(counts)[::-1]
feats = np.array(vect_title.get_feature_names())
a = pd.Series(feats[inds], name='Word')
b = pd.Series(counts[inds], name='Count')
c = pd.concat([a, b], axis=1)
display(c.head(10))

plt.barh(sorted(a[:10]), sorted(b[:10]))
plt.yticks(np.arange(10), a[:10][::-1])
plt.title('Word Count in Titles')
plt.xlabel('counts')
plt.show()
```

| | Word | Count |
|---|---|---|
| **0** | learning | 61 |
| **1** | networks | 55 |
| **2** | using | 48 |
| **3** | network | 40 |
| **4** | deep | 39 |
| **5** | data | 35 |
| **6** | based | 29 |
| **7** | analysis | 28 |
| **8** | neural | 27 |
| **9** | model | 26 |



We converted the abstracts into BoW, again removing commonly used words. The corpus of abstracts contain 7602 words and 490 abstracts.

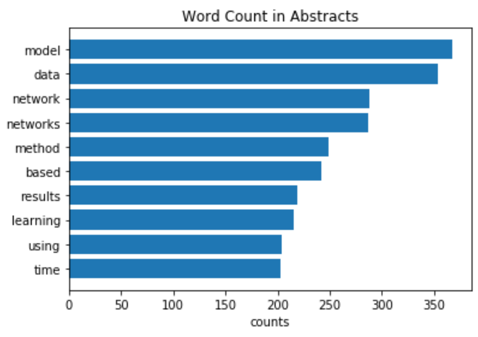
In [10]:
```
# Converting abstracts to BoW vector representation
w = abstracts
vect_title = CountVectorizer(stop_words='english').fit(w)
bow_title = vect_title.transform(w)
print("No. of words considered in corpus:", len(vect_title.vocabulary_))
print("No. of documents converted to BoW vectos:",
      (bow_title.toarray().shape[0]))
print("Length of each BoW vector:", (bow_title.toarray().shape[1]))
```

```
No. of words considered in corpus: 7602
No. of documents converted to BoW vectos: 490
Length of each BoW vector: 7602
```

Below are the most used words in the abstract.

In [11]:
```
counts = np.array((bow_title.sum(axis=0)).tolist()[0])
inds = np.argsort(counts)[::-1]
feats = np.array(vect_title.get_feature_names())
a = pd.Series(feats[inds], name='Word')
b = pd.Series(counts[inds], name='Count')
c = pd.concat([a, b], axis=1)
display(c.head(10))

plt.barh(sorted(a[:10]), sorted(b[:10]))
plt.yticks(np.arange(10), a[:10][::-1])
plt.title('Word Count in Abstracts')
plt.xlabel('counts')
plt.show()
```

| | Word | Count |
|---|---|---|
| 0 | model | 367 |
| 1 | data | 354 |
| 2 | network | 288 |
| 3 | networks | 287 |
| 4 | method | 249 |
| 5 | based | 242 |
| 6 | results | 219 |
| 7 | learning | 215 |
| 8 | using | 204 |
| 9 | time | 203 |



Below are the title, abstract, date created, and authors of the paper in the first row.

In [12]:
```
print("Title of paper in 1st row: {}\n".format(df.title[0]))
print("Abstract of paper in 1st row: {}\n".format(abstracts[0]))
print("Date Paper in 1st row was created: {}\n".format(df.created[0]))
print("Authors of paper in 1st: {}\n".format(df.authors[0]))
```

```
Title of paper in 1st row: designing random graph models using variational autoencoders with   applications to chemical design

Abstract of paper in 1st row: some recent works revealed that deep neural networks (dnns) are vulnerable to so-called adversar
ial attacks where input examples are intentionally perturbed to fool dnns. in this work, we revisit the dnn training process t
hat includes adversarial examples into the training dataset so as to improve dnn's resilience to adversarial attacks, namely,
adversarial training. our experiments show that different adversarial strengths, i.e., perturbation levels of adversarial exam
ples, have different working zones to resist the attack. based on the observation, we propose a multi-strength adversarial tra
ining method (mat) that combines the adversarial training examples with different adversarial strengths to defend adversarial
attacks. two training structures - mixed mat and parallel mat - are developed to facilitate the tradeoffs between training tim
e and memory occupation. our results show that mat can substantially minimize the accuracy degradation of deep learning system
s to adversarial attacks on mnist, cifar-10, cifar-100, and svhn.

Date Paper in 1st row was created: 2018-02-14

Authors of paper in 1st: ['samanta', 'de', 'ganguly', 'gomez-rodriguez']
```

## Models

### Converting to BoW with TF-IDF

Aside from converting text into a bag-of-words vector (BoW) by counts, another method is by getting the term frequency-inverse document frequency (TF-IDF) values. Simply put, the counts (term frequency) are multiplied by a factor that reduces the value of common, uninformative words based on how many documents contain those words. The equation is:

$$TFIDF = TF \times log\left(\frac{N+1}{N_w+1}\right) + 1$$

Where, $TF$ is the term frequency, $N$ is the total number of documents in the corpus, and $N_w$ is the number of documents containing the word $w$.

In this paper, `Scikit-learn`'s `TfidfVectorizer` package was used. The words that occur in less than 10% of the documents were removed as well as those that occur in more than 50% of the documents, to eliminate more non-value adding words.

## K Means Clustering

K Means Clustering is used to cluster unlabeled datapoints based on the similarity of their features. The parameter k, corresponds to the number of clusters that will be generated.

The abstracts were first converted into BoW vectors using Tfidf vectorizer. As mentioned above, words that occur in less than 10% of the documents or more than 50% of the documents were dropped, "stop words" or common English words like "the", "and", or "or" were also dropped.

Afterwards, k Means clustering was applied for 2, 3, 4, and 5 clusters. The findings are discussed and visualized below.

***2 Clusters***

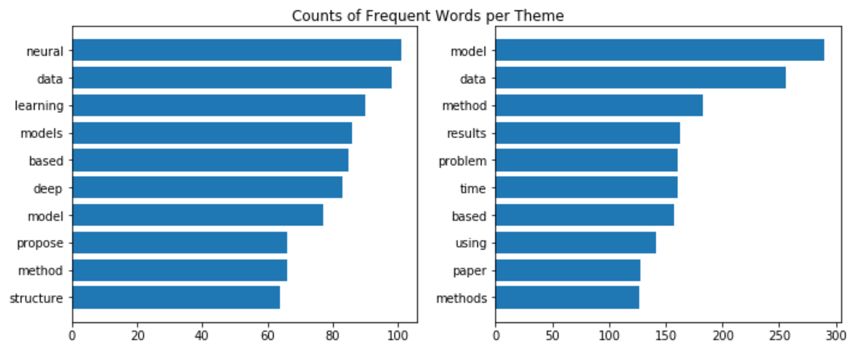
Because of the removal of the most common words, the number of elements in the BoW vector was reduced to 79. This value is consistent for all models used here.

In [13]:
```
def clustering(docs, clusters=2):
    vect = TfidfVectorizer(min_df=0.1, max_df=0.5, stop_words="english")
    X = vect.fit_transform(docs)
    kmeans = KMeans(n_clusters=clusters, max_iter=500).fit(X)
    # kmeans.labels_

    big_inds = []
    for c in range(clusters):
        small_inds = [i for i in range(
            len(kmeans.labels_)) if kmeans.labels_[i] == c]
        big_inds.append(small_inds)

    clustered_docs = []
    for i in range(clusters):
        cluster_inds = big_inds[i]
        small_docs = np.array(docs)[cluster_inds]
        clustered_docs.append(small_docs)

    print("No. of clusters:", clusters)
    print("No. of words:", len(vect.get_feature_names()))

    return kmeans, clustered_docs

clusters = 2
kmeans, clustered_abs = clustering(abstracts, clusters)
```

```
No. of clusters: 2
No. of words: 79
```

The most important words in the abstracts per theme or cluster are shown below.

In [14]:
```
# Showing most frequent words per cluster
def themes_knn(clustered_docs, clusters=2):
    """
    Imput
    =====
    clusters: Int. Number of clusters
    clustered_docs: List of lists of clustered docs

    Outpu
    =====
    Dataframe of most common words per cluster.
    Bar chart of most common words per cluster
    """
    cv = CountVectorizer(min_df=0.1, max_df=0.5, stop_words="english")
    df_0 = pd.DataFrame()
    max_vals = []

    for i in range(clusters):
        cv_fit = cv.fit_transform(clustered_docs[i])
        feature_names = np.array(cv.get_feature_names())

        vals = cv_fit.toarray().sum(axis=0)  # get total count per word
        sorting = np.argsort(vals)[::-1]
        # Highest Counts

        s = pd.Series(feature_names[sorting])
        c = pd.Series(vals[sorting])
        df_0['Theme ' + str(i)] = s
        df_0['Counts ' + str(i)] = c

        max_vals.append(vals[sorting])

    plt.figure(figsize=(10, 4))
    for i in range(clusters):
        plt.subplot(1, clusters, i + 1)
        x = df_0.iloc[:, 2*i][:10]
        y = max_vals[i][:10]
        plt.yticks(np.arange(10), x[::-1])
        plt.barh(sorted(x), sorted(y))

    plt.suptitle('Counts of Frequent Words per Theme', y=1)
    plt.tight_layout()
    plt.show()

    return df_0


df = themes_knn(clustered_abs, clusters)
display(df.head(10))
```



| | Theme 0 | Counts 0 | Theme 1 | Counts 1 |
|---|---|---|---|---|
| **0** | neural | 101 | model | 290.0 |
| **1** | data | 98 | data | 256.0 |
| **2** | learning | 90 | method | 183.0 |
| **3** | models | 86 | results | 163.0 |
| **4** | based | 85 | problem | 161.0 |
| **5** | deep | 83 | time | 161.0 |
| **6** | model | 77 | based | 157.0 |
| **7** | propose | 66 | using | 141.0 |
| **8** | method | 66 | paper | 128.0 |
| **9** | structure | 64 | methods | 127.0 |

Using two clusters, the main themes are "data", "model", and "method" for the first theme and "neural", "model", and "nodes" for the second theme. Note that "model" is present in both themes, meaning that the themes partially overlap. One theme might be about time-related problems, and the other might be about deep learning and neural networks.

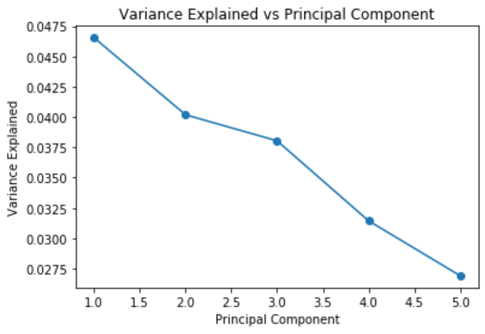
The number of abstracts contained in each theme or cluster are shown below.

In [15]:
```
for i in range(clusters):
    print("Abstracts with Theme {}: {}".format(i,len(clustered_abs[i])))
```

```
Abstracts with Theme 0: 143
Abstracts with Theme 1: 347
```

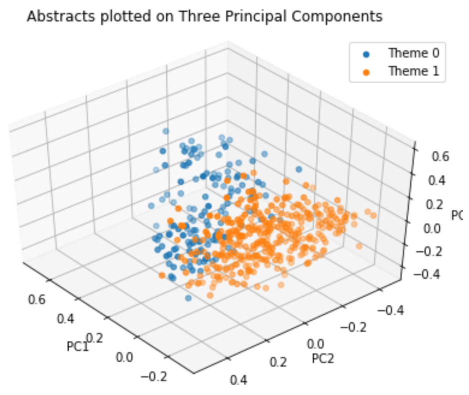
Principal component analysis was performed to find the top three components that explain the most information (variance). Plotting the variance explained (information) vs the nth principal component, we observe that the top three principal components explain the most information.

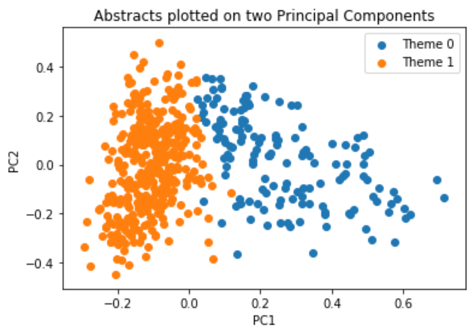
In [16]:
```
comps = 5

vect = TfidfVectorizer(min_df=0.1, max_df=0.8, stop_words="english")
X = vect.fit_transform(abstracts)
pca = PCA(n_components=comps)
pca.fit(X.toarray())
X = pca.transform(X.toarray())
y_ = pca.explained_variance_ratio_
plt.plot(range(1, comps + 1), y_, marker='o')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.title('Variance Explained vs Principal Component')
plt.show()
```



To visualize the separation of clusters, the clusters were projected on the top three principal components on a 3D plot, and the top two principal components on the 2D plot. From the plot below, we can see that the clusters are segregated.

```
In [17]: def plot_knn(kmeans, clusters):
             fig = plt.figure(1)
             ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=45, azim=140)

             for i in range(clusters):
                 ax.scatter(X[kmeans.labels_ == i, 0], X[kmeans.labels_ == i, 1],
                            X[kmeans.labels_ == i, 2], label='Theme ' + str(i))

             ax.set_title('Abstracts plotted on Three Principal Components\n\n')
             ax.set_xlabel('PC1')
             ax.set_ylabel('PC2')
             ax.set_zlabel('PC3')

             ax.legend()
             plt.show()

             fig = plt.figure()

             for i in range(clusters):
                 plt.scatter(X[kmeans.labels_ == i, 0], X[kmeans.labels_ == i, 1],
                            label = 'Theme '+str(i))
                 plt.title('Abstracts plotted on two Principal Components')
                 plt.xlabel('PC1')
                 plt.ylabel('PC2')
                 plt.legend()
             plt.show()

         plot_knn(kmeans, clusters)
```



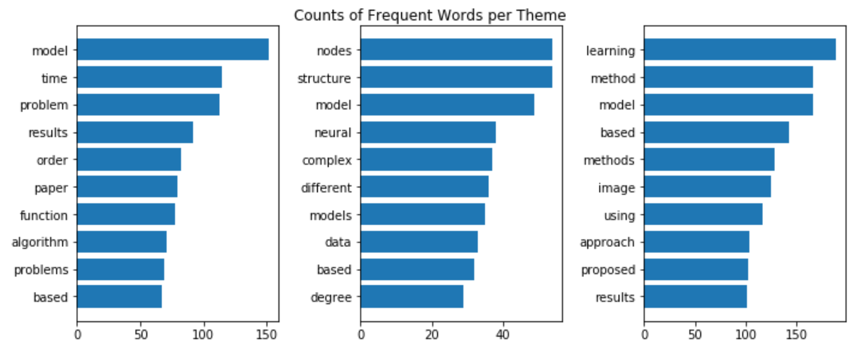


***3 Clusters***

The most important words in the abstracts per theme or cluster are shown below.

```
In [18]: clusters = 3
         kmeans, clustered_abs = clustering(abstracts, clusters);

         # Showing most frequent words per cluster
         df = themes_knn(clustered_abs, clusters)
         display(df.head(10))
```

```
No. of clusters: 3
No. of words: 79
```



| | Theme 0 | Counts 0 | Theme 1 | Counts 1 | Theme 2 | Counts 2 |
|---|---|---|---|---|---|---|
| **0** | model | 152 | nodes | 54 | learning | 189 |
| **1** | time | 115 | structure | 54 | method | 167 |
| **2** | problem | 113 | model | 49 | model | 166 |
| **3** | results | 92 | neural | 38 | based | 143 |
| **4** | order | 82 | complex | 37 | methods | 129 |
| **5** | paper | 79 | different | 36 | image | 125 |
| **6** | function | 78 | models | 35 | using | 117 |
| **7** | algorithm | 71 | data | 33 | approach | 104 |
| **8** | problems | 69 | based | 32 | proposed | 102 |
| **9** | based | 67 | degree | 29 | results | 101 |

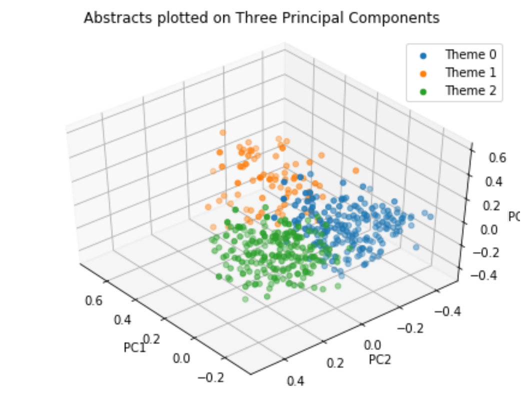
One theme might be about time-related problems, another might be about methods for image-related problems, and the other might be about neural networks. The number of abstracts contained in each theme or cluster are shown below.

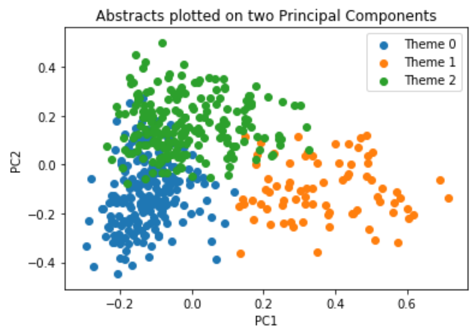
```
In [19]: for i in range(clusters):
             print("Abstracts with Theme {}: {}".format(i,len(clustered_abs[i])))
```

```
Abstracts with Theme 0: 207
Abstracts with Theme 1: 78
Abstracts with Theme 2: 205
```

To visualize the separation of clusters, the clusters were projected on the top three principal components on a 3D plot, and the top two principal components on the 2D plot. We can see that the clusters are separate.

```
In [20]: plot_knn(kmeans, clusters)
```



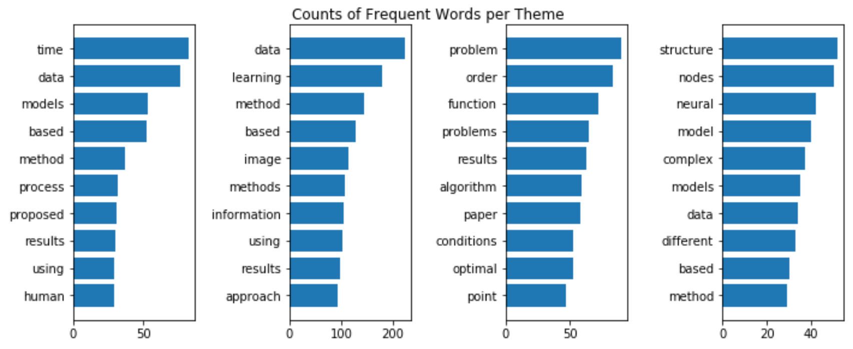


***4 Clusters***

The most important words in the abstracts per theme or cluster are shown below.

```
In [21]: clusters = 4
         kmeans, clustered_abs = clustering(abstracts, clusters);

         # Showing most frequent words per cluster
         df = themes_knn(clustered_abs, clusters)
         display(df.head(10))
```

```
No. of clusters: 4
No. of words: 79
```



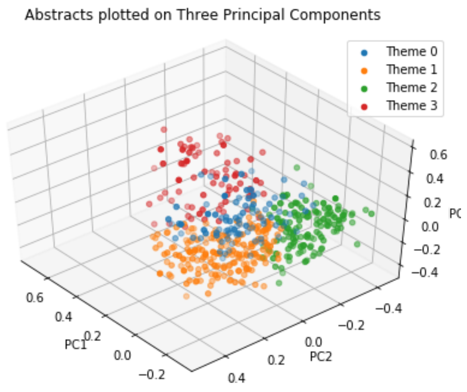
| | Theme 0 | Counts 0 | Theme 1 | Counts 1 | Theme 2 | Counts 2 | Theme 3 | Counts 3 |
|---|---|---|---|---|---|---|---|---|
| **0** | time | 82 | data | 223 | problem | 90.0 | structure | 52 |
| **1** | data | 76 | learning | 179 | order | 83.0 | nodes | 50 |
| **2** | models | 53 | method | 145 | function | 72.0 | neural | 42 |
| **3** | based | 52 | based | 128 | problems | 65.0 | model | 40 |
| **4** | method | 37 | image | 114 | results | 63.0 | complex | 37 |
| **5** | process | 32 | methods | 108 | algorithm | 59.0 | models | 35 |
| **6** | proposed | 31 | information | 104 | paper | 58.0 | data | 34 |
| **7** | results | 30 | using | 103 | conditions | 52.0 | different | 33 |
| **8** | using | 29 | results | 99 | optimal | 52.0 | based | 30 |
| **9** | human | 29 | approach | 94 | point | 47.0 | method | 29 |

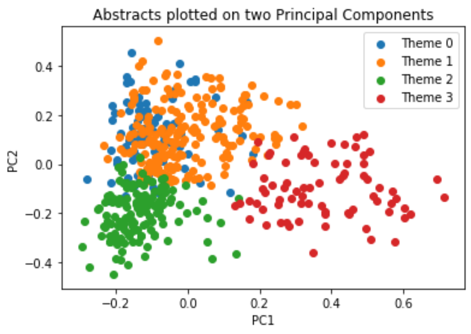
The number of abstracts contained in each theme or cluster are shown below.

```
In [22]: for i in range(clusters):
             print("Abstracts with Theme {}: {}".format(i,len(clustered_abs[i])))

         Abstracts with Theme 0: 88
         Abstracts with Theme 1: 191
         Abstracts with Theme 2: 135
         Abstracts with Theme 3: 76
```

To visualize the separation of clusters, the clusters were projected on the top three principal components on a 3D plot, and the top two principal components on the 2D plot. Immediately, we see that there is an overlap between two of the themes.

```
In [23]: plot_knn(kmeans, clusters)
```



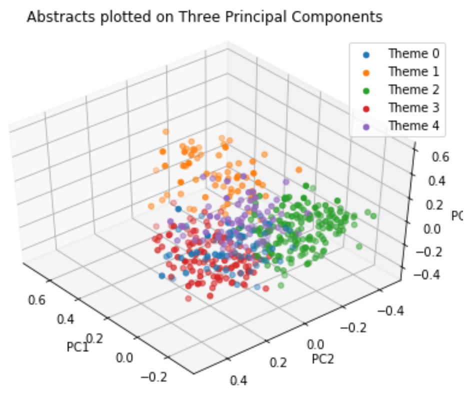


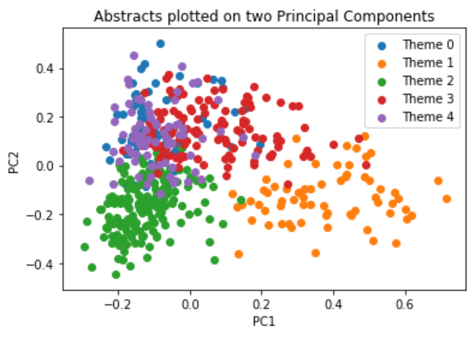
***5 Clusters***

Finally, we plot the data using five clusters. The charts below show multiple overlaps between clusters.

```
In [24]: clusters = 5
         kmeans, clustered_abs = clustering(abstracts, clusters);
         plot_knn(kmeans, clusters)
```

```
No. of clusters: 5
No. of words: 79
```





Based on the plots on the principal components above, clustering into two or three clusters results in well segregated clusters, while clustering into four or five results in multiple overlaps between clusters. Below are the first two sentences in five abstracts belonging to each of the three clusters, using k=3.

In [25]:
```
# First five abstracts in Topics
clusters = 3
kmeans, clustered_abs = clustering(abstracts, clusters);
for i in range(clusters):
    num = 1
    print("Theme", i)
    print("=====================")
    for a in clustered_abs[i][:5]:
        print(num, ".".join(a.split(".")[:2]) + "\n")
        num += 1
```

```
No. of clusters: 3
No. of words: 79
Theme 0
=====================
1 the goal of this paper is to present two algorithms for solving systems of inclusion problems, with all component of the sys
tems being a sum of two maximal monotone operators. the algorithms are variants of the forward-backward splitting method and o
ne being a hybrid with the alternating projection method

2 we provide a compactness criterion for the set of laws $\mathfrak{p}^{ac}_{sem}(\theta)$ on the skorokhod space for which th
e canonical process $x$ is a semimartingale having absolutely continuous characteristics with differential characteristics tak
ing values in some given set $\theta$ of l\'evy triplets. whereas boundedness of $\theta$ implies tightness of $\mathfrak{p}^
{ac}_{sem}(\theta)$, closedness fails in general, even when choosing $\theta$ to be additionally closed and convex, as a seque
nce of purely discontinuous martingales may converge to a diffusion

3 we consider the minimization of a cost function $f$ on a manifold $m$ using riemannian gradient descent and riemannian trust
regions (rtr). we focus on satisfying necessary optimality conditions within a tolerance $\varepsilon$

4 this paper considers optimization over multiple renewal systems coupled by time average constraints. these systems act async
hronously over variable length frames

5 we construct subgame-perfect equilibria with mixed strategies for symmetric stochastic timing games with arbitrary strategic
incentives. the strategies are qualitatively different for local first- or second-mover advantages, which we analyse in turn

Theme 1
=====================
1 we introduce a family of unsupervised domain-free and (asymptotically) model-independent algorithms based on algorithmic inf
ormation theory designed to minimize the loss of any (enumerable computable) property contributing to the object's algorithmic
content and thus important to preserve in the process of data dimension reduction when forcing the algorithm to delete the lea
st important features. being independent of any particular criterion and of general purpose, they are universal in a fundament
al mathematical sense

2 finding graph indices which are unbiased to network size is of high importance both within a given field and across fields f
or enhancing comparability over the cornucopia of modern network science studies. this is also important within studies for co
mparability of subnetworks within a given network, for example

3 we propose a statistical mechanics approach to a coevolving spin system with an adaptive network of interactions. the dynami
cs of node states and network connections is driven by both spin configuration and network topology

4 in the domain of sequence modelling, recurrent neural networks (rnn) have been capable of achieving impressive results in a
variety of application areas including visual question answering, part-of-speech tagging and machine translation. however this
success in modelling short term dependencies has not successfully transitioned to application areas such as trajectory predict
ion, which require capturing both short term and long term relationships

5 the emergence of large-scale connectivity underlies the proper functioning of many networked systems, ranging from social ne
tworks and technological infrastructure to global trade networks. percolation theory characterizes network formation following
stochastic local rules, while optimization models of network formation assume a single controlling authority or one global obj
ective function

Theme 2
=====================
1 some recent works revealed that deep neural networks (dnns) are vulnerable to so-called adversarial attacks where input exam
ples are intentionally perturbed to fool dnns. in this work, we revisit the dnn training process that includes adversarial exa
mples into the training dataset so as to improve dnn's resilience to adversarial attacks, namely, adversarial training

2 several experiments in high-energy physics and astrophysics can be treated as on/off measurements, where an observation pote
ntially containing a new source or effect ("on" measurement) is contrasted with a background-only observation free of the effe
ct ("off" measurement). in counting experiments, the significance of the new source or effect can be estimated with a widely-u
sed formula from [lima], which assumes that both measurements are poisson random variables

3 modeling and verifying real-world cyber-physical systems is challenging, which is especially so for complex systems where ma
nually modeling is infeasible. in this work, we report our experience on combining model learning and abstraction refinement t
o analyze a challenging system, i

4 we present a probabilistic model of an intrusion in a renewal process. given a process and a sequence of events, an intrusio
n is a subsequence of events that is not produced by the process

5 learning tasks on source code (i.e
```

## Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is an algorithm that can be used to extract themes from a set of documents. LDA can be used to calculate the probability that a document belongs to a particular group given the set of words it contains. Unlike k Means classifier, however, LDA considers the fact that a document may have multiple topics, so boundaries between "clusters" or themes are "fuzzy" and not as clear cut as kNN. Instead LDA gives the probabilities that a document belongs to a certain cluster.

Similar to what was done when clustering using kMeans, abstracts were first converted to TfIdf bag of words. These vectors were then fed into `Scikit-learn`'s LDA package. Again, words occurring in less than 10% of the documents or more than 50% of the documents were dropped. Common english words were also removed.

LDA was run on 3 and 2 clusters. The results are discussed and visualized below.

***3 Clusters***

```
In [26]: def run_lda(clusters, docs):
             vect = TfidfVectorizer(min_df=0.1, max_df=0.5, stop_words="english")
             X = vect.fit_transform(docs)
             lda = LatentDirichletAllocation(
                 n_components=clusters, learning_method='batch', max_iter=100,
                 random_state=42)
             docs_topics = lda.fit_transform(X)

             print("Number of clusters:", lda.components_.shape[0])
             print("Number of words:", lda.components_.shape[1])

             return vect, lda, docs_topics

         clusters = 3
         vect, lda, abstract_topics = run_lda(clusters, abstracts)
```

```
Number of clusters: 3
Number of words: 79
```

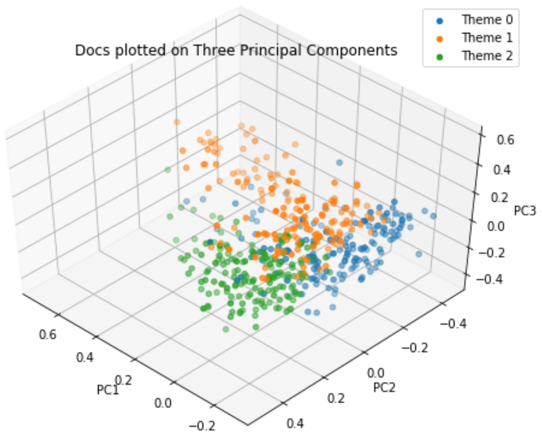
The most important words per cluster or theme are shown in the table below. One of the themes is about networks, another is probably about finding optimal algorithms to solve problems, the other one's theme cannot easily be inferred but it has something to do with functions and order.

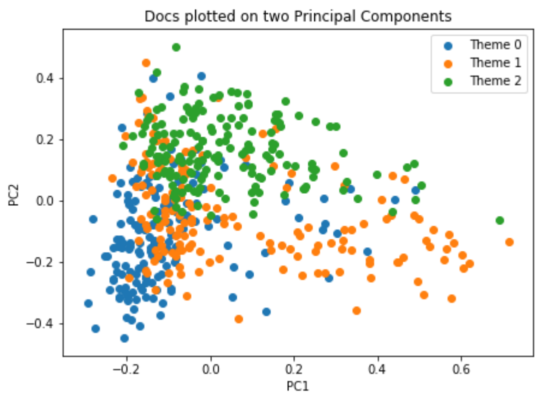
```
In [27]: def themes_lda(vect, lda):
             """
             Imput
             =====
             vect: bag of words vectorizer function (Counts/Tfidf)
             lda: latent Dirichlet Allocation function

             Output
             =====
             Dataframe of most common words per cluster.
             """
             feature_names = np.array(vect.get_feature_names())
             sorts = np.argsort(lda.components_, axis=1)[:, ::-1]
             df_0 = pd.DataFrame()
             for i in range(len(sorts)):
                 s = pd.Series(feature_names[sorts[i]])
                 df_0['Theme ' + str(i)] = s
             return df_0

         display(themes_lda(vect, lda).head(10))
```

| | Theme 0 | Theme 1 | Theme 2 |
|---|---|---|---|
| 0 | problem | model | learning |
| 1 | function | network | image |
| 2 | algorithm | networks | data |
| 3 | algorithms | analysis | method |
| 4 | point | data | approach |
| 5 | problems | time | methods |
| 6 | optimal | systems | based |
| 7 | optimization | information | neural |
| 8 | local | structure | performance |
| 9 | solution | properties | deep |

Principal component analysis was performed to find the top three components that explain the most information (variance). To visualize the separation of clusters, the clusters were projected on the top three principal components on a 3D plot, and the top two principal components on the 2D plot.

```
In [28]: # Cluster labeling
         def plot_lda(docs, docs_topics, clusters):
             labels = [0]*len(docs)
             for c in range(clusters):
                 theme = np.argsort(docs_topics[:, c])[::-1]
                 for i in theme[:int(len(docs)/clusters)]:
                     labels[i] = c

             comps=3
             vect = TfidfVectorizer(min_df=0.1, max_df=0.5, stop_words="english")
             X = vect.fit_transform(docs)
             pca = PCA(n_components=comps)
             pca.fit(X.toarray())
             X = pca.transform(X.toarray())

             fig = plt.figure(1, figsize=(7, 5))
             ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

             for i in range(clusters):
                 inds = [w for w in range(len(labels)) if labels[w] == i]
                 ax.scatter(X[inds, 0], X[inds, 1], X[inds, 2], label='Theme ' + str(i))

                 ax.set_title('Docs plotted on Three Principal Components\n\n')
                 ax.set_xlabel('PC1')
                 ax.set_ylabel('PC2')
                 ax.set_zlabel('PC3')

             ax.legend()
             plt.show()

             fig = plt.figure(1, figsize=(7, 5))

             for i in range(clusters):
                 inds = [w for w in range(len(labels)) if labels[w] == i]
                 color = np.argsort(docs_topics[:, c])[::-1][:len(inds)]
                 #cmaps = ['Blues', 'Oranges', 'Greens']
                 plt.scatter(X[inds, 0], X[inds, 1], label='Theme ' + str(i))
                 plt.title('Docs plotted on two Principal Components')
                 plt.xlabel('PC1')
                 plt.ylabel('PC2')
                 plt.legend()

             plt.show()

         plot_lda(abstracts, abstract_topics, clusters)
```





The charts above show that there is some overlap between two of the themes identified.

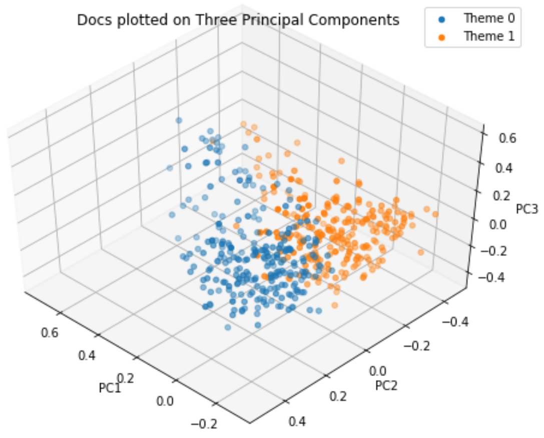
***2 Clusters***

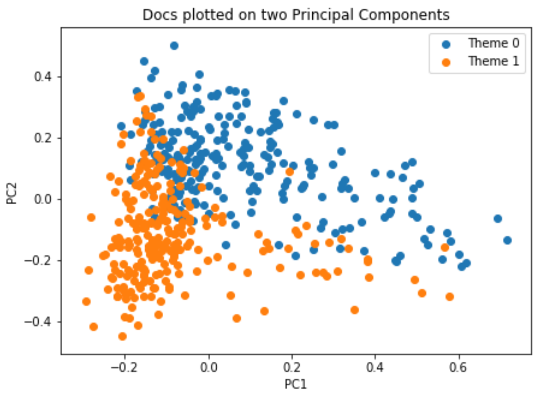
```
In [29]: clusters = 2
         vect, lda, abstract_topics = run_lda(clusters, abstracts)
```

```
Number of clusters: 2
Number of words: 79
```

Running LDA with two clusters produces a clearer boundary between the two clusters as shown in the charts below.

```
In [30]: plot_lda(abstracts, abstract_topics, clusters)
```





Below shows the most important words per theme. One is clearly about networks, while the the other has something to do with time.

```
In [31]: display(themes_lda(vect, lda).head(10))
```

| | Theme 0 | Theme 1 |
|---|---|---|
| **0** | network | problem |
| **1** | networks | time |
| **2** | learning | function |
| **3** | data | model |
| **4** | methods | analysis |
| **5** | models | order |
| **6** | method | algorithm |
| **7** | based | problems |
| **8** | approach | number |
| **9** | model | study |

The first two sentences of the top five abstracts for each of the two clusters are shown below.

In [32]:
```
# First five abstracts in Topics
for i in range(clusters):
    theme = np.argsort(abstract_topics[:, i])[::-1]
    print("Theme", i)
    print("=====================")
    for i in theme[:5]:
        print(i,".".join(abstracts[i].split(".")[:2])+"\n")
```

```
Theme 0
=====================
171 accurate medical image segmentation is essential for diagnosis, surgical planning and many other applications. convolution
al neural networks (cnns) have become the state-of-the-art automatic segmentation methods

327 existing learning-based atmospheric particle-removal approaches such as those used for rainy and hazy images are designed
with strong assumptions regarding spatial frequency, trajectory, and translucency. however, the removal of snow particles is m
ore complicated because it possess the additional attributes of particle size and shape, and these attributes may vary within
a single image

151 spectral clustering (sc) is one of the most widely used methods for data clustering. it first finds a low-dimensonal embed
ding $u$ of data by computing the eigenvectors of the normalized laplacian matrix, and then performs k-means on $u^\top$ to ge
t the final clustering result

166 image captioning has been recently gaining a lot of attention thanks to the impressive achievements shown by deep captioni
ng architectures, which combine convolutional neural networks to extract image representations, and recurrent neural networks
to generate the corresponding captions. at the same time, a significant research effort has been dedicated to the development
of saliency prediction models, which can predict human eye fixations

73 generating emotional language is a key step towards building empathetic natural language processing agents. however, a majo
r challenge for this line of research is the lack of large-scale labeled training data, and previous studies are limited to on
ly small sets of human annotated sentiment labels

Theme 1
=====================
483 we develop a complete analysis of a general entry-exit-scrapping model. in particular, we consider an investment project t
hat operates within a random environment and yields a payoff rate that is a function of a stochastic economic indicator such a
s the price of or the demand for the project's output commodity

162 in this paper we study continuous-time stochastic control problems with both monotone and classical controls motivated by
the so-called public good contribution problem. that is the problem of n economic agents aiming to maximize their expected uti
lity allocating initial wealth over a given time period between private consumption and irreversible contributions to increase
the level of some public good

46 the article continues the study of the 'regular' arrangement of a collection of sets near a point in their intersection. su
ch regular intersection or, in other words, transversality properties are crucial for the validity of qualification conditions
in optimisation as well as subdifferential, normal cone and coderivative calculus, and convergence analysis of computational a
lgorithms

379 temporal inhomogeneities observed in various natural and social phenomena have often been characterized in terms of scalin
g behaviors in the autocorrelation function with a decaying exponent $\gamma$, the interevent time distribution with a power-l
aw exponent $\alpha$, and the burst size distributions. here the interevent time is defined as a time interval between two con
secutive events in the event sequence, and the burst size denotes the number of events in a bursty train detected for a given
time window

256 in several multiobjective decision problems pairwise comparison matrices (pcm) are applied to evaluate the decision varian
ts. the problem that arises very often is the inconsistency of a given pcm
```

## Results

The goal of this paper is to cluster abstracts and find common themes per cluster. Two models were used to cluster abstracts, namely k Means and Latent Dirichlet Allocation (LDA). Principal component analysis was performed to find the three axes or components that capture the most information. The data points, colored per cluster, were then plotted or projected on the top three principal components (3D plot) and top two principal components (2D plot). The most important words per cluster, which may indicate the themes, were also determined.

Based on the charts, k Means clustering was able to form clear boundaries between clusters for two clusters or three clusters. Beyond this, overlaps between clusters form. This means that two or three clusters is preferrable to distinguish the groups clearly. When three clusters were used for LDA however, two clusters clearly overlap. For LDA, two clusters was found to clearly separate abstracts. The table below shows the top words in each theme for each clustering method.

| Method | Clusters | Themes |
|---|---|---|
| k Means | 2 | model, data, method |
| | | learning, data, neural |
| | 3 | time, function, problems |
| | | model, data, learning |
| | | structure, nodes, model |
| LDA | 2 | network, networks, learning |
| | | problem, time, function |

Although k Means was able to generate two or three separate clusters, as projected on the principal components, the clusters are more difficult to interpret and some top words overlap. For k=3 we can infer that one cluster is about time-related problems, but the other two clusters are more vague. LDA, on the other hand was able to identify networks as a clear theme, and the other is about time, similar to what k means identified.

We have shown that both k Means and LDA can be used to cluster technical paper abstracts. The two methods were both able to identify the same theme about problem, time, and function. In general, LDA was able to show more distinguishable themes.

## References and Acknowledgements

Data from AIM Access Lab Team.