Machine Learning :: Text feature extraction (tf-idf) – Part I

18/09/2011 Machine Learning, Python feature extraction, machine learning, text mining, tf-idf, vector space model, vsm

Short introduction to Vector Space Model (VSM)

In information retrieval or text mining, the term frequency – inverse document frequency (also called **tf-idf**), is a well know method to evaluate how important is a word

in a document. tf-idf are is a very interesting way to convert the textual representation of information into a Vector Space Model (VSM), or into sparse features, we'll discuss more about it later, but first, let's try to understand what is tf-idf and the VSM.

VSM has a very confusing past, see for example the paper The most influential paper Gerard Salton Never Wrote that explains the history behind the ghost cited paper which in fact never existed; in sum, VSM is an algebraic model representing textual information as a vector, the components of this vector could represent the importance of a term (tf–idf) or even the absence or presence (Bag of Words) of it in a document; it is important to note that the classical VSM proposed by Salton incorporates local and global parameters/information (in a sense that it uses both the isolated term being analyzed as well the entire collection of documents). VSM, interpreted in a *lato sensu,* is a space where text is represented as a vector of numbers instead of its original string textual representation; the VSM represents the features extracted from the document.

Let's try to mathematically define the VSM and tf-idf together with concrete examples, for the concrete examples I'll be using Python (as well the amazing scikits.learn Python module).

Going to the vector space

The first step in modeling the document into a vector space is to create a dictionary of terms present in documents. To do that, you can simple select all terms from the document and convert it to a dimension in the vector space, but we know that there are some kind of words (stop words) that are present in almost all documents, and what we're doing is extracting important features from documents, features do identify them among other similar documents, so using terms like "the, is, at, on", etc.. isn't going to help us, so in the information extraction, we'll just ignore them.

Let's take the documents below to define our (stupid) document space:

Antonio BIHL Vaz > Curitiba on Rastreamento em tempo real de avioes em Porto Alegre utilizando Raspberry Pi + Radio UHF (SDR RTL2832U)

Antonio on Rastreamento em tempo real de avioes em Porto Alegre utilizando Raspberry Pi + Radio UHF (SDR RTL2832U)

Recent Posts

Real time Drone object tracking using Python and OpenCV

Arduino and OLED display to monitor Redis for fun and profit

Recebendo dados de baloes meteorologicos da Aeronautica

Simple and effective coin segmentation using Python and OpenCV

Despesas de Custeio e Lei de Benford

Universality, primes and space communication

The beauty of Bitcoin P2P network

Protocoin – a pure Python Bitcoin protocol implementation

Mapa de calor dos dados de acidentes de transito do DataPoa

Book Suggestion: Codex Seraphinianus

```
Train Document Set:

d1: The sky is blue.
d2: The sun is bright.

Test Document Set:

d3: The sun in the sky is bright.
d4: We can see the shining sun, the bright sun.
```

Now, what we have to do is to create a index vocabulary (dictionary) of the words of the train document set, using the documents $d1$ and $d2$ from the document set, we'll have the following index vocabulary denoted as $\mathrm{E}(t)$ where the t is the term:

$$\mathrm{E}(t) = \begin{cases} 1, & \text{if } t \text{ is "blue"} \\ 2, & \text{if } t \text{ is "sun"} \\ 3, & \text{if } t \text{ is "bright"} \\ 4, & \text{if } t \text{ is "sky"} \end{cases}$$

Note that the terms like "is" and "the" were ignored as cited before. Now that we have an index vocabulary, we can convert the test document set into a vector space where each term of the vector is indexed as our index vocabulary, so the first term of the vector represents the "blue" term of our vocabulary, the second represents "sun" and so on. Now, we're going to use the **term-frequency** to represent each term in our vector space; the term-frequency is nothing more than a measure of how many times the terms present in our vocabulary $\mathrm{E}(t)$ are present in the documents $d3$ or $d4$, we define the term-frequency as a couting function:

$$\mathrm{tf}(t, d) = \sum_{x \in d} \mathrm{fr}(x, t)$$

where the $\mathrm{fr}(x, t)$ is a simple function defined as:

Machine Learning :: Cosine Similarity for Vector Space Models (Part III)

Rastreamento em tempo real de avioes em Porto Alegre utilizando Raspberry Pi + Radio UHF (SDR RTL2832U)

Raspberry Pi & Arduino: a laser pointer communication and a LDR voltage sigmoid

Machine Learning :: Cosine Similarity for Vector Space Models (Part III)
2 comments

Machine Learning :: Text feature extraction (tf-idf) – Part I
2 comments

Real time Drone object tracking using Python and OpenCV
0 comments

Machine Learning :: Text feature extraction (tf-idf) – Part II
1 comment

Simple and effective coin segmentation using Python and OpenCV
1 comment

Raspberry Pi & Arduino: a laser pointer communication and a LDR voltage sigmoid
0 comments

Genetic Programming meets Python
0 comments

Rastreamento em tempo real de avioes em Porto Alegre utilizando Raspberry Pi + Radio UHF (SDR RTL2832U)

$$\text{fr}(x,t) = \begin{cases} 1, & \text{if } x = t \\ 0, & \text{otherwise} \end{cases}$$

So, what the $tf(t,d)$ returns is how many times is the term t is present in the document d. An example of this, could be $tf(\text{“}sun\text{”}, d4) = 2$ since we have only two occurrences of the term “sun” in the document $d4$. Now you understood how the term-frequency works, we can go on into the creation of the document vector, which is represented by:

$$\vec{v_{d_n}} = (\text{tf}(t_1, d_n), \text{tf}(t_2, d_n), \text{tf}(t_3, d_n), \ldots, \text{tf}(t_n, d_n))$$

Each dimension of the document vector is represented by the term of the vocabulary, for example, the $\text{tf}(t_1, d_2)$ represents the frequency-term of the term 1 or t_1 (which is our “blue” term of the vocabulary) in the document d_2.

Let’s now show a concrete example of how the documents d_3 and d_4 are represented as vectors:

$$\vec{v_{d_3}} = (\text{tf}(t_1, d_3), \text{tf}(t_2, d_3), \text{tf}(t_3, d_3), \ldots, \text{tf}(t_n, d_3))$$
$$\vec{v_{d_4}} = (\text{tf}(t_1, d_4), \text{tf}(t_2, d_4), \text{tf}(t_3, d_4), \ldots, \text{tf}(t_n, d_4))$$

which evaluates to:

$$\vec{v_{d_3}} = (0, 1, 1, 1)$$
$$\vec{v_{d_4}} = (0, 2, 1, 0)$$

As you can see, since the documents d_3 and d_4 are:

```
d3: The sun in the sky is bright.
d4: We can see the shining sun, the bright sun.
```

The resulting vector $\vec{v_{d_3}}$ shows that we have, in order, 0 occurrences of the term “blue”, 1 occurrence of the term “sun”, and so on. In the $\vec{v_{d_3}}$, we have 0 occurences of the term “blue”, 2 occurrences of the term “sun”, etc.

But wait, since we have a collection of documents, now represented by vectors, we can

2 comments

Hacking into Python objects internals
1 comment

Python: acessing near real-time MODIS images and fire data from NASA’s Aqua and Terra satellites
0 comments

represent them as a matrix with $|D| \times F$ shape, where $|D|$ is the cardinality of the document space, or how many documents we have and the F is the number of features, in our case represented by the vocabulary size. An example of the matrix representation of the vectors described above is:

$$M_{|D| \times F} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix}$$

As you may have noted, these matrices representing the term frequencies tend to be very sparse (with majority of terms zeroed), and that's why you'll see a common representation of these matrix as sparse matrices.

Python practice

Environment Used: Python v.2.7.2, Numpy 1.6.1, Scipy v.0.9.0, Sklearn (Scikits.learn) v.0.9.

Since we know the theory behind the term frequency and the vector space conversion, let's show how easy is to do that using the amazing scikit.learn Python module.

Scikit.learn comes with lots of examples as well real-life interesting datasets you can use and also some helper functions to download 18k newsgroups posts for instance.

Since we already defined our small train/test dataset before, let's use them to define the dataset in a way that scikit.learn can use:

```
1. train_set = ("The sky is blue.", "The sun is bright.")
2. test_set = ("The sun in the sky is bright.",
3.    "We can see the shining sun, the bright sun.")
```

In scikit.learn, what we have presented as the term-frequency, is called **CountVectorizer**, so we need to import it and create a news instance:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
```

The **CountVectorizer** already uses as default "analyzer" called **WordNGramAnalyzer**, which is responsible to convert the text to lowercase, accents removal, token extraction, filter stop words, etc… you can see more information by printing the class information:

```
print vectorizer

CountVectorizer(analyzer__min_n=1,
analyzer__stop_words=set(['all', 'six', 'less', 'being', 'indeed', 'over', 'move', 'anyway',
'four', 'not', 'own', 'through', 'yourselves', (...)
```

Let's create now the vocabulary index:

```
vectorizer.fit_transform(train_set)
print vectorizer.vocabulary
{'blue': 0, 'sun': 1, 'bright': 2, 'sky': 3}
```

See that the vocabulary created is the same as $E(t)$ (except because it is zero-indexed).

Let's use the same vectorizer now to create the sparse matrix of our **test_set** documents:

```
smatrix = vectorizer.transform(test_set)

print smatrix

(0, 1)      1
(0, 2)      1
(0, 3)      1
(1, 1)      2
```

```
9.  (1, 2)      1
```

Note that the sparse matrix created called **smatrix** is a Scipy sparse matrix with elements stored in a Coordinate format. But you can convert it into a dense format:

```
1.  smatrix.todense()
2.
3.  matrix([[0, 1, 1, 1],
4.  ........[0, 2, 1, 0]], dtype=int64)
```

Note that the sparse matrix created is the same matrix $M_{|D| \times F}$ we cited earlier in this post, which represents the two document vectors $\vec{v_{d3}}$ and $\vec{v_{d4}}$.

We'll see in the next post how we define the **idf** (*inverse document frequency*) instead of the simple term-frequency, as well how logarithmic scale is used to adjust the measurement of term frequencies according to its importance, and how we can use it to classify documents using some of the well-know machine learning approaches.

I hope you liked this post, and if you really liked, leave a comment so I'll able to know if there are enough people interested in these series of posts in Machine Learning topics.

As promised, here is the second part of this tutorial series.

References

The classic Vector Space Model

The most influential paper Gerard Salton never wrote

Wikipedia: tf-idf

Wikipedia: Vector space model

Scikits.learn Examples

Updates

21 Sep 11 – *fixed some typos and the vector notation*
22 Sep 11 – *fixed import of sklearn according to the new 0.9 release and added the environment section*
02 Oct 11 – *fixed Latex math typos*
18 Oct 11 – *added link to the second part of the tutorial series*
04 Mar 11 – Fixed formatting issues

← *The Interactive Robotic Painting Machine !*
Machine Learning :: Text feature extraction (tf-idf) – Part II →

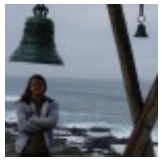
53 thoughts on "Machine Learning :: Text feature extraction (tf-idf) – Part I"

18/09/2011 at 16:19

latex path not specified.
all over the text

justrreadrrr

Reply



★Christian S. Perone

18/09/2011 at 16:22

I'm using the latex from wordpress.com service, for me it is working, maybe they service is down for a while =(thanks for reporting.

Reply

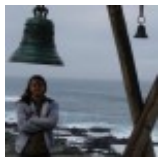
Patrick Durusau

18/09/2011 at 16:49

The link to the *The most influential paper Gerard Salton Never Wrote* fails. Try the cached copy at CiteSeer: The most influential paper Gerard Salton Never Wrote.

Very enjoyable post! I have pointed to it from my blog: http://tm.durusau.net/?p=15199

Reply

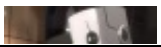


★Christian S. Perone

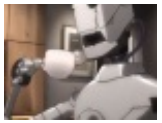
18/09/2011 at 16:56

Thank you Patrick, I'm glad you liked it. I updated the link with the CiteSeer copy.

Reply



18/09/2011 at 22:31



Helio Perroni Filho

Very interesting read. Keep the good work.

↳ Reply

Anand Jeyahar

19/09/2011 at 06:11

Thanks, the mix of actual examples with theory is very handy to see the theory in action and helps retain the theory better. Though in this particular post, i was a little disappointed as i felt it ended too soon. I would like more longer articles. But i guess longer articles turn off majority of the readers.

↳ Reply

dvdgrs

19/09/2011 at 12:06

Very interesting blogpost, I'm sure up for more on the topic :)!

I recently had to handle VSM & TF-IDF in Python too, in a text-processing task of returning most similar strings of an input-string. I haven't looked at scikits.learn, but it sure looks useful and straightforward.

I use Gensim (VSM for human beings: http://radimrehurek.com/gensim/) together with NLTK for preparing the data (aka word tokenizing, lowering words, and removing stopwords). I can highly recommend both libraries!

For some more (slightly out of date) details of my approach, see: http://graus.nu/blog/simple-keyword-extraction-in-python/

Thanks for the post, and looking forward to part II :).

↪ Reply

19/09/2011 at 18:17

Thanks for posting this, would love to see more.

Trey

↪ Reply

20/09/2011 at 04:57

Very well written and interesting!

Johan

↪ Reply

21/09/2011 at 11:22

Thanks for this, most interesting. I look forward to reading your future posts on the subject.

Led

↪ Reply

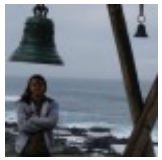
Pingback: Machine Learning :: Text feature extraction (tf-idf) – Part II | Pyevolve

Niu

24/11/2011 at 12:30

It is very useful and easy for start and is well organized. Thanks.

Reply



★Christian S. Perone

24/11/2011 at 13:31

Thanks, I'm glad you liked it.

Reply

baali

21/12/2011 at 07:12

Thanks a lot for this writeup. At times its really good to know what is cooking backstage behind all fancy and magical functions.

Reply

GEETHA r

04/01/2012 at 04:34

Thank you! It is very useful for me to learn about the vector space model. But I am having some doubts, please make me clear.
1. In my work I have added terms,Inverse document frequency. I want to achieve more accuracy. So I want to add some more, please suggest

me…

↪ Reply

alex

10/01/2012 at 15:52

Great post, I will certainly try this out.

I would be interested to see a similar detailed break down on using something like svmlight in conjunction with these techniques.

Thanks!

↪ Reply

Jaques Grobler

26/03/2012 at 10:16

Hey again – my outputs are slighlty different to yours.. Think there may have been changes to the module on scikit-learn.

Will let you know what i find out.

Take care

↪ Reply

27/03/2012 at 06:54

Jaques Grobler

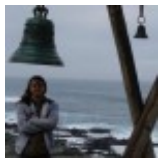
Hello there,
so basically the class feature_selection.text.Vectorizer in Sklearn is now deprecated and replaced by feature_selection.text.TfidfVectorizer.

The whole module has been completely re-factored –
here's the change-log from the Scikit-learn website:
http://scikit-learn.org/dev/whats_new.html

See under 'API changes summary' for what's changed

Just thought I'd give you a headsup about this.
Enjoyed your post regardless!
Take care

↳ Reply



★Christian S. Perone

27/03/2012 at 10:11

Hello Jaques, great thanks for the feedback !

↳ Reply

Anita Mazur

20/04/2012 at 06:36

Thank you for your post. I am currently working on a way how to index documents, but with vocabulary terms taken from a thesaurus in SKOS format.
Your posts are interesting and very helpful to me.

Reply

★Christian S. Perone

20/04/2012 at 15:06

Thanks for the feedback Anita, I'm glad you liked it.

Reply

Zach

06/06/2012 at 16:24

Hey thanks for the very insightful post! I had no idea modules existed in Python that could do that for you (I calculated it the hard way :/)

Just curious did you happen to know about using tf-idf weighting as a feature selection or text categorization method. I've been looking at many papers (most from China for some reason) but am finding numerous ways of approaching this question.

If there's any advice or direction to steer me towards as far as additional resources, that would be greatly appreciated.

Reply

Andres Soto

03/08/2012 at 22:01

Hi
I am using python-2.7.3, numpy-1.6.2-win32-superpack-python2.7, scipy-0.11.0rc1-win32-superpack-python2.7, scikit-learn-0.11.win32-

py2.7
I tried to repeat your steps but I couldn´t print the
vectorizer.vocabulary (see below).
Any suggestions?
Regards
Andres Soto
>>> train_set = (“The sky is blue.”, “The sun is bright.”)
>>> test_set = (“The sun in the sky is bright.”,
“We can see the shining sun, the bright sun.”)
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> vectorizer = CountVectorizer()
>>> print vectorizer
CountVectorizer(analyzer=word, binary=False, charset=utf-8,
charset_error=strict, dtype=, input=content,
lowercase=True, max_df=1.0, max_features=None, max_n=1, min_n=1,
preprocessor=None, stop_words=None, strip_accents=None,
token_pattern=bww+b, tokenizer=None, vocabulary=None)
>>> vectorizer.fit_transform(train_set)
<2×6 sparse matrix of type ''
with 8 stored elements in COOrdinate format>
>>> print vectorizer.vocabulary

Traceback (most recent call last):
File “”, line 1, in
print vectorizer.vocabulary
AttributeError: ‘CountVectorizer’ object has no attribute ‘vocabulary’
>>>

Reply

Andres Soto

06/08/2012 at 14:32

I tried to fix the parameters of CountVectorizer (analyzer = WordNGramAnalyzer, vocabulary = dict) but it didn't work. Therefore I decided to install sklearn 0.9 and it works, so we could say that everything is OK but I still would like to know what is wrong with version sklearn 0.11

Reply

★Christian S. Perone

06/08/2012 at 17:48

Hello Andres, what I know is that this API has changed a lot on the sklearn 0.10/0.11, I heard some discussions about these changes but I can't remember where right now.

Reply

Gavin Igor

16/08/2012 at 23:41

Thanks for the great overview, looks like the part 2 link is broken. It would be great if you could fix it. Thank You.

Reply

17/08/2012 at 09:55

Thanks for the feedback Gavin, the link is ok, it seems that the

★Christian S. Perone

problem is sourceforge hosting that is throwing some errors.

↳ Reply

Gavin Igor

24/08/2012 at 17:48

I am using a mac and running 0.11 version but I got the following error I wonder how i change this according to the latest api

>> train_set
(‘The sky is blue.’, ‘The sun is bright.’)
>>> vectorizer.fit_transform(train_set)
<2×6 sparse matrix of type ''
with 8 stored elements in COOrdinate format>
>>> print vectorizer.vocabulary
Traceback (most recent call last):
File “”, line 1, in
AttributeError: ‘CountVectorizer’ object has no attribute ‘vocabulary’
>>> vocabulary

↳ Reply

creativega

03/10/2012 at 05:18

Hello, Mr. Perone! Thank you very much, I’m newbie in TF-IDF and your posts have helped me a lot to understand it. Greetings from Japan^^

↳ Reply

★Christian S. Perone

03/10/2012 at 16:21

Great thanks for the feedback, I'm very glad you liked and that the post helped you !

↳ Reply

Mohit

20/10/2012 at 08:18

wonderful post… It helps me to understand VSM concept 🙂

↳ Reply

Ryan

30/01/2013 at 04:08

Thanks for this awesome post! Eminently readable introduction to the topic.

↳ Reply

Thomas

15/09/2013 at 07:42

very well written. Good example presented in a form that makes it easy to follow and understand. Curious now to read more…
Thank you for sharing your knowledge
Thomas, Germany

↳ Reply

★Christian S. Perone

15/09/2013 at 12:47

Thanks Thomas, I appreciate your feedback.

↳ Reply

UA

17/09/2013 at 16:59

Hi. I am having trouble understanding how to compute tf-idf weights for a text file I have which contains 300k lines of text. Each line is considered as a document. Example excerpt from the text file:

hacking hard
jeetu smart editor
shyamal vizualizr
setting demo hacks
vivek mans land
social routing guys minute discussions learn photography properly
naseer ahmed yahealer
sridhar vibhash yahoo search mashup
vaibhav chintan facebook friend folio
vaibhav chintan facebook friend folio
judgess comments
slickrnot

I'm pretty confused as to what I should do. Thanks

↳ Reply

Nishant

08/10/2013 at 02:49

Thanks. It was helpful. Was looking for a good Python Vectorizer tutorial.

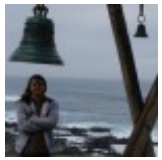
↳ Reply

Igor

09/10/2013 at 15:59

That's really interesting post, thanks a lot!

↳ Reply



★Christian S. Perone

15/10/2013 at 12:56

Thanks for the feedback Igor.

↳ Reply

Navdeep

08/11/2013 at 08:30

Really helpful post

↳ Reply

06/12/2013 at 07:02

Haneef

Really a very good effort in explaining in such a simple way.

↩ Reply

Williame Rocha

20/01/2014 at 18:19

it is a very good text.

Thanks for explanation.

↩ Reply

need info

10/03/2014 at 15:25

Its giving idf vector as zero if test is same as train. why???

↩ Reply

chamso

10/04/2014 at 18:24

thank you very much i encourage you to continue this is very helpful post <3

↩ Reply

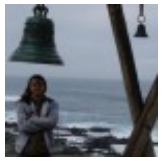
05/05/2014 at 11:49

Matthieu

Thanks
A well written clear explanation

Definitely a reference when taking the first steps in text mining.

↳ Reply



★Christian S. Perone

06/05/2014 at 09:56

Thank you Matthieu !

↳ Reply

purna

27/06/2014 at 01:26

Good post with example

↳ Reply

Hannah

30/07/2014 at 11:21

thank you very much. this post helped me alot. very well written and clear explanation.
Hannah

↳ Reply

Tim I.

02/09/2014 at 08:04

Awesome stuff. I really appreciate the simplicity and clarity of the information. A great, great help.

↳ Reply

★Christian S. Perone

02/09/2014 at 10:29

Great thanks Tim, I'm glad you liked it.

↳ Reply

Ganesh

13/11/2014 at 13:48

Thanks for the great post. You have explained it in simple words, so that a novice like me can understand. Will move on to read the next part!

↳ Reply

Usher

25/03/2015 at 16:39

Solution to question of Andres and Gavin:

>>> print vectorizer.vocabulary_

(with underscore at the end in new versions of scikit!)

will output:

{u’blue': 0, u’bright': 1, u’sun': 4, u’is': 2, u’sky': 3, u’the': 5}

↪ Reply

sanja7s

01/06/2015 at 09:17

Thanks, a great one and useful!

↪ Reply

jpf

24/06/2015 at 20:16

” … you can simple select …” -> “>>> you can simply select …”

↪ Reply

Leave a Reply

Your email address will not be published.

Name

Email

Website

Comment

You may use these HTML tags and attributes: <a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <i> <q cite=""> <s> <strike>

Post Comment

Search …

Tags

3d announce aqua arduino Article benford's law bitcoin C c++0x comparative evolution evolutionary algorithms ga genetic algorithm Genetic

Meta

Log in

Entries RSS

Comments RSS

WordPress.org

Algorithms genetic programming jit jython machine learning matplotlib modis News Philosophy programming Pyevolve Python raspberry pi rpi Science sigevolution sklearn sony psp stallion statistics svn terra text mining tf-idf tsp twitter ubigraph usgs vector space model visualization vsm