Get the Text

In [1]:

```
Text = "Machine learning (ML) is the study of computer algorithms that improve automatically through experience. \
It is seen as a subset of artificial intelligence. \
Machine learning algorithms build a mathematical model based on sample data, known as training data, \
in order to make predictions or decisions without being explicitly programmed to do so. \
Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer visi
on, \
where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks. \
The study of mathematical optimization delivers methods, theory and application domains to the field of machine l
earning. \
Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. \
In its application across business problems, machine learning is also referred to as predictive analytics."
```

Cleaning the data

In [2]:

```
import nltk
from nltk import word_tokenize
import string

#nltk.download('punkt')

def clean(text):
    text = text.lower()
    printable = set(string.printable)
    text = filter(lambda x: x in printable, text)
    text = "".join(list(text))
    return text

Cleaned_text = clean(Text)
print(Cleaned_text)
text = word_tokenize(Cleaned_text)

print ("Tokenized Text: \n")
print (text)
```

```
machine learning (ml) is the study of computer algorithms that improve automatically through experie
nce. it is seen as a subset of artificial intelligence. machine learning algorithms build a mathemat
ical model based on sample data, known as training data, in order to make predictions or decisions w
ithout being explicitly programmed to do so. machine learning algorithms are used in a wide variety
of applications, such as email filtering and computer vision, where it is difficult or infeasible to
develop conventional algorithms to perform the needed tasks. the study of mathematical optimization
delivers methods, theory and application domains to the field of machine learning. data mining is a
related field of study, focusing on exploratory data analysis through unsupervised learning. in its
application across business problems, machine learning is also referred to as predictive analytics.
Tokenized Text:

['machine', 'learning', '(', 'ml', ')', 'is', 'the', 'study', 'of', 'computer', 'algorithms', 'that'
, 'improve', 'automatically', 'through', 'experience', '.', 'it', 'is', 'seen', 'as', 'a', 'subset',
'of', 'artificial', 'intelligence', '.', 'machine', 'learning', 'algorithms', 'build', 'a', 'mathema
tical', 'model', 'based', 'on', 'sample', 'data', ',', 'known', 'as', 'training', 'data', ',', 'in',
'order', 'to', 'make', 'predictions', 'or', 'decisions', 'without', 'being', 'explicitly', 'programm
ed', 'to', 'do', 'so', '.', 'machine', 'learning', 'algorithms', 'are', 'used', 'in', 'a', 'wide', '
variety', 'of', 'applications', ',', 'such', 'as', 'email', 'filtering', 'and', 'computer', 'vision'
, ',', 'where', 'it', 'is', 'difficult', 'or', 'infeasible', 'to', 'develop', 'conventional', 'algor
ithms', 'to', 'perform', 'the', 'needed', 'tasks', '.', 'the', 'study', 'of', 'mathematical', 'optim
ization', 'delivers', 'methods', ',', 'theory', 'and', 'application', 'domains', 'to', 'the', 'field
', 'of', 'machine', 'learning', '.', 'data', 'mining', 'is', 'a', 'related', 'field', 'of', 'study',
',', 'focusing', 'on', 'exploratory', 'data', 'analysis', 'through', 'unsupervised', 'learning', '.'
, 'in', 'its', 'application', 'across', 'business', 'problems', ',', 'machine', 'learning', 'is', 'a
lso', 'referred', 'to', 'as', 'predictive', 'analytics', '.']
```

POS Tagging (useful for lemmatization)

In [3]:

```
nltk.download('averaged_perceptron_tagger')

POS_tag = nltk.pos_tag(text)

print ("Tokenized Text with POS tags: \n")
print (POS_tag)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\kruti\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Tokenized Text with POS tags:

```
[('machine', 'NN'), ('learning', 'NN'), ('(', '('), ('ml', 'NN'), (')', ')'), ('is', 'VBZ'), ('the',
'DT'), ('study', 'NN'), ('of', 'IN'), ('computer', 'NN'), ('algorithms', 'NN'), ('that', 'WDT'), ('i
mprove', 'VBP'), ('automatically', 'RB'), ('through', 'IN'), ('experience', 'NN'), ('.', '.'), ('it'
, 'PRP'), ('is', 'VBZ'), ('seen', 'VBN'), ('as', 'IN'), ('a', 'DT'), ('subset', 'NN'), ('of', 'IN'),
('artificial', 'JJ'), ('intelligence', 'NN'), ('.', '.'), ('machine', 'NN'), ('learning', 'VBG'), ('
algorithms', 'JJ'), ('build', 'VB'), ('a', 'DT'), ('mathematical', 'JJ'), ('model', 'NN'), ('based',
'VBN'), ('on', 'IN'), ('sample', 'NN'), ('data', 'NNS'), (',', ','), ('known', 'VBN'), ('as', 'IN'),
('training', 'NN'), ('data', 'NNS'), (',', ','), ('in', 'IN'), ('order', 'NN'), ('to', 'TO'), ('make
', 'VB'), ('predictions', 'NNS'), ('or', 'CC'), ('decisions', 'NNS'), ('without', 'IN'), ('being', '
VBG'), ('explicitly', 'RB'), ('programmed', 'VBN'), ('to', 'TO'), ('do', 'VB'), ('so', 'RB'), ('.',
'.'), ('machine', 'NN'), ('learning', 'VBG'), ('algorithms', 'NNS'), ('are', 'VBP'), ('used', 'VBN')
, ('in', 'IN'), ('a', 'DT'), ('wide', 'JJ'), ('variety', 'NN'), ('of', 'IN'), ('applications', 'NNS'
), (',', ','), ('such', 'JJ'), ('as', 'IN'), ('email', 'NN'), ('filtering', 'NN'), ('and', 'CC'), ('
computer', 'NN'), ('vision', 'NN'), (',', ','), ('where', 'WRB'), ('it', 'PRP'), ('is', 'VBZ'), ('di
fficult', 'JJ'), ('or', 'CC'), ('infeasible', 'JJ'), ('to', 'TO'), ('develop', 'VB'), ('conventional
', 'JJ'), ('algorithms', 'NNS'), ('to', 'TO'), ('perform', 'VB'), ('the', 'DT'), ('needed', 'JJ'), (
'tasks', 'NNS'), ('.', '.'), ('the', 'DT'), ('study', 'NN'), ('of', 'IN'), ('mathematical', 'JJ'), (
'optimization', 'NN'), ('delivers', 'NNS'), ('methods', 'NNS'), (',', ','), ('theory', 'NN'), ('and'
, 'CC'), ('application', 'NN'), ('domains', 'NNS'), ('to', 'TO'), ('the', 'DT'), ('field', 'NN'), ('
of', 'IN'), ('machine', 'NN'), ('learning', 'NN'), ('.', '.'), ('data', 'NNS'), ('mining', 'NN'), ('
is', 'VBZ'), ('a', 'DT'), ('related', 'JJ'), ('field', 'NN'), ('of', 'IN'), ('study', 'NN'), (',', '
,'), ('focusing', 'VBG'), ('on', 'IN'), ('exploratory', 'NN'), ('data', 'NNS'), ('analysis', 'NN'),
('through', 'IN'), ('unsupervised', 'JJ'), ('learning', 'NN'), ('.', '.'), ('in', 'IN'), ('its', 'PR
P$'), ('application', 'NN'), ('across', 'IN'), ('business', 'NN'), ('problems', 'NNS'), (',', ','),
('machine', 'NN'), ('learning', 'NN'), ('is', 'VBZ'), ('also', 'RB'), ('referred', 'VBN'), ('to', 'T
O'), ('as', 'IN'), ('predictive', 'JJ'), ('analytics', 'NNS'), ('.', '.')]
```

Lemmatization

In [4]:

```
#nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer

wordnet_lemmatizer = WordNetLemmatizer()

adjective_tags = ['JJ','JJR','JJS']

lemmatized_text = []

for word in POS_tag:
    if word[1] in adjective_tags:
        lemmatized_text.append(str(wordnet_lemmatizer.lemmatize(word[0],pos="a")))
    else:
        lemmatized_text.append(str(wordnet_lemmatizer.lemmatize(word[0]))) #default POS = noun

print ("Text tokens after lemmatization of adjectives and nouns: \n")
print (lemmatized_text)
```

```
Text tokens after lemmatization of adjectives and nouns:

['machine', 'learning', '(', 'ml', ')', 'is', 'the', 'study', 'of', 'computer', 'algorithm', 'that',
'improve', 'automatically', 'through', 'experience', '.', 'it', 'is', 'seen', 'a', 'a', 'subset', 'o
f', 'artificial', 'intelligence', '.', 'machine', 'learning', 'algorithms', 'build', 'a', 'mathemati
cal', 'model', 'based', 'on', 'sample', 'data', ',', 'known', 'a', 'training', 'data', ',', 'in', 'o
rder', 'to', 'make', 'prediction', 'or', 'decision', 'without', 'being', 'explicitly', 'programmed',
'to', 'do', 'so', '.', 'machine', 'learning', 'algorithm', 'are', 'used', 'in', 'a', 'wide', 'variet
y', 'of', 'application', ',', 'such', 'a', 'email', 'filtering', 'and', 'computer', 'vision', ',', '
where', 'it', 'is', 'difficult', 'or', 'infeasible', 'to', 'develop', 'conventional', 'algorithm', '
to', 'perform', 'the', 'needed', 'task', '.', 'the', 'study', 'of', 'mathematical', 'optimization',
'delivers', 'method', ',', 'theory', 'and', 'application', 'domain', 'to', 'the', 'field', 'of', 'ma
chine', 'learning', '.', 'data', 'mining', 'is', 'a', 'related', 'field', 'of', 'study', ',', 'focus
ing', 'on', 'exploratory', 'data', 'analysis', 'through', 'unsupervised', 'learning', '.', 'in', 'it
', 'application', 'across', 'business', 'problem', ',', 'machine', 'learning', 'is', 'also', 'referr
ed', 'to', 'a', 'predictive', 'analytics', '.']
```

POS Tagging for filtering

In [5]:

```
POS_tag = nltk.pos_tag(lemmatized_text)
print ("Lemmatized text with POS tags: \n")
print (POS_tag)
```

```
Lemmatized text with POS tags:

[('machine', 'NN'), ('learning', 'NN'), ('(', '('), ('ml', 'NN'), (')', ')'), ('is', 'VBZ'), ('the',
'DT'), ('study', 'NN'), ('of', 'IN'), ('computer', 'NN'), ('algorithm', 'NN'), ('that', 'WDT'), ('im
prove', 'VBP'), ('automatically', 'RB'), ('through', 'IN'), ('experience', 'NN'), ('.', '.'), ('it',
'PRP'), ('is', 'VBZ'), ('seen', 'VBN'), ('a', 'DT'), ('a', 'DT'), ('subset', 'NN'), ('of', 'IN'), ('
artificial', 'JJ'), ('intelligence', 'NN'), ('.', '.'), ('machine', 'NN'), ('learning', 'VBG'), ('al
gorithms', 'JJ'), ('build', 'VB'), ('a', 'DT'), ('mathematical', 'JJ'), ('model', 'NN'), ('based', '
VBN'), ('on', 'IN'), ('sample', 'NN'), ('data', 'NNS'), (',', ','), ('known', 'VBN'), ('a', 'DT'), (
'training', 'NN'), ('data', 'NNS'), (',', ','), ('in', 'IN'), ('order', 'NN'), ('to', 'TO'), ('make'
, 'VB'), ('prediction', 'NN'), ('or', 'CC'), ('decision', 'NN'), ('without', 'IN'), ('being', 'VBG')
, ('explicitly', 'RB'), ('programmed', 'VBN'), ('to', 'TO'), ('do', 'VB'), ('so', 'RB'), ('.', '.'),
('machine', 'NN'), ('learning', 'VBG'), ('algorithm', 'NNS'), ('are', 'VBP'), ('used', 'VBN'), ('in'
, 'IN'), ('a', 'DT'), ('wide', 'JJ'), ('variety', 'NN'), ('of', 'IN'), ('application', 'NN'), (',',
',',), ('such', 'PDT'), ('a', 'DT'), ('email', 'NN'), ('filtering', 'NN'), ('and', 'CC'), ('computer'
, 'NN'), ('vision', 'NN'), (',', ','), ('where', 'WRB'), ('it', 'PRP'), ('is', 'VBZ'), ('difficult',
'JJ'), ('or', 'CC'), ('infeasible', 'JJ'), ('to', 'TO'), ('develop', 'VB'), ('conventional', 'JJ'),
('algorithm', 'NNS'), ('to', 'TO'), ('perform', 'VB'), ('the', 'DT'), ('needed', 'JJ'), ('task', 'NN
'), ('.', '.'), ('the', 'DT'), ('study', 'NN'), ('of', 'IN'), ('mathematical', 'JJ'), ('optimization
', 'NN'), ('delivers', 'NNS'), ('method', 'VBP'), (',', ','), ('theory', 'NN'), ('and', 'CC'), ('app
lication', 'NN'), ('domain', 'NN'), ('to', 'TO'), ('the', 'DT'), ('field', 'NN'), ('of', 'IN'), ('ma
chine', 'NN'), ('learning', 'NN'), ('.', '.'), ('data', 'NNS'), ('mining', 'NN'), ('is', 'VBZ'), ('a
', 'DT'), ('related', 'JJ'), ('field', 'NN'), ('of', 'IN'), ('study', 'NN'), (',', ','), ('focusing'
, 'VBG'), ('on', 'IN'), ('exploratory', 'NN'), ('data', 'NNS'), ('analysis', 'NN'), ('through', 'IN'
), ('unsupervised', 'JJ'), ('learning', 'NN'), ('.', '.'), ('in', 'IN'), ('it', 'PRP'), ('applicatio
n', 'NN'), ('across', 'IN'), ('business', 'NN'), ('problem', 'NN'), (',', ','), ('machine', 'NN'), (
'learning', 'NN'), ('is', 'VBZ'), ('also', 'RB'), ('referred', 'VBN'), ('to', 'TO'), ('a', 'DT'), ('
predictive', 'JJ'), ('analytics', 'NNS'), ('.', '.')]
```

Getting rid of unwanted stopwords

In [6]:

```
stopwords = []

wanted_POS = ['NN','NNS','NNP','NNPS','JJ','JJR','JJS','VBG','FW']

for word in POS_tag:
    if word[1] not in wanted_POS:
        stopwords.append(word[0])

punctuations = list(str(string.punctuation))

stopwords = stopwords + punctuations
```

In [7]:

```
stopword_file = open("long_stopwords.txt", "r")
#Source = https://www.ranks.nl/stopwords

lots_of_stopwords = []

for line in stopword_file.readlines():
    lots_of_stopwords.append(str(line.strip()))

stopwords_plus = []
stopwords_plus = stopwords + lots_of_stopwords
stopwords_plus = set(stopwords_plus)

#Stopwords_plus contain total set of all stopwords
```

Removing Stopwords from Lemmatized text

In [8]:

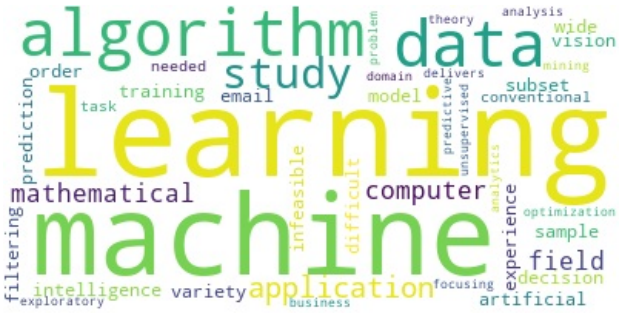
```
processed_text = []
for word in lemmatized_text:
    if word not in stopwords_plus:
        processed_text.append(word)
print (processed_text)
```

```
['machine', 'learning', 'study', 'computer', 'algorithm', 'experience', 'subset', 'artificial', 'int
elligence', 'machine', 'learning', 'algorithms', 'mathematical', 'model', 'sample', 'data', 'trainin
g', 'data', 'order', 'prediction', 'decision', 'machine', 'learning', 'algorithm', 'wide', 'variety'
, 'application', 'email', 'filtering', 'computer', 'vision', 'difficult', 'infeasible', 'conventiona
l', 'algorithm', 'needed', 'task', 'study', 'mathematical', 'optimization', 'delivers', 'theory', 'a
pplication', 'domain', 'field', 'machine', 'learning', 'data', 'mining', 'field', 'study', 'focusing
', 'exploratory', 'data', 'analysis', 'unsupervised', 'learning', 'application', 'business', 'proble
m', 'machine', 'learning', 'predictive', 'analytics']
```

In [9]:

```
# I am gonna import the word cloud library
from wordcloud import WordCloud

# for creating a word cloud we need to pull all our text together
# I am gonna do that by joining all the processed_data together in a single variable
para = ','.join(list(processed_text))

# Lets create an instance for word cloud
wordcloud = WordCloud(background_color = 'white',
                      max_words = 5000,
                      contour_width = 3,
                      contour_color = 'steelblue')

# generate our word cloud
wordcloud.generate(para)

# Lets visualize it
wordcloud.to_image()
```

Out[9]:



Vocabulary creation

In [10]:

```
vocabulary = list(set(processed_text))
print (vocabulary)
```

```
['mathematical', 'learning', 'email', 'infeasible', 'theory', 'predictive', 'problem', 'data', 'stud
y', 'intelligence', 'model', 'order', 'prediction', 'computer', 'decision', 'optimization', 'variety
', 'task', 'algorithms', 'exploratory', 'business', 'field', 'algorithm', 'unsupervised', 'focusing'
, 'subset', 'artificial', 'analysis', 'filtering', 'difficult', 'analytics', 'mining', 'needed', 'de
livers', 'machine', 'conventional', 'training', 'vision', 'experience', 'sample', 'domain', 'applica
tion', 'wide']
```

Building a Graph

In [11]:

```
import numpy as np
import math
vocab_len = len(vocabulary)

weighted_edge = np.zeros((vocab_len,vocab_len),dtype=np.float32)

score = np.zeros((vocab_len),dtype=np.float32)
window_size = 3
covered_coocurrences = []

for i in range(0,vocab_len):
    score[i]=1
    for j in range(0,vocab_len):
        if j==i:
            weighted_edge[i][j]=0
        else:
            for window_start in range(0,(len(processed_text)-window_size)):

                window_end = window_start+window_size

                window = processed_text[window_start:window_end]

                if (vocabulary[i] in window) and (vocabulary[j] in window):

                    index_of_i = window_start + window.index(vocabulary[i])
                    index_of_j = window_start + window.index(vocabulary[j])

                    # index_of_x is the absolute position of the xth term in the window
                    # (counting from 0)
                    # in the processed_text

                    if [index_of_i,index_of_j] not in covered_coocurrences:
                        weighted_edge[i][j]+=1/math.fabs(index_of_i-index_of_j)
                        covered_coocurrences.append([index_of_i,index_of_j])
```

Calculated weighted summation

In [12]:

```
inout = np.zeros((vocab_len),dtype=np.float32)

for i in range(0,vocab_len):
    for j in range(0,vocab_len):
        inout[i]+=weighted_edge[i][j]
```

Scoring Vertices

In [13]:

```
MAX_ITERATIONS = 50
d=0.85
threshold = 0.0001 #convergence threshold

for iter in range(0,MAX_ITERATIONS):
    prev_score = np.copy(score)

    for i in range(0,vocab_len):

        summation = 0
        for j in range(0,vocab_len):
            if weighted_edge[i][j] != 0:
                summation += (weighted_edge[i][j]/inout[j])*score[j]

        score[i] = (1-d) + d*(summation)

    if np.sum(np.fabs(prev_score-score)) <= threshold: #convergence condition
        print("Converging at iteration "+str(iter)+"....")
        break
```

```
Converging at iteration 30....
```

In [14]:

```
for i in range(0,vocab_len):
    print("Score of "+vocabulary[i]+": "+str(score[i]))
```

```
Score of mathematical: 1.3414217
Score of learning: 3.3069468
Score of email: 0.7507528
Score of infeasible: 0.80758595
Score of theory: 0.75666744
Score of predictive: 0.39872605
Score of problem: 0.68849766
Score of data: 2.3142533
Score of study: 1.8658237
Score of intelligence: 0.72746646
Score of model: 0.73550504
Score of order: 0.73915064
Score of prediction: 0.73398274
Score of computer: 1.3645318
Score of decision: 0.7121059
Score of optimization: 0.75195575
Score of variety: 0.73689526
Score of task: 0.7234718
Score of algorithms: 0.69295734
Score of exploratory: 0.7181293
Score of business: 0.6952718
Score of field: 1.247756
Score of algorithm: 1.9629959
Score of unsupervised: 0.6983106
Score of focusing: 0.7074837
Score of subset: 0.779203
Score of artificial: 0.7681685
Score of analysis: 0.711092
Score of filtering: 0.76079714
Score of difficult: 0.81194323
Score of analytics: 0.15
Score of mining: 0.6763761
Score of needed: 0.7395537
Score of delivers: 0.76453316
Score of machine: 2.648495
Score of conventional: 0.78400546
Score of training: 0.7163138
Score of vision: 0.7955469
Score of experience: 0.76164645
Score of sample: 0.73371696
Score of domain: 0.7014836
Score of application: 1.949768
Score of wide: 0.7189271
```

Phrase Partioning

In [15]:

```
phrases = []

phrase = " "
for word in lemmatized_text:

    if word in stopwords_plus:
        if phrase!= " ":
            phrases.append(str(phrase).strip().split())
        phrase = " "
    elif word not in stopwords_plus:
        phrase+=str(word)
        phrase+=" "

print("Partitioned Phrases (Candidate Keyphrases): \n")
print(phrases)
```

```
Partitioned Phrases (Candidate Keyphrases):

[['machine', 'learning'], ['study'], ['computer', 'algorithm'], ['experience'], ['subset'], ['artifi
cial', 'intelligence'], ['machine', 'learning', 'algorithms'], ['mathematical', 'model'], ['sample',
'data'], ['training', 'data'], ['order'], ['prediction'], ['decision'], ['machine', 'learning', 'alg
orithm'], ['wide', 'variety'], ['application'], ['email', 'filtering'], ['computer', 'vision'], ['di
fficult'], ['infeasible'], ['conventional', 'algorithm'], ['needed', 'task'], ['study'], ['mathemati
cal', 'optimization', 'delivers'], ['theory'], ['application', 'domain'], ['field'], ['machine', 'le
arning'], ['data', 'mining'], ['field'], ['study'], ['focusing'], ['exploratory', 'data', 'analysis'
], ['unsupervised', 'learning'], ['application'], ['business', 'problem'], ['machine', 'learning'],
['predictive', 'analytics']]
```

In [16]:

```
unique_phrases = []

for phrase in phrases:
    if phrase not in unique_phrases:
        unique_phrases.append(phrase)

print("Unique Phrases (Candidate Keyphrases): \n")
print(unique_phrases)
```

```
Unique Phrases (Candidate Keyphrases):

[['machine', 'learning'], ['study'], ['computer', 'algorithm'], ['experience'], ['subset'], ['artifi
cial', 'intelligence'], ['machine', 'learning', 'algorithms'], ['mathematical', 'model'], ['sample',
'data'], ['training', 'data'], ['order'], ['prediction'], ['decision'], ['machine', 'learning', 'alg
orithm'], ['wide', 'variety'], ['application'], ['email', 'filtering'], ['computer', 'vision'], ['di
fficult'], ['infeasible'], ['conventional', 'algorithm'], ['needed', 'task'], ['mathematical', 'opti
mization', 'delivers'], ['theory'], ['application', 'domain'], ['field'], ['data', 'mining'], ['focu
sing'], ['exploratory', 'data', 'analysis'], ['unsupervised', 'learning'], ['business', 'problem'],
['predictive', 'analytics']]
```

Multiword KeyPhrases

In [17]:

```
for word in vocabulary:
    #print word
    for phrase in unique_phrases:
        if (word in phrase) and ([word] in unique_phrases) and (len(phrase)>1):
            #if len(phrase)>1 then the current phrase is multi-worded.
            #if the word in vocabulary is present in unique_phrases as a single-word-phrase
            # and at the same time present as a word within a multi-worded phrase,
            # then I will remove the single-word-phrase from the list.
            unique_phrases.remove([word])

print("Thinned Unique Phrases (Candidate Keyphrases): \n")
print(unique_phrases)
```

```
Thinned Unique Phrases (Candidate Keyphrases):

[['machine', 'learning'], ['study'], ['computer', 'algorithm'], ['experience'], ['subset'], ['artifi
cial', 'intelligence'], ['machine', 'learning', 'algorithms'], ['mathematical', 'model'], ['sample',
'data'], ['training', 'data'], ['order'], ['prediction'], ['decision'], ['machine', 'learning', 'alg
orithm'], ['wide', 'variety'], ['email', 'filtering'], ['computer', 'vision'], ['difficult'], ['infe
asible'], ['conventional', 'algorithm'], ['needed', 'task'], ['mathematical', 'optimization', 'deliv
ers'], ['theory'], ['application', 'domain'], ['field'], ['data', 'mining'], ['focusing'], ['explora
tory', 'data', 'analysis'], ['unsupervised', 'learning'], ['business', 'problem'], ['predictive', 'a
nalytics']]
```

Scoring Key phrases

In [18]:

```
phrase_scores = []
keywords = []
for phrase in unique_phrases:
    phrase_score=0
    keyword = ''
    for word in phrase:
        keyword += str(word)
        keyword += " "
        phrase_score+=score[vocabulary.index(word)]
    phrase_scores.append(phrase_score)
    keywords.append(keyword.strip())

i=0
for keyword in keywords:
    print ("Keyword: '"+str(keyword)+"', Score: "+str(phrase_scores[i]))
    i+=1
```

```
Keyword: 'machine learning', Score: 5.95544171333313
Keyword: 'study', Score: 1.865823745727539
Keyword: 'computer algorithm', Score: 3.327527642250061
Keyword: 'experience', Score: 0.7616464495658875
Keyword: 'subset', Score: 0.7792029976844788
Keyword: 'artificial intelligence', Score: 1.4956349730491638
Keyword: 'machine learning algorithms', Score: 6.64839905500412
Keyword: 'mathematical model', Score: 2.0769267678260803
Keyword: 'sample data', Score: 3.0479702949523926
Keyword: 'training data', Score: 3.0305671095848083
Keyword: 'order', Score: 0.7391506433486938
Keyword: 'prediction', Score: 0.7339827418327332
Keyword: 'decision', Score: 0.7121058702468872
Keyword: 'machine learning algorithm', Score: 7.918437600135803
Keyword: 'wide variety', Score: 1.4558223485946655
Keyword: 'email filtering', Score: 1.511549949645996
Keyword: 'computer vision', Score: 2.1600786447525024
Keyword: 'difficult', Score: 0.8119432330131531
Keyword: 'infeasible', Score: 0.8075859546661377
Keyword: 'conventional algorithm', Score: 2.747001349925995
Keyword: 'needed task', Score: 1.4630255103111267
Keyword: 'mathematical optimization delivers', Score: 2.857910633087158
Keyword: 'theory', Score: 0.75666743516922
Keyword: 'application domain', Score: 2.6512515544891357
Keyword: 'field', Score: 1.247756004333496
Keyword: 'data mining', Score: 2.9906294345855713
Keyword: 'focusing', Score: 0.70748370885849
Keyword: 'exploratory data analysis', Score: 3.74347460269928
Keyword: 'unsupervised learning', Score: 4.0052573680877686
Keyword: 'business problem', Score: 1.383769452571869
Keyword: 'predictive analytics', Score: 0.5487260520458221
```