Hong Kong Baptist University

Department of Computer Science

COMP 4075 Social Computing and Web Intelligence/

COMP 7630 Web Intelligence and Its Applications

Semester 2, 2019-20

Lab 2 Introduction to Scikit Learn for Web Mining

Objective:

This lab aims to show students how to use the Scikit-learn package for classification and clustering.

Preparation:

We will use sklearn, numpy, nltk and gensim packages in this lab and thus we need to install them. Go to Anaconda prompt and install the packages first, e.g.,

pip install sklearn

1 *Existing dataset from scikit-learn package*

1.1 *About scikit-learn package*[1] [2]

Scikit-learn is a free, simple and efficient software machine learning library in Python. It supports various classification, regression and clustering algorithms.

1.2 *Importing the 20 newsgroups text dataset*[3]

Note: Sample Program for this task: ***lab2_20newsgroups.py***

Scikit-Learn comes with the 20 newsgroup text dataset that comprises around 18,000 newsgroups posts on 20 topics and is split into the training set and the test set. We can use the dataset by importing the dataset directly from the library.

1 https://en.wikipedia.org/wiki/Scikit-learn

2 http://scikit-learn.org/stable/index.html

3 http://qwone.com/~jason/20Newsgroups/

```
from sklearn.datasets import fetch_20newsgroups
```

By using function *fetch_20newsgroups()* [4], we can import the data. We can change the parameters ***subset*** and ***categories*** for accessing different subsets of the data.

For example:

```
newsgroups_data = fetch_20newsgroups(subset='all')
print('newsgroups_data.target_names: ')
print(newsgroups_data.target_names)
```

Output:

```
list(newsgroups_data.target_names):
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
'comp.windows.x', 'misc.forsale', 'rec.autos',
'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey',
'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space',
'soc.religion.christian', 'talk.politics.guns',
'talk.politics.mideast', 'talk.politics.misc',
'talk.religion.misc']
```

The above output shows all the category labels in the dataset. The structure of newsgroup_data is as follows:

	newsgroups_data		
	data	*...*	*target*
0	*From: Mamatha Devineni Ratnam … PENS RULE!!!*	*…*	*10*
1	*From: mblawson@midway.ec …562 B.C. |*	*…*	*3*
2	*From: hilmi-er@dsv.su.se …Stockholm University*	*…*	*17*

where “data” is the content and “target” is the index referring to *newsgroups_data.target_names,* i.e. the target category label.

Therefore, we can access the data details by referring to *newsgroups_data.data and newsgroups_data.target*:

```
print('Size of newsgroups_data.data: %d' % len(newsgroups_data.data))
```

[4] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_20newsgroups.html

```
for i in range(3):
    print('Doc Number %d' % i)
    print('Doc Type: %s' %
newsgroups_data.target_names[newsgroups_data.target[i]])
    print(newsgroups_data.data[i])
    print('')
```

The document details:

```
Size of newsgroups_data.data: 18846
Doc Number 0
Doc Type: rec.sport.hockey
From: Mamatha Devineni Ratnam <mr47+@andrew.cmu.edu>
Subject: Pens fans reactions
Organization: Post Office, Carnegie Mellon, Pittsburgh, PA
Lines: 12
NNTP-Posting-Host: po4.andrew.cmu.edu



I am sure some bashers of Pens fans are pretty confused about the lack
of any kind of posts about the recent Pens massacre of the Devils.
Actually,
I am  bit puzzled too and a bit relieved. However, I am going to put an
end
to non-PIttsburghers' relief with a bit of praise for the Pens. Man,
they
are killing those Devils worse than I thought. Jagr just showed you why
he is much better than his regular season stats. He is also a lot
fo fun to watch in the playoffs. Bowman should let JAgr have a lot of
fun in the next couple of games since the Pens are going to beat the
pulp out of Jersey anyway. I was very disappointed not to see the
Islanders lose the final
regular season game.          PENS RULE!!!
```

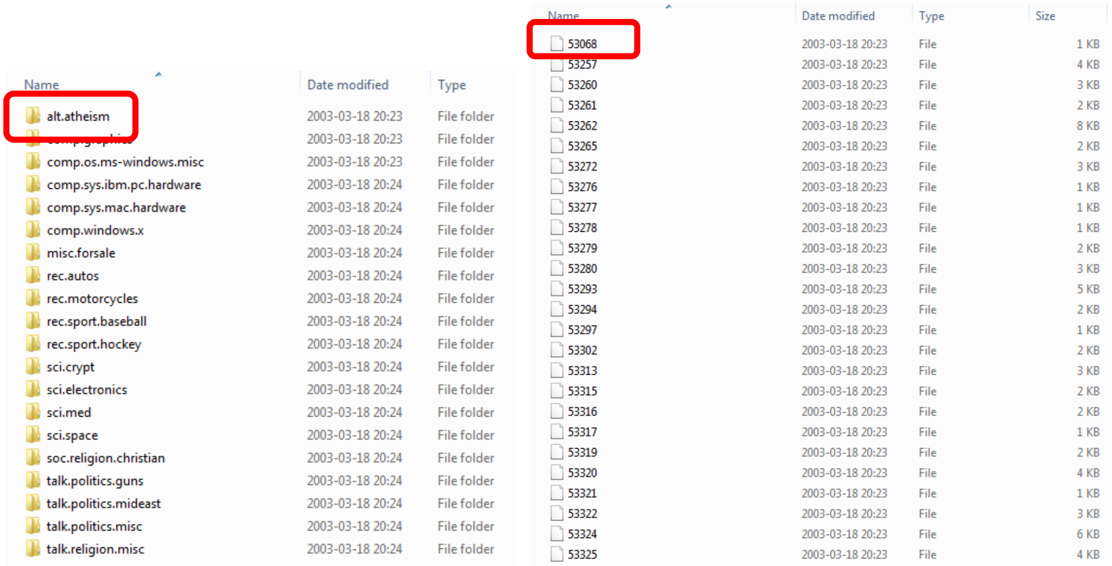
1.3 <u>*Importing data from local files*</u>

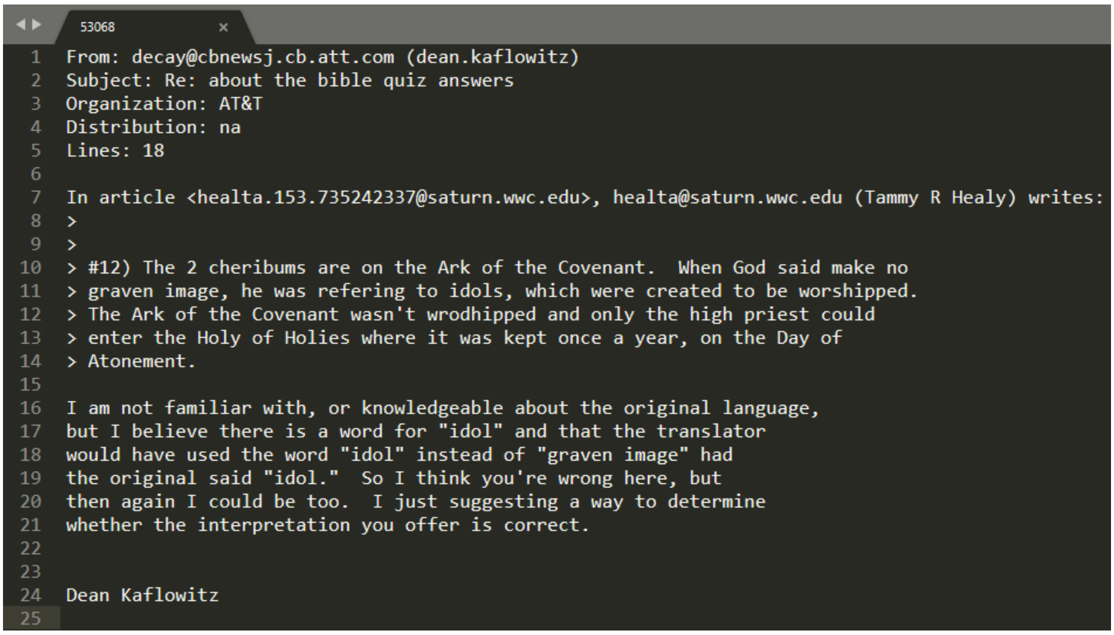
Note: Sample Program for this task: ***lab2_20newsgroups.py, twenty_newsgroups.py***

While it is good that scikit-learn provides some data for our exercise, it is more common that we need to acquire data from online sources, pre-process them, and stored them as files for subsequent data analysis. This section is about how to read data from local files.

You can first download the 20 newsgroups data we prepared from http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz (instead of importing from scikit-learn package). After uncompressing the downloaded file, we can see two folders consisting of training and test data, respectively.

The folders of training and test data both contain 20 subfolders with the subfolder names corresponding to the categories (e.g., alt.atheism). Each folder contains a lot of individual samples in plain text.




53068

```
From: decay@cbnewsj.cb.att.com (dean.kaflowitz)
Subject: Re: about the bible quiz answers
Organization: AT&T
Distribution: na
Lines: 18

In article <healta.153.735242337@saturn.wwc.edu>, healta@saturn.wwc.edu (Tammy R Healy) writes:
>
>
> #12) The 2 cheribums are on the Ark of the Covenant.  When God said make no
> graven image, he was refering to idols, which were created to be worshipped.
> The Ark of the Covenant wasn't wrodhipped and only the high priest could
> enter the Holy of Holies where it was kept once a year, on the Day of
> Atonement.

I am not familiar with, or knowledgeable about the original language,
but I believe there is a word for "idol" and that the translator
would have used the word "idol" instead of "graven image" had
the original said "idol."  So I think you're wrong here, but
then again I could be too.  I just suggesting a way to determine
whether the interpretation you offer is correct.


Dean Kaflowitz
```

You can then run the program ***lab2_load_local_data.py*** as shown below and can display the same results as in section 1.2. It uses a function named ***load_20newsgroups()*** from ***twenty_newsgroups.py*** file to read data from the local files.

Notice that there are two minor differences between the function *load_20newsgroups()* and the function *fetch_20newsgroups() used in the previous section*. First, the parameter ***data_home*** in *load_20newsgroups()* is mandatory and it specifies the main folder holding folders of training and test data. Second, the parameter ***download_if_missing*** is not needed in *load_20newsgroups()* as we do not need to download the data.

```
from twenty_newsgroups import load_20newsgroups

newsgroups_data = load_20newsgroups(data_home='./ ', subset='all')
# list out all the categories name in the dataset
print('newsgroups_data.target_names:')
print(newsgroups_data.target_names)

print('Size of newsgroups_data.data: %d' % len(newsgroups_data.data))
for i in range(3):
        print('Doc Number %d' % i)
        print('Target Index: %d' % newsgroups_data.target[i])
        print('Doc Type: %s' %
newsgroups_data.target_names[newsgroups_data.target[i]])
        print(newsgroups_data.data[i])
        print('')
```

```
newsgroups_data.target_names:
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware',
'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space',
'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc',
'talk.religion.misc']

Size of newsgroups_data.data: 18846
Doc Number 0
Target Index: 10
Doc Type: rec.sport.hockey
From: Mamatha Devineni Ratnam <mr47+@andrew.cmu.edu>
Subject: Pens fans reactions
Organization: Post Office, Carnegie Mellon, Pittsburgh, PA
Lines: 12
NNTP-Posting-Host: po4.andrew.cmu.edu


I am sure some bashers of Pens fans are pretty confused about the lack
of any kind of posts about the recent Pens massacre of the Devils. Actually,
I am  bit puzzled too and a bit relieved. However, I am going to put an end
to non-PIttsburghers' relief with a bit of praise for the Pens. Man, they
are killing those Devils worse than I thought. Jagr just showed you why
he is much better than his regular season stats. He is also a lot
fo fun to watch in the playoffs. Bowman should let JAgr have a lot of
fun in the next couple of games since the Pens are going to beat the pulp out of Jersey anyway. I
was very disappointed not to see the Islanders lose the final
regular season game.          PENS RULE!!!
```

2 Feature extraction

In order to perform text mining, we need to turn the text content into numerical feature vectors (like tf-idf), and this process is also called Vectorization. The *sklearn.feature_extraction* module helps extract features from datasets like text and image to a format supported by machine learning algorithms.[5]

2.1 Bag-of-words[6]

Note: Sample Program for this task: ***lab2_bag_of_words.py***

The bag-of-words model is commonly used in document classification. In this model, the words in each document is stored in "a bag" and the occurrence of each word will be used as a feature for representing the document.

The required packages are:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
```

We can build a simple bag-of-words feature vector by using the function **fit_transform()** of the class **CountVectorizer()**.

```
categories = ['alt.atheism', 'comp.graphics', 'sci.med',
'soc.religion.christian']
twenty_train = fetch_20newsgroups(subset='train',
categories=categories)
count_vect = CountVectorizer()
bow_train_counts = count_vect.fit_transform(twenty_train.data)
```

5 http://scikit-learn.org/stable/modules/feature_extraction.html
6 https://en.wikipedia.org/wiki/Bag-of-words_model

There are 2,257 documents in the training dataset and 35,788 tokens (i.e., distinct terms) in the dataset. Therefore, the size of the bag-of-words dataset is 2,257*35,788.

```
print('Number of documents in twenty_train.data:')
print(len(twenty_train.data))
print('Number of extracted features:')
print(len(count_vect.get_feature_names()))
print('Size of bag-of-words:')
print(bow_train_counts.shape)
```

Sample Output:

```
Number of documents in twenty_train.data:
2257
Number of extracted features:
35788
Size of bag-of-words:
(2257, 35788)
```

The bag of words feature vector is as follows:

```
print('Bag of words: [(doc_id, features_id): Occurrence] ')
print(bow_train_counts)
```

Sample output:

```
Bag of words: [(doc_id, features_id): Occurrence]
  (0, 230)      1
  (0, 12541)    1
  (0, 3166)     1
  (0, 14085)    1
  (0, 20459)    1
  (0, 35416)    1
  (0, 3062)     1
  (0, 2326)     2
  (0, 177)      2
  (0, 31915)    1
  (0, 33572)    1
  (0, 9338)     1
  (0, 26175)    1
  (0, 4378)     1
  (0, 17556)    1
  (0, 32135)    1
  (0, 15837)    1
  (0, 9932)     1
  (0, 32270)    1
  (0, 18474)    1
  (0, 27836)    1
  (0, 5195)     1
  (0, 12833)    2
  (0, 25337)    1
  (0, 25361)    1
  :      :
  (2256, 6430)  1
  (2256, 24052) 1
  (2256, 22270) 1
  (2256, 35638) 2
  (2256, 32233) 1
```

In the output, features_id refers to the corresponding feature and the feature list can be accessed by *count_vect.get_feature_names().*

```
print(count_vect.get_feature_names())
```

Here is part of the output:

```
'_this_', '_threats', '_thrice', '_told_', '_treefold', '_true', '_tsv_', '_university_physics_',
'_unquestioningly', '_until_', '_us_', '_used_', '_very_', '_want_', '_wants_', '_was_',
'_washington', '_waving', '_we', '_what_', '_when', '_which', '_whole', '_whole_', '_why',
'_with', '_writing_', '_you', '_you_', '_your_', '_yourself_', 'a000', 'a0000', 'a1', 'a10k',
'a1200', 'a137490', 'a16pd', 'a2623', 'a2i', 'a2wj', 'a2x', 'a6', 'a66', 'a7d56e52',
'a7f2fc4e8b01023c', 'a7f81fd2f801023c', 'a7fac234c902019a', 'a9', 'a_', 'a_costa',
'a_valid_return_e', 'aa', 'aa00076', 'aa00449', 'aa09923', 'aa17104', 'aa21125', 'aa888', 'aaa',
'aaaa', 'aaai', 'aacc', 'aad', 'aah', 'aalborg', 'aamrl', 'aanerud', 'aangeboden',
'aantal_snijpunten', 'aao', 'aaoepp', 'aap', 'aaplay', 'aarhus', 'aario', 'aarnet', 'aaron',
'aaronc', 'aaronson', 'aatchoo', 'aau', 'ab', 'ab20', 'ab4z', 'ab961', 'abacus', 'abad',
'abandon', 'abandoned', 'abandoning', 'abates', 'abberation', 'abbott', 'abbreviation', 'abc',
'abdomen', 'abdominal', 'abducted', 'abduction', 'abdullah', 'abeit', 'abekas', 'abel',
'aberdeen', 'aberrant', 'aberration', 'abhin', 'abhorent', 'abhorrences', 'abhorrent', 'abide',
'abideth', 'abiding', 'abigail', 'abild', 'abildskov', 'abilities', 'ability', 'abiliy',
'abington', 'abiogenesis', 'ablaze', 'able', 'ably', 'abner', 'abnormal', 'abnormalities',
'abnormally', 'abo', 'aboard', 'abodes', 'abolish', 'abolished', 'abolishment', 'abolition',
'abomination', 'abord', 'abort', 'abortion', 'abortions', 'abou', 'abound', 'about', 'above',
'above_', 'abp', 'abp2', 'abpsoft', 'abput', 'abraam', 'abraham', 'abram', 'abrash', 'abrasions',
'abraxis', 'abreast', 'abri', 'abridged', 'abroad', 'abruptly', 'abs', 'abscence', 'abscess',
'absence', 'absent', 'absitinence', 'absol', 'absolute', 'absolutely', 'absolutes', 'absolutism',
'absolutist', 'absolve', 'absorb', 'absorbed', 'absorbtion', 'absorption', 'abstact', 'abstain',
'abstaining', 'abstinence', 'abstinencne', 'abstract', 'abstraction', 'abstractions', 'abstracts',
'abstractsoft', 'absurd', 'absurdities', 'absurdity', 'absurdum', 'absure', 'abundance',
'abundant', 'abundantly', 'abuse', 'abused', 'abusers', 'abuses', 'abusing', 'abusive', 'ac',
'ac534', 'ac999266', 'acad', 'acad1', 'acad11', 'acad2', 'acad3', 'academ07', 'academia',
'academic', 'academically', 'academics', 'academy', 'acadia', 'acadiau', 'acaps', 'acc',
'accelerator', 'accelerators', 'accent', 'accentuate', 'accept', 'acceptable', 'acceptably',
'acceptance', 'accepted', 'accepting', 'accepts', 'accesible', 'access', 'accessed', 'accessible',
'accessories', 'accid', 'accident', 'accidental', 'accidentally', 'accidently', 'accidents',
'acclimatization', 'acclimatize', 'acclimitazation', 'accom', 'accommodate', 'accommodation',
'accomodate', 'accomodated', 'accomodations', 'accompanied', 'accompanies', 'accompany',
'accompanying', 'accomplish', 'accomplished', 'accomplishment', 'accomplishments', 'accor',
'accord', 'accord2', 'accordance', 'accorded', 'according', 'accordingly', 'accosted', 'account',
'accountability', 'accountable', 'accountants', 'accounted', 'accounting', 'accounts', 'accpak',
'accpeted', 'accredit', 'accreditation', 'accredited', 'accrodingly', 'accrued', 'accuartely',
'accuate', 'accucorp', 'accufont', 'accukey', 'accukey1', 'accukey2', 'accumulate', 'accumulated',
'accumulation', 'accupuncture', 'accupuntunce', 'accuracy', 'accurate', 'accurately',
'accusation', 'accusations', 'accuse', 'accused', 'accuses', 'accusing', 'accustomed', 'accutane',
'ace', 'acegr', 'aceh', 'aceitunas', 'acenet', 'acepting', 'acetic', 'acetominophen', 'acf2',
'acf3', 'ache', 'acheive', 'acheived', 'aches', 'achieve', 'achieved', 'achievement', 'achieves',
'achieving', 'achim', 'aching', 'achive', 'achived', 'achses', 'acid', 'acidic', 'acidophilis',
'acidophilius', 'acidophilous', 'acidophilus', 'acids', 'acis', 'ack', 'acknosledge',
'acknowledge', 'acknowledged', 'acknowledgement', 'acknowledgements', 'acknowledges',
'acknowledging', 'acknowledgment', 'acknowleding', 'aclimatized', 'acm', 'acme', 'acn', 'acne',
'acns', 'acooper', 'acording', 'acorn', 'acosta', 'acoustical', 'acoustique', 'acpub',
'acquaintace', 'acquaintance', 'acquire', 'acquired', 'acquiring', 'acquisition', 'acquittal',
'acri', 'acrid', 'acronym', 'across', 'acs', 'acsa', 'acsc', 'acsu', 'act', 'actaully', 'acted',
'actg', 'acting', 'action', 'actions', 'activa', 'activate', 'activated', 'activating',
'activation', 'active', 'actively', 'activists', 'activities', 'activitiests', 'activity',
'acton', 'actor', 'actors', 'actrix', 'acts', 'actual', 'actualization', 'actuallay', 'actually',
'actuelles', 'acupuncture', 'acupuncturist', 'acupuncturists', 'acutally', 'acutane', 'acute',
'acuuracy', 'acvax2', 'acyclovir', 'ad', 'ad70', 'ad96', 'ad994', 'ad99s461', 'adabas',
'adaclabs', 'adage', 'adagio', 'adam', 'adamantly', 'adams', 'adamski', 'adapt', 'adaptable',
'adaptation', 'adaptations', 'adapted', 'adapter', 'adapting', 'adaptiove', 'adaptive', 'adaptor',
'aday', 'adc', 'adcs00', 'adcs01', 'add', 'adda', 'added', 'addendum', 'adder', 'addict',
'addiction', 'addictive', 'addicts', 'adding', 'addison', 'addition', 'additional',
'additionally', 'additions', 'additive', 'additives', 'additivity', 'address',
```

The features being used here are just word occurrence frequencies. In the next section, tf-itf will be used as the features.

3 Document Classification

Note: Sample program for this task: ***lab2_naivebayes.py***

3.1 Preparing dataset

We will use several packages in sklearn and numpy.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
import numpy as np
```

Also, we need to load the training dataset.

```
categories = ['alt.atheism', 'comp.graphics', 'sci.med',
'soc.religion.christian']
twenty_train = fetch_20newsgroups(subset='train',
categories=categories)
```

3.2 Extracting features from text files

3.2.1 Tokenize

```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(twenty_train.data)
```

3.2.2 Building Tf-idf matrix

We used tf-idf as the features in this section. Note: Scikit-learn provides you the function and there is no need to write your code to compute the tf-idf features. Also, the provided function is more efficient.

```
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

3.3 ***Training and testing a classifier***[7]

After the features are extracted from the dataset, we can then train a classifier, like ***Naïve Bayes Classifier*** as shown in the following example, to predict the post category label.

```
clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target)
```

Then, we can use the trained model to predict the outcome on the documents in the test set. A new document should first be tokenized and transformed into the format that fits the classifier. The steps are similar to those in 3.2, except that we need to change the function *fit_transform*() to *transform()*.

```
twenty_test = fetch_20newsgroups(subset='test', categories=categories)
X_test_counts = count_vect.transform(twenty_test.data)
X_test_tfidf = tfidf_transformer.transform(X_test_counts)
```

Pass the prepared data to the trained classifier and predict the result.

```
predicted = clf.predict(X_test_tfidf)
```

Print out the result.

```
for doc, category in zip(twenty_test.data, predicted):
    print('Classified as: %s\n%s\n' %
(twenty_train.target_names[category], doc))
```

Part of the sample output:

```
Classified as: soc.religion.christian
From: bj368@cleveland.Freenet.Edu (Mike E. Romano)
Subject: Re: Drop your drawers and the doctor will
Organization: Case Western Reserve University, Clev
Lines: 11
NNTP-Posting-Host: hela.ins.cwru.edu


This is not an unusual practice if the doctor is al
member of a nudist colony.



--
Sir, I admit your gen'ral rule
That every poet is a fool;
But you yourself may serve to show it,
That every fool is not a poet.    A. Pope


Classified as: sci.med
From: Donald Mackie <Donald_Mackie@med.umich.edu>
Subject: Re: re:use of haldol and the elderly
Organization: UM Anesthesiology
Lines: 40
Distribution: world
NNTP-Posting-Host: 141.214.86.38
X-UserAgent: Nuntius v1.1.1d9
X-XXMessage-ID: <A8009C58410C5626@38.86.214.141.in-
X-XXDate: Sun, 25 Apr 93 01:51:52 GMT
```

[7] https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html#training-a-classifier

Exercise 1

Identify the category label for the following text by using the same classifier above.

```
new_doc = ['God is love', 'OpenGL on the GPU is fast', 'Atheism is the absence of belief in the existence of deities', 'Computer graphics are pictures and films created using computers', 'The Ten Commandments']
```

Display the result by format:

Classified as: category type

Document Content

Part of the sample output:

```
Classified as: soc.religion.christian
God is love
```

3.4 Evaluating the performance of the model

3.4.1 *Model Performance*

For evaluating the model performance, we usually use the following terminologies:

Confusion Matrix:[8]

	PREDICTED CLASS LABEL			
ACTUAL CLASS LABEL		True	False	
	True	True Positive (TP)	False Negative (FN)	Actual Positive = TP+FN
	False	False Postive (FP)	True Negative (TN)	Actual Negative = FP+TN

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

The accuracy can be calculated by the following code.

```
print('Accuracy: %.3f\n' % np.mean(predicted == twenty_test.target))
```

The accuracy of this model is **0.835**.

We can use the ***classification metrics module*** to compute the confusion matrix and classification report.

```
print(metrics.confusion_matrix(twenty_test.target, predicted))
print(metrics.classification_report(twenty_test.target, predicted,
target_names=twenty_test.target_names))
```

[8] https://en.wikipedia.org/wiki/Confusion_matrix

The output of confusion matrix:

```
[[192   2   6 119]
 [  2 347   4  36]
 [  2  11 322  61]
 [  2   2   1 393]]
```

It can be interpreted as follows:

		Predicted Class				
		alt.atheism	comp.graphics	sci.med	soc.religion.christian	Total
Actual Class	alt.atheism	192	2	6	119	319
	comp.graphics	2	347	4	36	389
	sci.med	2	11	322	61	396
	soc.religion.christian	2	2	1	393	398
	Total	198	362	333	609	1502

The output of classification report:

```
                        precision    recall  f1-score   support

           alt.atheism       0.97      0.60      0.74       319
         comp.graphics       0.96      0.89      0.92       389
               sci.med       0.97      0.81      0.88       396
soc.religion.christian       0.65      0.99      0.78       398

              accuracy                           0.83      1502
             macro avg       0.89      0.82      0.83      1502
          weighted avg       0.88      0.83      0.84      1502
```

4 Vector Representation

4.1 Dimension reduction using truncated SVD[9]

In statistics, machine learning, and information theory, dimension reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables.[10] This section shows how to reduce the dimension of the tf-idf vector of each document via Truncated SVD and use the reduced tf-idf to perform classification. Here we use Logisitic Regression[11] classifier rather than Naïve Bayes, because the latter cannot handle negative features that are commonly seen after dimension reduction.

Sample program for this section can be found in **lab2_truncated_svd.py.*

Similar to the previous section, we need to import all the required packages first. Then, we load data and obtain raw word count.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import TruncatedSVD
import numpy as np

categories = ['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
twenty_train = fetch_20newsgroups(subset='train', categories=categories)
twenty_test = fetch_20newsgroups(subset='test', categories=categories)

# raw frequency
count_vect = CountVectorizer()
X_train = count_vect.fit_transform(twenty_train.data)
X_test = count_vect.transform(twenty_test.data)
```

Since we want to investigate the influence of different data formats to the performance of the same model, we allow ourselves to manually specify different types of input, as follows.

[9] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
[10] https://en.wikipedia.org/wiki/Dimensionality_reduction
[11] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

```
method = int(input('input your data pre-processing method: 0 (default) for
raw freqency, 1 for tf-idf, and 2 for tf-idf with truncated svd: '))

if method >= 1:
    # tf-idf
    tfidf_transformer = TfidfTransformer()
    X_train = tfidf_transformer.fit_transform(X_train)
    X_test = tfidf_transformer.transform(X_test)

if method == 2:
    svd = TruncatedSVD(n_components=50, n_iter=25, random_state=12)
    X_train = svd.fit_transform(X_train)
    X_test = svd.transform(X_test)

clf = LogisticRegression().fit(X_train, twenty_train.target)
predicted = clf.predict(X_test)

print('Accuracy: %.3f\n' % np.mean(predicted == twenty_test.target))
```

In particular, we can set the dimension of reduced vectors (i.e., n_components). The performances of different data types are summarized in the table below.

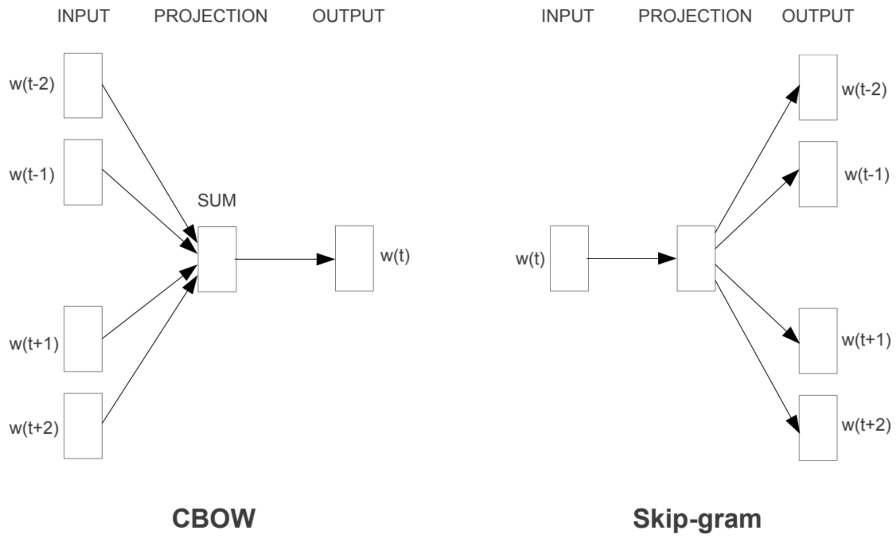
Input	**Dimension of SVD**	**Accuracy**
Word Count	N/A	0.891
Tf-idf	N/A	0.897
Reduced tf-idf	50	0.846
	500	0.889
	1000	0.895
	5000	0.897

From the table, we can observe that Tf-idf is better than Word Count, which confirms that Tf-idf is a better way of representation. When the dimension of SVD is quite small, the Reduced tf-idf even performs worse than Word Count, because it lost a lot of useful information during the dimension reduction process. When the dimension is large enough, the performance of Reduced tf-idf is close to Tf-idf.

4.2 Word embeddings from Word2Vec[12] [Reference Only]

Note: Sample programmes for this task are: ***train_word2vec.py, plot_word_embeddings.py, lab2_word_similarity.py, lab2_word2vec_classification.py****.*

Word embedding is a vector that can capture the semantic meaning of the word. Word2Vec consists of two typical training methods to obtain such semantic embeddings, i.e., CBOW (continuous bag-of-words) and Skip-gram. In CBOW, the model predicts the current word from a window of surrounding context words, while in Skip-gram, the model uses the current word to predict the surrounding window of context words. There are also other ways to obtain word embeddings, e.g., GloVe[13].



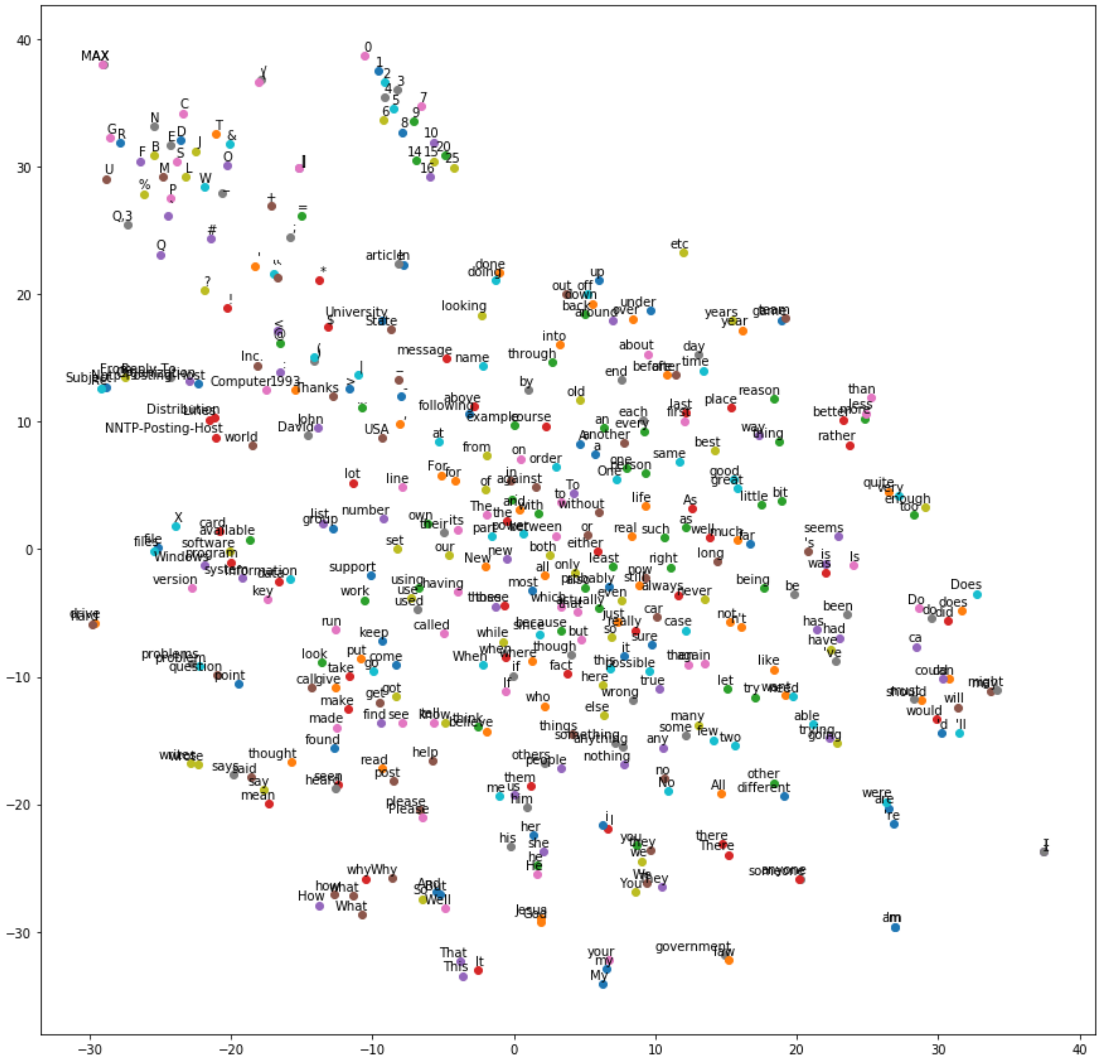
Model image from the paper "Efficient Estimation of Word Representations in Vector Space".

The original Word2Vec embeddings file[14] trained on Google News is very large (approximately 4GB). Therefore, we trained a small one called **20news-vectors-negative100.bin** on 20newsgroups (see **train_word2vec.py**) using an off-the-shelf package Gensim[15].

To show how word embeddings can capture the words' semantic meanings, we can reduce the dimension of word vectors to 2D and visualize them (**plot_word_embeddings.py**).

---

12 https://en.wikipedia.org/wiki/Word2vec
13 https://nlp.stanford.edu/projects/glove/
14 https://drive.google.com/open?id=0B7XkCwpI5KDYNlNUTTlSS21pQmM
15 https://radimrehurek.com/gensim/models/word2vec.html



Taking a look at the image, we can find that the semantically close words are grouped together, such as numbers, punctuation, and alphabets.

The Gensim pakage actually provides many functions[16] for us to find words of large similarity.

```
from gensim.models import Word2Vec

word_vectors = Word2Vec.load('./20news-vectors-negative100.model')

result = word_vectors.wv.most_similar(positive=['woman', 'husband'],
negative=['man'])
print(result)
similarity = word_vectors.wv.similarity('football', 'baseball')
```

[16] https://radimrehurek.com/gensim/models/keyedvectors.html

```
print(similarity)
similarity = word_vectors.wv.similarity('football', 'mac')
print(similarity)
result = word_vectors.wv.similar_by_word('baseball')
print(result)
```

The first one shows the famous example “King - Man + Woman = Queen”, but since the dataset is quite small and may not be able to well capture the meaning of “King” and “Queen”, we use “Husband” and “Wife” here instead.

```
[('wife', 0.7159417271614075), ('poison', 0.688998818397522), ('sister', 0.6712672710418701), ('daughter',
0.6493616104125977), ('father', 0.6438934206962585), ('stomach', 0.6399938464164734), ('neighbours',
0.6371129751205444), ('girlfriend', 0.621023416519165), ('breasts', 0.6200904846191406), ('her',
0.6110406517982483)]
0.7033003
0.3495584
[('hockey', 0.7883487939834595), ('football', 0.7033002972602844), ('game', 0.6999071836471558), ('league',
0.6912289261817932), ('team', 0.6898064017295837), ('playoff', 0.6772750616073608), ('playoffs',
0.6693721413612366), ('games', 0.6637589335441589), ('teams', 0.6592584252357483), ('pitching',
0.6519320607185364)]
```

At last, we see how to leverage word embeddings for classification. The idea is that we compute the mean embedding of all the words in one document, and regard it as the document embedding (this may not be reasonable enough). Afterwards, we can perform classification via Logistic Regression by treating document embeddings as features.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.linear_model import LogisticRegression
from gensim.models import KeyedVectors
from nltk import word_tokenize
import numpy as np

def get_doc_embedding(text, word_vectors):
    vector_list = []
    word_list = word_tokenize(text)
    for w in word_list:
        if w in word_vectors:
            # skip the words that are not in word2vec
            vector_list.append(word_vectors[w])

    doc_matrix = np.asarray(vector_list, dtype=np.float32)
    doc_vec = np.mean(doc_matrix, axis=0)
```

```
    return doc_vec

def get_data_embedding(doc_list, word_vectors):
    vector_list = []
    for doc in doc_list:
        doc_vec = get_doc_embedding(doc, word_vectors)
        vector_list.append(doc_vec)
    doc_matrix = np.asarray(vector_list, dtype=np.float32)

    return doc_matrix

categories = ['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
twenty_train = fetch_20newsgroups(subset='train', categories=categories)
twenty_test = fetch_20newsgroups(subset='test', categories=categories)

word2vec = KeyedVectors.load_word2vec_format('./20news-vectors-
negative100.bin', binary=True)
X_train = get_data_embedding(twenty_train.data, word2vec)
X_test = get_data_embedding(twenty_test.data, word2vec)

clf = LogisticRegression().fit(X_train, twenty_train.target)
predicted = clf.predict(X_test)

print('Accuracy: %.3f\n' % np.mean(predicted == twenty_test.target))
```

The accuracy (**0.830**) is actually not satisfactory compared against Tf-idf's **0.897** in the last subsection, because simply computing mean embedding of the words in one document cannot well characterize the relationship between these words. Moreover, to unleash the power of word embeddings, usually other more complex models, like CNN[17] and RNN[18], are utilized instead of logistic regression.

5 *Document Clustering*

Clustering is the task of grouping a set of objects into different groups, or called clusters. The similarity between two arbitrary selected objects within a cluster,

[17] https://en.wikipedia.org/wiki/Convolutional_neural_network
[18] https://en.wikipedia.org/wiki/Recurrent_neural_network

should be larger than two arbitrary selected objects from different clusters.[19] One commonly used clustering method is k-means and we will use this as an example in this section.

5.1 Preparing dataset

Sample program for this section can be found in **lab2_clustering.py.*

We will use several packages in sklearn and numpy.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import metrics
from sklearn.cluster import KMeans
import numpy as np
```

Also, we need to load the training dataset and define the value of **K**, which is the number of cluster that we expected.

```
categories = ['alt.atheism','talk.religion.misc','comp.graphics','sci.space']
dataset = fetch_20newsgroups(subset='all', categories=categories)
labels = dataset.target
k = len(np.unique(labels))
```

5.2 Extracting features from text files

TfidfVectorizer() is equivalent to CountVectorizer and TfidfTransformer we used in Section 3.[20]

```
vectorizer = TfidfVectorizer(max_df=0.5, min_df=2, stop_words='english')
X = vectorizer.fit_transform(dataset.data)
```

5.3 Performing clustering[21]

```
km = KMeans(n_clusters=k, max_iter=100, n_init=1)
km.fit_transform(X)
```

[19] https://en.wikipedia.org/wiki/Cluster_analysis#cluster

[20] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer

[21] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

5.4 *Evaluating the performance of the model*

There are several measures for evaluating clustering performance. We will use ARI, homogeneity, completeness and V-measure here as the ground truth is available. [22] If no ground truth is provided, we can access the silhouette_score to evaluate the performance. [23]

```
print('Clustering statistic: \n')
print('ARI: %0.3f' % metrics.adjusted_rand_score(labels, km.labels_))
print('Homogeneity: %0.3f' % metrics.homogeneity_score(labels, km.labels_))
print('Completeness: %0.3f' % metrics.completeness_score(labels,
km.labels_))
print('V-measure: %0.3f\n' % metrics.v_measure_score(labels, km.labels_))
print('Top terms per cluster:')
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(k):
    print('Cluster %d:' % i, end='')
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind], end='')
    print()
```

Sample output: ***(Note: The result may not be exactly the same.)***

```
Clustering statistic:

ARI: 0.321
Homogeneity: 0.335
Completeness: 0.473
V-measure: 0.393

Top terms per cluster:
Cluster 0:
 uk
 ac
 mathew
 mantis
 buffalo
 mcc
 rusnews
 lilley
 cc
 university

Cluster 1:
 sgi
 keith
 livesey
```

[22] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.cluster

[23] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score

Here is a brief summary of the performance evaluation measure in sklearn:

Classification	**Clustering**	
	With ground truth	**Without ground truth**
Accuracy	Adjusted Rand Index	Silhouette Coefficient
Precision	Homogeneity	
Recall	Completeness	
F1-score	V-measure	

Exercise 2:

Change the parameters as follow and generate the results with the same format as Section 5.4.

2.1. *categories = None* and *k = len(np.unique(labels))*

2.2. *categories = None* and *k = 6*

6 *Word Cloud*

Apart from just displaying the word directly, we can also build a word-cloud to visualize the result.

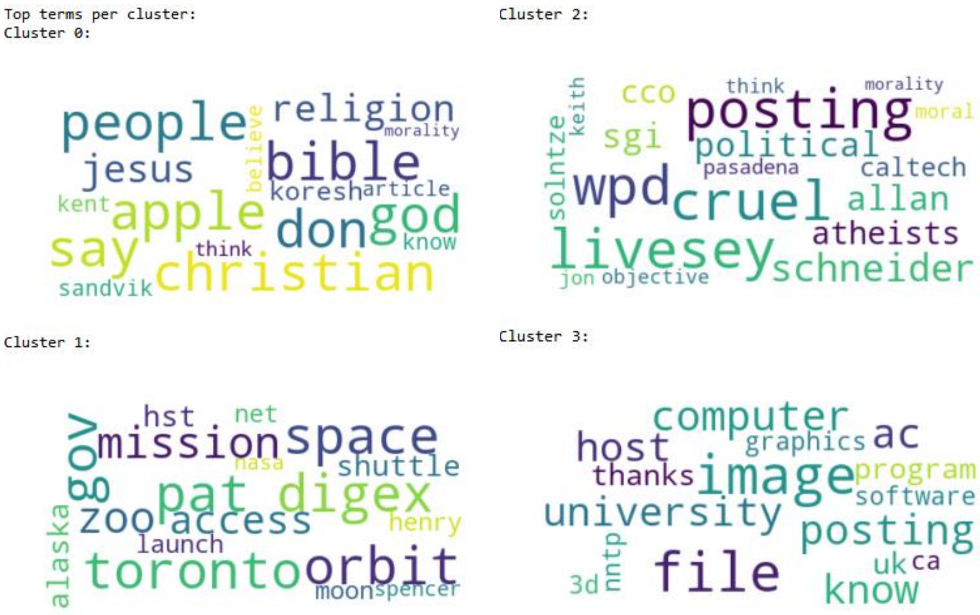


Figure 1 Word Cloud Examples

In order to build the word-cloud, we need to install the package "wordcloud". Go to command prompt and type ***"pip install wordcloud"***.

Sample Program for this task: ***lab2_wordcloud.py.***

The required packages for this example are:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

To generate a word cloud, we need to pass the processed text to function generate() under class WordCloud().[24]

```
wordcloud = WordCloud().generate(text)
```

[24] https://amueller.github.io/word_cloud/generated/wordcloud.WordCloud.html#wordcloud.WordCloud

For example,

```
text = 'To introduce the fundamental concepts as well as practical applications of contemporary \
            Artificial Intelligence incorporating knowledge discovery and data mining social \
            network intelligence and intelligent agents and advanced Information Technology in the \
            context of Web empowered social computing systems environments and activities To \
            discuss the techniques and issues central to the development of social computing and Web \
            intelligence systems'
wordcloud = WordCloud().generate(text)
wordcloud.to_file('wordcloud.png')
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Sample Output:

Note: The result may not be exactly the same.

Exercise 3

Use the data set ***fetch_20newsgroups(subset='all', categories=None)*** (you can refer to Section 5) to perform clustering (where k = 6) and use the clustered result to generate the corresponding word cloud and give each cluster a name.

For example,

Cluster 1: Microsoft window OS

Cluster 2: Computer Hardware

7 K-means Plot per Iteration [Optional]

In previous exercises, we run k-means clustering and get the best fit result directly. In this part, we aim to investigate how the clustering algorithm is applied to the data set and visualize the result per iteration.

Sample Program for this task: ***lab2_kmeans_plot.py.***

Import k-means and matplotlib.

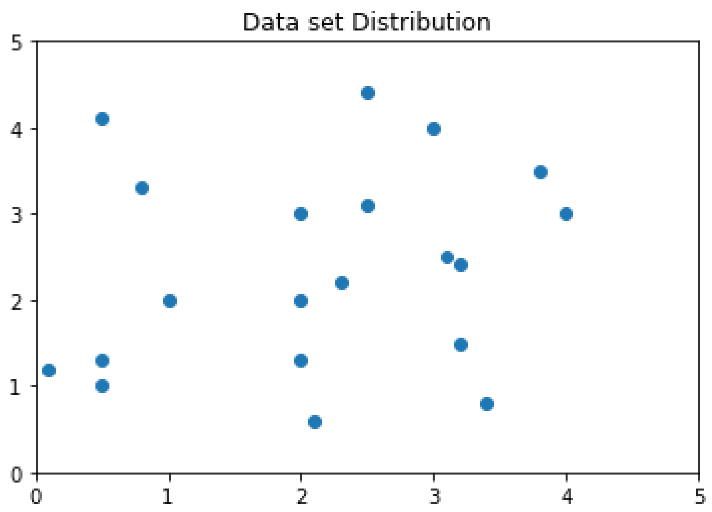
```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
```

We want to apply k-means clustering to the following data set:

```
x = np.array([2.0, 2.3, 2.1, 3.2, 3.8, 3.1, 3, 2, 0.5, 2.5, 0.5, 1, 3.4, 2.5, 4, 0.1, 0.8,
3.2, 2, 0.5])
y = np.array([1.3, 2.2, 0.6, 1.5, 3.5, 2.5, 4, 2, 4.1, 4.4, 1.3, 2, 0.8, 3.1, 3, 1.2, 3.3,
2.4, 3, 1.0])
```

First, use scatter plot to see the distribution.[25]

```
plt.scatter(x,y)
plt.title('Data set Distribution')
plt.xlim([0, 5])
plt.ylim([0, 5])
plt.show()
```



[25] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html#matplotlib.pyplot.scatter

Before applying k-means, we need to initialize the following parameters.

```
X = np.array(list(zip(x, y)))
K = 3
centroids = np.array([(0, 0), (3, 3), (6, 6)])
```

X refers to the merged data set x and y in an array format with size 20*2.

Also, we can initial centers by passing an ndarray in size n_clusters*n_features.

In the above example, we want to generate 3 clusters with 2 features (x and y). Therefore, the centers we provided are: [(0, 0), (3, 3), (6, 6)].

As parameter init in function Kmeans() required an ndarray for customized centroids, we need to format the given centers to ndarray by using np.array().

On the other hand, we also need to initialize colors and markers for plotting.

```
colors = ['b', 'g', 'c']
markers = ['o', 'v', 's']
```

They can visualize clusters in different colors and symbols. Details about colors and markers can be found here:

Color:
https://matplotlib.org/api/colors_api.html
Marker:
https://matplotlib.org/api/_as_gen/matplotlib.markers.MarkerStyle.html#matplotlib.markers.MarkerStyle

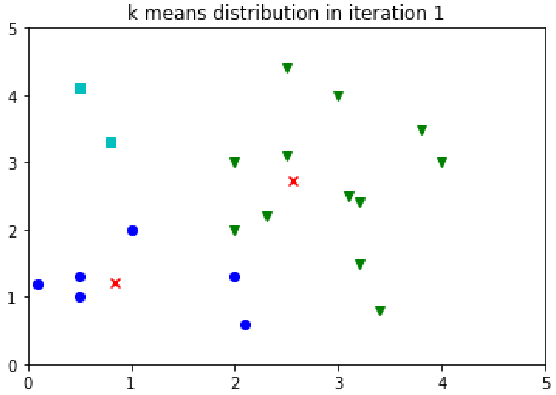
With the following code, we can plot data point on 2D plane and show the centroids of the k-means for each iteration.

```
for i in range(1, 4):
        print('=======For iteration %i========' % i)
        print('Centroids before applying kmeans:')
        print(centroids)
        kmeans_model = KMeans(n_clusters=K, init = centroids,
        max_iter=1).fit(X)
        centroids = np.array(kmeans_model.cluster_centers_)
        print('Centroids after applying kemans: ')
        print(centroids)
        plt.plot()
        plt.title('k means distribution in iteration %i' % i)
        for i, l in enumerate(kmeans_model.labels_):
                plt.plot(x[i], y[i], color=colors[l], marker=markers[l])
```

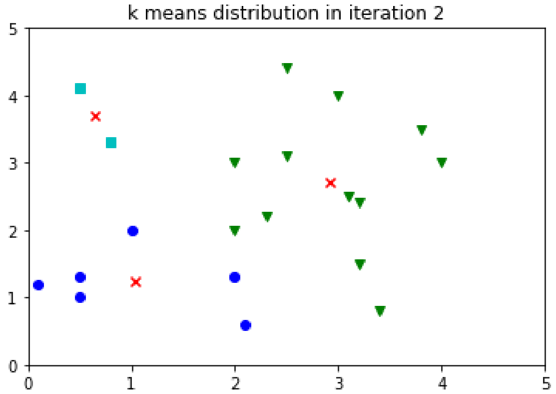
```
    plt.xlim([0, 5])
    plt.ylim([0, 5])
    plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='r')
    plt.show()
```

Sample Output:

```
========For iteration 1========
Centroids before applying kmeans:
[[0 0]
 [3 3]
 [6 6]]
Centroids after applying kemans:
[[ 0.84        1.22      ]
 [ 2.55333333  2.74      ]
 [ 0.5         4.1       ]]
```




```
========For iteration 2========
Centroids before applying kmeans:
[[ 0.84        1.22      ]
 [ 2.55333333  2.74      ]
 [ 0.5         4.1       ]]
Centroids after applying kemans:
[[ 1.03333333  1.23333333]
 [ 2.91666667  2.7       ]
 [ 0.65        3.7       ]]
```



8 Appendix

8.1 Information about different classifiers

We have introduced Naïve Bayes Classifier, Logistic Regression Classifier and K-means Clustering in this lab section. Here are some other common algorithms for data mining:

Model	Example Link
Linear Regression	http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py
Decision Trees	http://scikit-learn.org/stable/modules/tree.html#classification
DBSCAN	http://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html#sphx-glr-auto-examples-cluster-plot-dbscan-py
Hierarchical Clustering	http://scikit-learn.org/stable/auto_examples/cluster/plot_ward_structured_vs_unstructured.html#sphx-glr-auto-examples-cluster-plot-ward-structured-vs-unstructured-py