Wk2_SLP2_Feature extraction

March 8, 2021

[3]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
from sklearn import metrics
%matplotlib inline
```

0.1 Introduction

This week we will look at working with text as data, how to extract features from text and the use of a clustering algorithm.

We will take some samples of texts and look at how to extract a fixed set of features from each text to use in clustering. We'll then look at how to measure the similarity or distance between two texts. Finally we'll look at the KMeans clustering algorithm.

New concepts this week: - using **feature extraction** methods to create features from texts - **sparse arrays** are used to store arrays where many of the values will be zero - comparing the similarity of two samples using a **distance metric** - the **kmeans clustering** algorithm

0.2 Finding Text Data

The example this week is derived from this example in the sklearn documentation.

We will use some data from sklearn, this is the 20 newsgroups dataset containing messages from the old Usenet discussion boards. We select just four of the groups giving us messages on four topics. We choose two that are probably quite close together (atheism and religion) and two that should be quite different.

The result is an sklearn dataset, the actual data is available as dataset.data, the newsgroup names are in dataset.target.

[4]:
```
# Load some categories from the training set
categories = [
    'alt.atheism',
    'talk.religion.misc',
    'comp.graphics',
    'sci.space',
]
```

```
dataset = fetch_20newsgroups(subset='all', categories=categories,
                             shuffle=True, random_state=42)
len(dataset.target)
```

[4]: 3387

[5]:
```
# we can look at the first message in the data
print(dataset.data[0])
```

```
From: healta@saturn.wwc.edu (Tammy R Healy)
Subject: Re: who are we to judge, Bobby?
Lines: 38
Organization: Walla Walla College
Lines: 38

In article <1993Apr14.213356.22176@ultb.isc.rit.edu> snm6394@ultb.isc.rit.edu
(S.N. Mozumder ) writes:
>From: snm6394@ultb.isc.rit.edu (S.N. Mozumder )
>Subject: Re: who are we to judge, Bobby?
>Date: Wed, 14 Apr 1993 21:33:56 GMT
>In article <healta.56.734556346@saturn.wwc.edu> healta@saturn.wwc.edu (TAMMY R
HEALY) writes:
>>Bobby,
>>
>>I would like to take the liberty to quote from a Christian writer named
>>Ellen G. White.  I hope that what she said will help you to edit your
>>remarks in this group in the future.
>>
>>"Do not set yourself as a standard.  Do not make your opinions, your views
>>of duty, your interpretations of scripture, a criterion for others and in
>>your heart condemn them if they do not come up to your ideal."
>>                         Thoughts Fromthe Mount of Blessing p. 124
>>
>>I hope quoting this doesn't make the atheists gag, but I think Ellen White
>>put it better than I could.
>>
>>Tammy
>
>Point?
>
>Peace,
>
>Bobby Mozumder
>
My point is that you set up your views as the only way to believe.  Saying
that all eveil in this world is caused by atheism is ridiculous and
```

```
counterproductive to dialogue in this newsgroups.  I see in your posts a
spirit of condemnation of the atheists in this newsgroup bacause they don'
t believe exactly as you do.  If you're here to try to convert the atheists
here, you're failing miserably.  Who wants to be in position of constantly
defending themselves agaist insulting attacks, like you seem to like to do?!
I'm sorry you're so blind that you didn't get the messgae in the quote,
everyone else has seemed to.

Tammy
```

0.3 Feature Extraction

We can't work directly with text as data - we need some kind of numerical or categorical features to use in the algorithms we're working with. Text has a variable number of words per sample, we need a fixed set of features.

A very common feature type for text treats each sample as a *bag of words* and just records how often each word is present in the text. Each word becomes a feature, the value is the count of how many times it occurs in the sample. Of course, there will be thousands of words in general, so we just choose the N most frequent words as features.

The idea is that if a particular word occurs a lot in two texts, they might be similar. If the same pattern of words is frequent in both, even more so. However, some words are very frequent in all texts - and, of, is etc - they don't tell you much about what the text is saying; it is common to remove these common words, generally known as *stop words*, before you do any feature extraction.

SKLearn has a collection of text feature extraction methods that we can make use of. We'll use the simplest of these, CountVectorizer which just counts the number of times a word occurs in the text. We pass it a parameter that defines the maximum number of features (words) to use and the name of the stop word list.

Once we've made a vectorizer, we can use the *fit_transform* method to apply it to a set of data. In this first example we will just compute 10 features, just to make it easier to look at the results.

[6]:
```
count_vec = CountVectorizer(max_features=10, stop_words='english')
X = count_vec.fit_transform(dataset.data)
```

The result *X_count* is a SciPy sparse matrix.
Many of the feature values will be zero if the given word does not occur in the text. To store all of these zeros would be very wasteful of memory, so we use a *sparse matrix* which uses methods to only store the data that is non-zero. The SciPy sparse matrix classes support some of the matrix methods that you can use on regular Numpy arrays or Pandas dataframes, but not all.

In the example below we use the *getrow* method to get a single row and the *toarray* method to convert this to a regular numpy array.

First, we can look at the words that have been selected as features via the *feature_names* method on the vectorizer:

[7]:
```
count_vec.get_feature_names()
```

[7]:
```
['article',
 'com',
 'don',
 'edu',
 'god',
 'lines',
 'organization',
 'space',
 'subject',
 'writes']
```

Note that we only chose 10 features so they aren't likely to be very good at characterising the texts. You might also notice that we have 'words' like *com* and *edu*, probably from email addresses and *don* which is probably from *don't*. The question of what is a word is not a simple one.

0.4 Measuring Similarity

We now have a fixed size feature set for each text - the frequency of ten words. We can look at the features that have been computed for the first text:

[8]:
```
X.getrow(0).toarray()
```

[8]:
```
array([[2, 0, 1, 6, 0, 2, 1, 0, 2, 2]], dtype=int64)
```

this means that the word *article* appears twice in the text, *edu* appears six times and *com* and *god* do not appear at all.

If we want to measure the **similarity** of this text with another, we can compare their feature vectors. If we were dealing with points on a plane in a geometry problem, we could work out the **distance** between the points using Pythagoras Theorem. Two points that were very close could be said to be very similar. This is known as the **Euclidean distance** metric and in fact, we can use it for this problem too.

The Euclidean distance is defined as the square root of the sum of the squares of the differences between each pair of feature values:

$$distance = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

Here's an example of computing the distance between the first two rows of the dataset. I've done it explicitly with raw vector arithmetic and then using the SciPy *euclidean* function as a check:

[9]:
```
a1 = count_vec.transform([dataset.data[0]]).toarray()[0]
a2 = X.getrow(1).toarray()[0]

# import the scipy euclidean function as a check
from scipy.spatial.distance import euclidean
```

```
d1= np.sqrt((np.square(a1-a2)).sum())
d2= euclidean(a1, a2)
print("Distance between articles 0 and 1:", d1, d2)
```

```
Distance between articles 0 and 1: 4.58257569495584 4.58257569495584
```

Note that this distance isn't a physical distance in metres,m it has no units, we just know that if it is bigger, the articles are more different in their feature sets.

We can use this to look through the data and find the most similar article to a given target text. The function I've written below calculates the euclidean distance between a given target article and every other article in the dataset. It remembers the article with the smallest distance and returns it's index.

I've tested this using the vectorizer I made above (*count_vec*) to find the closest article to the first one in the datast (note that I've passed dataset.data[:1] to the function so that I don't just find the first article). The result is not very similar - we're only using 10 word features after all.

[10]:
```
def find_closest(dataset, target, vectorizer):
    """Find the most similar article in dataset to target using
    the given vectorizer to extract feature vectors
    Returns the index of the most similar article"""

    a1 = vectorizer.transform([target]).toarray()[0]
    X = vectorizer.transform(dataset)

    best = 0
    best_dist = 9999
    for i in range(X.shape[0]):
        a2 = X.getrow(i).toarray()[0]
        dist = euclidean(a1, a2)
        if dist < best_dist:
            best_dist = dist
            best = i
    return best

best = find_closest(dataset.data[1:], dataset.data[0], count_vec)

print("Closest article is ", best)
print(dataset.data[1:][best])
```

```
Closest article is  1210
From: andreasa@dhhalden.no (ANDREAS ARFF)
Subject: Re: Newsgroup Split
Lines: 41
Nntp-Posting-Host: pc137
Organization: Ostfold College

In article <NERONE.93Apr20085951@sylvester.cc.utexas.edu>
```

```
nerone@ccwf.cc.utexas.edu (Michael Nerone) writes:
>From: nerone@ccwf.cc.utexas.edu (Michael Nerone)
>Subject: Re: Newsgroup Split
>Date: 20 Apr 93 08:59:51
>In article <1quvdoINN3e7@srvr1.engin.umich.edu>, tdawson@engin.umich.edu (Chris
Herringshaw) writes:
>
>  CH> Concerning the proposed newsgroup split, I personally am not in
>  CH> favor of doing this.  I learn an awful lot about all aspects of
>  CH> graphics by reading this group, from code to hardware to
>  CH> algorithms.  I just think making 5 different groups out of this
>  CH> is a wate, and will only result in a few posts a week per group.
>  CH> I kind of like the convenience of having one big forum for
>  CH> discussing all aspects of graphics.  Anyone else feel this way?
>  CH> Just curious.
>
>I must agree.  There is a dizzying number of c.s.amiga.* newsgroups
>already.  In addition, there are very few issues which fall cleanly
>into one of these categories.
>
>Also, it is readily observable that the current spectrum of amiga
>groups is already plagued with mega-crossposting; thus the group-split
>would not, in all likelihood, bring about a more structured
>environment.
>
>--
>   /~~~~~~~~~~~~~~~~~~\/~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\
>  /    Michael Nerone   \"I shall do so with my customary lack of tact; and\
> /   Internet Address:   \since you have asked for this, you will be obliged\
>/nerone@ccwf.cc.utexas.edu\to pardon it."-Sagredo, fictional char of Galileo.\


Maybe I should point out that we are not talking about c.s.amiga.*.
Only comp.graphics.

Arff
"Also for the not religous confessor, there is a mystery of higher values,
who's birth mankind - to the last - builds upon. They are indisputible. And
often disregarded. Seldom you hear them beeing prized, as seldom as you hear
a seeing man prizeing what he sees." Per Lagerkvist, The Fist
(Free translation from Swedish)
              --Andreas Arff  andreasa@dhhalden.no--
```

Checkpoint Repeat the analysis I did above but use a larger number of features - say 200. Do you get a result that is more similar to the target article? (Hint: there is another article that directly quotes this one, that should be very similar).

[12]:
```
# make a new vectoriser with more features and repeat the analysis...
count_vec2 = CountVectorizer(max_features=200, stop_words='english')
X = count_vec2.fit_transform(dataset.data)

# look at first 20 words chosen...
print(count_vec2.get_feature_names()[:20])

#... and see how often they appear in the first text
print(X.getrow(0).toarray())
```

```
['10', '12', '14', '15', '16', '1993', '20', '24', '3d', 'ac', 'access',
'actually', 'argument', 'article', 'atheism', 'atheists', 'au', 'available',
'based', 'believe']
[[0 0 1 0 0 1 0 0 0 0 0 0 0 2 1 3 0 0 0 2 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1
  0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 2 0 0 0 0 0 0 0
  0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 2 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 1 0 0 0
  1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 2 0 0 0 0 0]]
```

[13]:
```
# use the 'find_closest' function defined above with the expanded vectoriser
best2 = find_closest(dataset.data[1:], dataset.data[0], count_vec2)

print(f"The index of the most similar article is {best2} and the text for that␣
 ↪article is printed below:")
print(dataset.data[1:][best2])
```

```
The index of the most similar article is 3157 and the text for that article is
printed below:
From: healta@saturn.wwc.edu (Tammy R Healy)
Subject: Re: who are we to judge, Bobby?
Lines: 31
Organization: Walla Walla College
Lines: 31

In article <kmr4.1572.734847158@po.CWRU.edu> kmr4@po.CWRU.edu (Keith M. Ryan)
writes:
>From: kmr4@po.CWRU.edu (Keith M. Ryan)
>Subject: Re: who are we to judge, Bobby?
>Date: Thu, 15 Apr 1993 04:12:38 GMT
>
>(S.N. Mozumder ) writes:
>>(TAMMY R HEALY) writes:
>>>I would like to take the liberty to quote from a Christian writer named
>>>Ellen G. White.  I hope that what she said will help you to edit your
>>>remarks in this group in the future.
>>>
```

```
>>>"Do not set yourself as a standard.  Do not make your opinions, your views
>>>of duty, your interpretations of scripture, a criterion for others and in
>>>your heart condemn them if they do not come up to your ideal."
>>>                          Thoughts Fromthe Mount of Blessing p. 124
>>
>>Point?
>
>       Point: you have taken it upon yourself to judge others; when only
>God is the true judge.
>
>---
>
>   Only when the Sun starts to orbit the Earth will I accept the Bible.
>
>
I agree totally with you!  Amen!  You stated it better and in less world
than I did.

Tammy
```

0.5 KMeans Clustering

Finally we look at the KMeans clustering algorithm. This makes use of the distance metric like the one we've used above. KMeans tries to find a given number of clusters in the data. It does this by grouping together the samples that are closest to one another using the distance metric.

KMeans starts by choosing K points (K is the number of clusters) somewhere in the space. These are the initial cluster centres. It then assigns each sample to one cluster based on which cluster centre it is closest too. Once all points are in a cluster, the cluster centre is re-computed and the process is repeated. This continues until there is no (or little) change to the centroids or until some maximum number of iterations.

In this example we ask the algorithm to look for 4 clusters in our data, the verbose flag will show the number of iterations as it runs:

[14]:
```
# add seed to ensure results are consistent over multiple runs of the algorithm
km = KMeans(n_clusters=4, init='k-means++', max_iter=100, n_init=1,␣
 ↪verbose=True, random_state=2021)
X_count = X
km.fit(X_count)
labels = dataset.target
```

```
Initialization complete
Iteration 0, inertia 654004.0
Iteration 1, inertia 494933.3339284395
Iteration 2, inertia 486572.27485645906
Iteration 3, inertia 485273.7501763236
```

```
Iteration 4, inertia 484925.53218183306
Iteration 5, inertia 484844.45952904725
Iteration 6, inertia 484764.8039701195
Iteration 7, inertia 484697.4813506927
Iteration 8, inertia 484674.72491218866
Iteration 9, inertia 484672.3718347494
Converged at iteration 9: strict convergence.
```

km is now the result of clustering, *km.labels_* are the labels assigned to each sample, they are just numbers 0..3 since the algorithm doesn't know what the true labels were.

This is an example of an **Unsupervised Learning** algorithm. We didn't tell it what the true answer was, we just asked it to look for a given number of clusters in the data.

To evaluate the result we can use the SKLearn metrics module. Here we compute:

- homogeneity – larger if each cluster contains members of a single class
- completeness – larger if all samples from a single class are in the same cluster
- v-measure – is the harmonic mean of the homogeneity and completeness

Ideally, these metrics would be close to 1.0

[15]:
```
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" % metrics.v_measure_score(labels, km.labels_))
```

```
Homogeneity: 0.009
Completeness: 0.076
V-measure: 0.016
```

0.6 Extension

As an extension exercise, repeat the KMeans clustering exercise but use an alternate feature vector. The TfidfVectorizer (from sklearn.feature_extraction.text import TfidfVectorizer) uses a measure tf-idf that tries to measure how characteristic a word is in a text. Words that are usually infrequent but occur many times in a text will have a higher score. Use a much higer number of features (say 1000) and see if you can get a better set of evaluation scores than in the example above.

[16]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# create TfidfVectorizer using the same options as with the CountVectorizer
tfidf_vec = TfidfVectorizer(max_features = 1000, stop_words='english')
X = tfidf_vec.fit_transform(dataset.data)

km_tfidf = KMeans(n_clusters=4, init='k-means++', max_iter=100, n_init=1,␣
 ↪verbose=True, random_state=2021)
X_tfidf = X
km_tfidf.fit(X_tfidf)
labels_tfidf = dataset.target
```

```
Initialization complete
Iteration 0, inertia 6027.639952145608
Iteration 1, inertia 3147.0244430588227
Iteration 2, inertia 3124.996617489158
Iteration 3, inertia 3120.0770873158904
Iteration 4, inertia 3116.9636734992855
Iteration 5, inertia 3114.257916579225
Iteration 6, inertia 3112.9944953693307
Iteration 7, inertia 3112.3891282597274
Iteration 8, inertia 3112.1012611398296
Iteration 9, inertia 3111.911934350998
Iteration 10, inertia 3111.58688547325
Iteration 11, inertia 3111.21346055637
Iteration 12, inertia 3111.1171722801478
Iteration 13, inertia 3111.0772832927946
Iteration 14, inertia 3111.0475892835802
Iteration 15, inertia 3111.0124073963725
Iteration 16, inertia 3111.001437155224
Iteration 17, inertia 3110.997213133489
Converged at iteration 17: strict convergence.
```

[17]:
```
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_tfidf, km_tfidf.
 ↪labels_))
print("Completeness: %0.3f" % metrics.completeness_score(labels_tfidf, km_tfidf.
 ↪labels_))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_tfidf, km_tfidf.
 ↪labels_))
```

```
Homogeneity: 0.475
Completeness: 0.475
V-measure: 0.475
```

The results indicate that whilst the KMeans algorithm produces better scores using the tfidf vectoriser relative to the count vectorizer, they are far from perfect and additional feature engineering and/or tuning would be required if being analysed in more than just a training scenario.