



BLOG

Text analysis in Pandas with some TF-IDF (again)


POSTED BY JAKUB NOWACKI, 18 SEPTEMBER 2017

Pandas (<http://pandas.pydata.org>) is a great tool for the analysis of tabular data via its DataFrame interface. Slightly less known are its capabilities for working with text data. In this post I'll present them on some simple examples. As a comparison I'll use my previous post about TF-IDF in Spark (</blog/word-count-in-spark-with-a-pinch-of-tf-idf>).

```
# Some used imports
%matplotlib inline
import pandas as pd
import numpy as np
import os
import glob
import matplotlib as mpl

# Just making the plots look better
mpl.style.use('ggplot')
mpl.rcParams['figure.figsize'] = (8,6)
mpl.rcParams['font.size'] = 12
```

First we load data, i.e. books previously (</blog/word-count-in-spark-with-a-pinch-of-tf-idf>) downloaded from Project Gutenberg site (<https://www.gutenberg.org/>). Here we use a handy `glob` module (<https://docs.python.org/3/library/glob.html>) to walk over text files in a directory

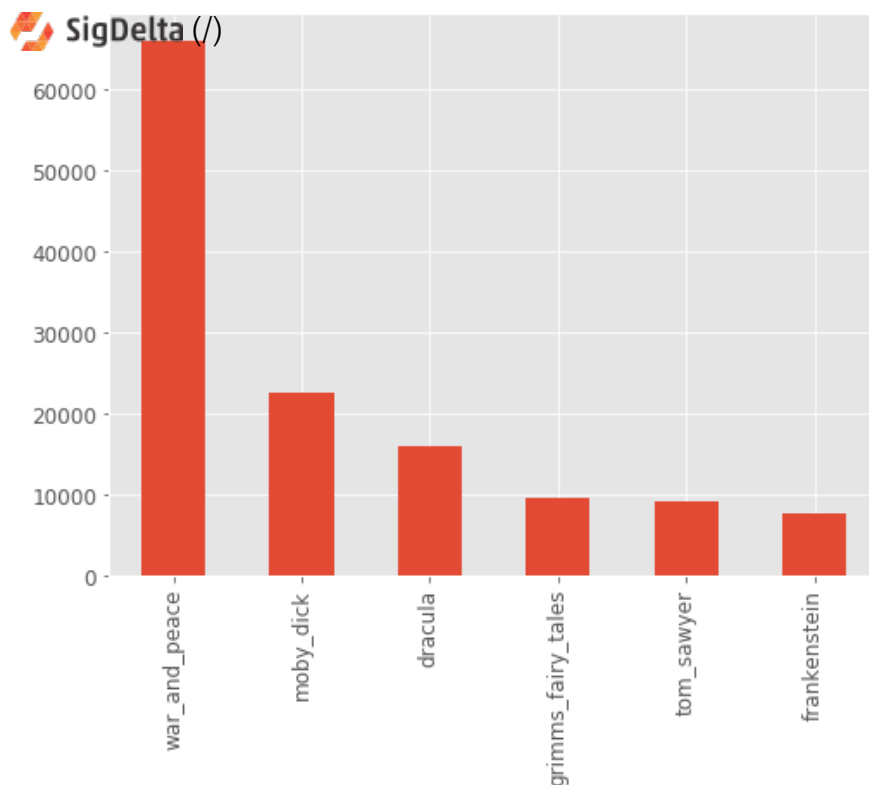
 and read in files line by line into DataFrame.

```
books = glob.glob('data/*.txt')
d = list()
for book_file in books:
    with open(book_file, encoding='utf-8') as f:
        book = os.path.basename(book_file.split('.')[0])
        d.append(pd.DataFrame({'book': book, 'lines': f.readlines()}))
doc = pd.concat(d)
doc.head()
```

	book	lines
0	dracula	The Project Gutenberg EBook of Dracula, by Br...
1	dracula	\n
2	dracula	This eBook is for the use of anyone anywhere a...
3	dracula	almost no restrictions whatsoever. You may co...
4	dracula	re-use it under the terms of the Project Guten...

Since we're now in Pandas, we can easily use its plotting capability (<http://pandas.pydata.org/pandas-docs/stable/visualization.html>) to look at the number of lines in the books. Note that `value_counts` of books create `Series` indexed by the book names, thus, we don't need to set index for plotting. Note that we can now choose plot styling; see demo page (https://matplotlib.org/examples/style_sheets/style_sheets_reference.html) for available styles.

```
doc['book'].value_counts().plot.bar();
```



Now, we need to tokenize the sentences into words aka terms. While we can do it in a loop, we can take advantage of the split function in the text toolkit for Pandas' Series; see this manual (<http://pandas.pydata.org/pandas-docs/stable/text.html>) for all the functions. To get it we just invoke the `strip` function, which is a part of `str`, i.e. `StringsMethods` object. The argument is regular expression pattern, on which the string, in our case the sentence, will be split. In our case it is the regular expression set of everything that is not a letter (capital or not) or digit and underscore. This is because `\W` is the reverse of `\w` which is a set `[A-Za-z0-9_]`, i.e. contains an underscore. In this case we want also to split on underscore so we had to add it to the splitting set. Note that this decision and other you'll make while deciding on the splitting set, will influence the tokenization, and thus the result. Hence, select the splitting pattern carefully. If you're not sure about your decision, use practicing pages, like regex101.com (<https://regex101.com/>) to check your patterns.

```
doc['words'] = doc.lines.str.strip().str.split('[\W_]+')
doc.head()
```

	book	lines	words
0	dracula	The Project Gutenberg EBook of Dracula, by Br...	[, The, Project, Gutenberg, EBook, of, Dracula...
1	dracula	\n	[]

	book	lines	words
0	dracula	This eBook is for the use of anyone	[This, eBook, is, for, the, use, of,
1	dracula	anywhere a...	anyone, a...
2	dracula	almost no restrictions whatsoever.	[almost, no, restrictions, whatsoever,
3	dracula	You may co...	You, ma...
4	dracula	re-use it under the terms of the	[re, use, it, under, the, terms, of, the,
5	dracula	Project Guten...	Proj...

As a result we get a new column with array of words. Now we need to flatten the resulted DataFrame somehow. The best approach is to use iterators and create new DataFrame with words; see this [StackOverflow post](https://stackoverflow.com/questions/38428796/how-to-do-lateral-view-explode-in-pandas) (<https://stackoverflow.com/questions/38428796/how-to-do-lateral-view-explode-in-pandas>) for the performance details of this solution. Note that this may take a few seconds depending on the machine; on mine it takes about 25s.

```
rows = list()
for row in doc[['book', 'words']].iterrows():
    r = row[1]
    for word in r.words:
        rows.append((r.book, word))

words = pd.DataFrame(rows, columns=['book', 'word'])
words.head()
```

	book	word
0	dracula	
1	dracula	The
2	dracula	Project
3	dracula	Gutenberg
4	dracula	EBook

Now we have DataFrame with words, but occasionally we get an empty string as a byproduct of the splitting. We should remove it as it would be counted as a term. To do that we can use function `len` as follows:

```
words = words[words.word.str.len() > 0]
words.head()
```

	book	word
1	dracula	The
2	dracula	Project

	book	word
3	dracula	Gutenberg
4	dracula	EBook
5	dracula	of

The *new words* DataFrame is now free of the empty strings.

If we want to calculate TF-IDF statistic we need to normalize the words by bringing them all to the same case. Again, we can use a `Series` string function for that.

```
words['word'] = words.word.str.lower()
words.head()
```

	book	word
1	dracula	the
2	dracula	project
3	dracula	gutenberg
4	dracula	ebook
5	dracula	of


First, lets calculate counts of the terms per book. We can do it as follows:

```
counts = words.groupby('book')\
    .word.value_counts()\
    .to_frame()\
    .rename(columns={'word': 'n_w'})
counts.head()
```

		n_w
book	word	
dracula	the	8093
	and	5976
	i	4847
	to	4745
	of	3748

Note that as a result we are getting a hierarchical index aka MultiIndex; see Pandas manual (<http://pandas.pydata.org/pandas-docs/stable/advanced.html>) for more details.

With this index we can now plot the results in much nicer way. E.e. we can get 5 words largest counts per book and plot it as shown below. Note that we need to reset, i.e. remove one level of index, as it doubles when we are using

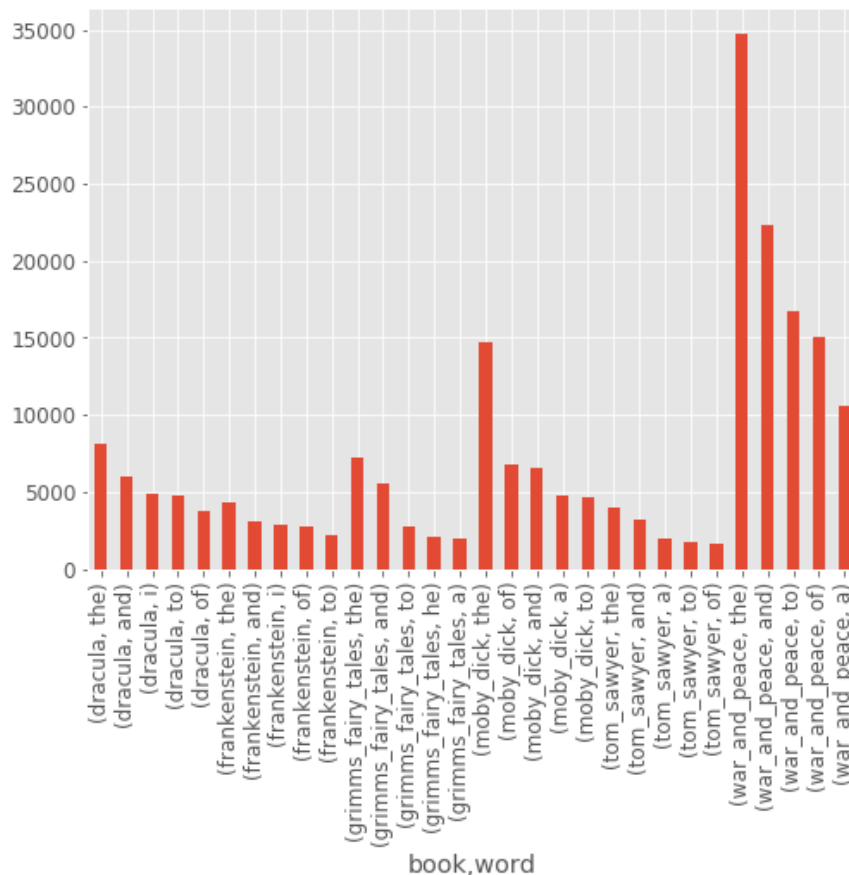
 largest function. I've made the process into a function, as I will reuse it further.

```
def pretty_plot_top_n(series, top_n=5, index_level=0):
    r = series\
        .groupby(level=index_level)\
        .nlargest(top_n)\
        .reset_index(level=index_level, drop=True)
    r.plot.bar()
    return r.to_frame()
```

```
pretty_plot_top_n(counts['n_w'])
```

		n_w
book	word	
dracula	the	8093
	and	5976
	i	4847
	to	4745
	of	3748
frankenstein	the	4371
	and	3046
	i	2850
	of	2760
	to	2174
grimms_fairy_tales	the	7224
	and	5551
	to	2749
	he	2096
	a	1978
moby_dick	the	14718
	of	6743
	and	6518
	a	4807
	to	4707
tom_sawyer	the	3973
	and	3193
	a	1955
	to	1807

book		n_w
	word	
	of	1585
war_and_peace	the	34725
	and	22307
	to	16755
	of	15008
	a	10584




To finish the TF as shown in the previous post (</blog/word-count-in-spark-with-a-pinch-of-tf-idf/>), we need the counts of the books to get the measure. As a reminder, below is the equation for TF:

$$TF = \frac{n_w}{n_d},$$

Note that I've renamed the column as it inherits the name by default.

```
word_sum = counts.groupby(level=0)\
    .sum()\
    .rename(columns={'n_w': 'n_d'})
word_sum
```

 SigDelta (/)	n_d
book	
dracula	166916
frankenstein	78475
grimms_fairy_tales	105336
moby_dick	222630
tom_sawyer	77612
war_and_peace	576627

Now we need to join both columns on book to get the sum of word per book into final DataFrame. Having both `n_w` and `n_d`, we can easily calculate TF as follows:

```
tf = counts.join(word_sum)

tf['tf'] = tf.n_w/tf.n_d


tf.head()
```

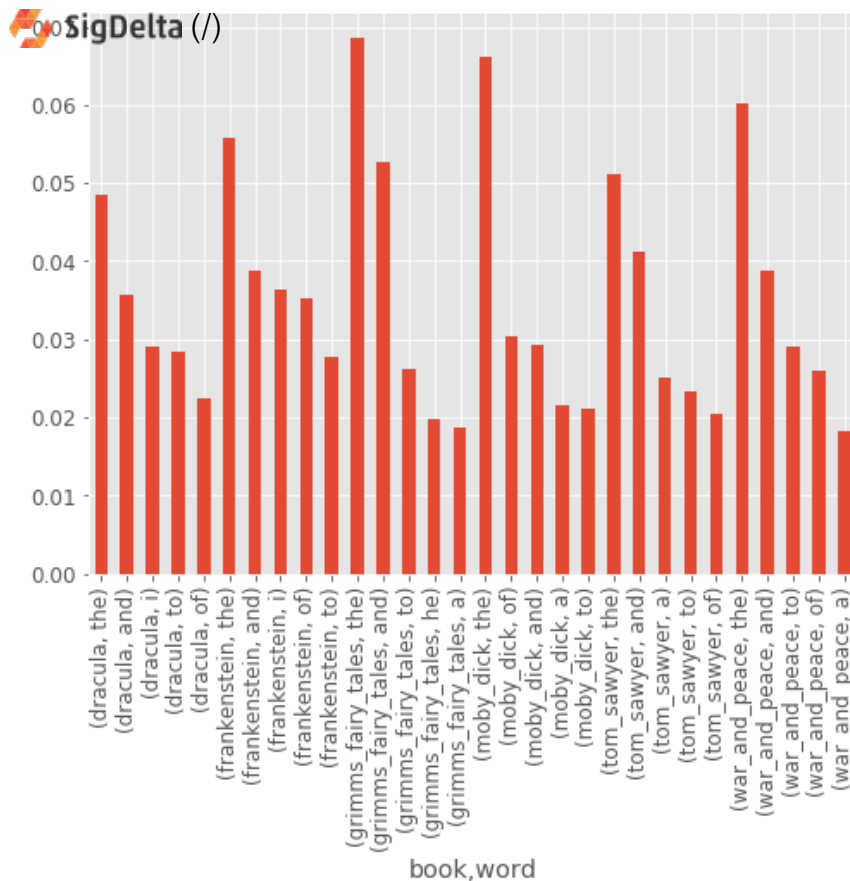
		n_w	n_d	tf
book	word			
dracula	the	8093	166916	0.048485
	and	5976	166916	0.035802
	i	4847	166916	0.029039
	to	4745	166916	0.028427
	of	3748	166916	0.022454

Again, lets look at the top 5 words with respect to TF.

```
pretty_plot_top_n(tf['tf'])
```

		tf
book	word	
dracula	the	0.048485
	and	0.035802
	i	0.029039
	to	0.028427
	of	0.022454
frankenstein	the	0.055699
	and	0.038815
	i	0.036317
	of	0.035170

 SigDelta (/)		tf
book	word	
	to	0.027703
grimms_fairy_tales	the	0.068581
	and	0.052698
	to	0.026097
	he	0.019898
	a	0.018778
moby_dick	the	0.066110
	of	0.030288
	and	0.029277
	a	0.021592
	to	0.021143
tom_sawyer	the	0.051191
	and	0.041141
	a	0.025189
	to	0.023282
	of	0.020422
war_and_peace	the	0.060221
	and	0.038685
	to	0.029057
	of	0.026027
	a	0.018355



As before (and as expected), we've got the English stop-words.

Now we can do IDF; the remainder of the formula is given below:


$$IDF = \log\left(\frac{c_d}{i_d}\right),$$

First we need to get the document count. The simplest solution is to use the Series' method `nunique`, i.e. get size of set of unique elements in a series.

```
c_d = words.book.nunique()
c_d
```

6

Similarly, to get the number of unique books that every term appeared in, we can use the same method but on a grouped data. Again we need to rename the column. Note that sorting values is only for the presentation and it is not needed for the further computations.



```
idf = words.groupby('word')\
    .book\
    .nunique()\
    .to_frame()\
    .rename(columns={'book': 'i_d'})\
    .sort_values('i_d')
idf.head()
```

	i_d
word	
laplandish	1
moluccas	1
molten	1
molliten	1
molière	1

Having all the components of the IDF formula, we can now calculate it as a new column.

```
idf['idf'] = np.log(c_d/idf.i_d.values)

idf.head()
```


	i_d	idf
word		
laplandish	1	1.791759
moluccas	1	1.791759
molten	1	1.791759
molliten	1	1.791759
molière	1	1.791759

To get the final DataFrame we need to join both DataFrames as follows:

```
tf_idf = tf.join(idf)

tf_idf.head()
```

		n_w	n_d	tf	i_d	idf
book	word					
dracula	the	8093	166916	0.0484856		0.0
	and	5976	166916	0.0358026		0.0
	i	4847	166916	0.0290396		0.0



		n_w	n_d	tf	i_d	idf
book	word					
	to	4745	166916	0.0284276		0.0
	of	3748	166916	0.0224546		0.0

Having now DataFrame with both TF and IDF values, we can calculate TF-IDF statistic.

```
tf_idf['tf_idf'] = tf_idf.tf * tf_idf.idf
tf_idf.head()
```

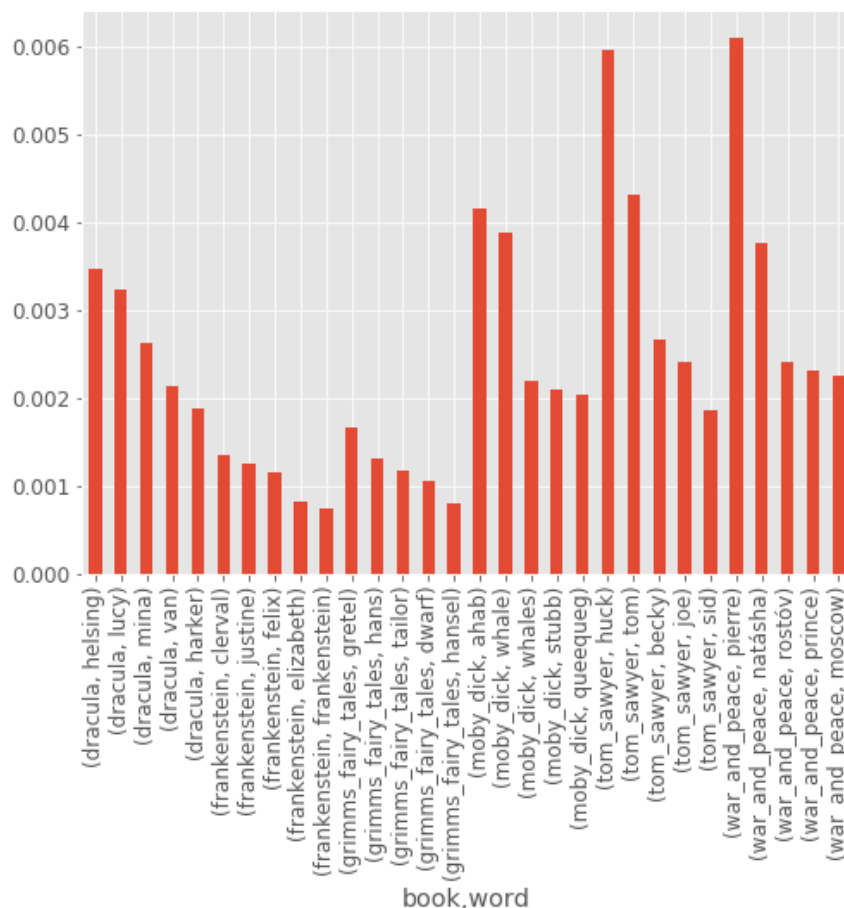
		n_w	n_d	tf	i_d	idf	tf_idf
book	word						
dracula	the	8093	166916	0.0484856		0.0	0.0
	and	5976	166916	0.0358026		0.0	0.0
	i	4847	166916	0.0290396		0.0	0.0
	to	4745	166916	0.0284276		0.0	0.0
	of	3748	166916	0.0224546		0.0	0.0

Again, lets see the top TF-IDF terms:

```
pretty_plot_top_n(tf_idf['tf_idf'])
```

		tf_idf
book	word	
dracula	helsing	0.003467
	lucy	0.003231
	mina	0.002619
	van	0.002126
	harker	0.001879
frankenstein	clerval	0.001347
	justine	0.001256
	felix	0.001142
	elizabeth	0.000813
	frankenstein	0.000731
grimms_fairy_tales	gretel	0.001667
	hans	0.001304
	tailor	0.001174
	dwarf	0.001055
	hansel	0.000799
moby_dick	ahab	0.004161
	whale	0.003873

SigDelta (1)		tf_idf
book	word	
	whales	0.002181
	stubb	0.002101
	queequeg	0.002036
tom_sawyer	huck	0.005956
	tom	0.004305
	becky	0.002655
	joe	0.002406
	sid	0.001847
war_and_peace	pierre	0.006100
	natasha	0.003769
	rostov	0.002411
	prince	0.002318
	moscow	0.002243



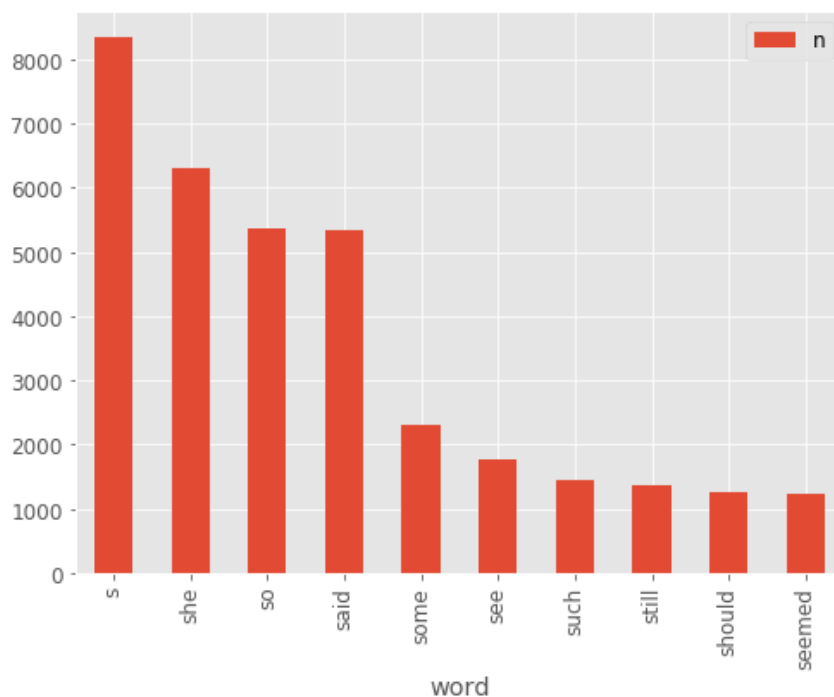
As before we've got a set of important words for the given document.

Note that this more of a demo of Pandas text processing. If you're considering using TF-IDF in a more production example, see some existing solutions like scikit-learn's TfidfVectorizer (http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html).

Note that I've just scratched a surface with the Pandas' text processing capabilities. There are many other useful functions like the `match` function shown below:

```
r = words[words.word.str.match('^s')]\
    .groupby('word')\
    .count()\
    .rename(columns={'book': 'n'})\
    .nlargest(10, 'n')\
r.plot.bar()\nr
```

	n
word	
s	8341
she	6317
so	5380
said	5337
some	2317
see	1760
such	1456
still	1368
should	1264
seemed	1233



Again, I encourage you to check Pandas manual () for more details.

Notebook with the above computations is available for download here (/assets/notebooks/2017-09-14-text-analysis-in-pandas.ipynb).

1 Comment

SigDelta

1 Login

Recommend

Tweet

Share

Sort by Best



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?



Name



Robert FC • 5 months ago

Wow, thank you so much. This was hugely informative

I am mostly an R user, and had struggled to find resources in Python that mimicked the TidyText method

^ | v • Reply • Share ›

ALSO ON SIGDELTA

Using language-detector aka large not serializable objects in Spark | SigDelta - ...

1 comment • a year ago

Chandan Kumar — I followed the whole example but the below statement still gives java.io.NotSerializableException exception. ...

How to install PySpark locally | SigDelta - data analytics, big data and machine

1 comment • a year ago

Joshua Mitchell — # Start pyspark via provided commandWhere is the provided command again? I might be blind but I don't see it.EDIT: is

[Subscribe](#) [Add Disqus to your site](#) [Add Disqus](#) [Disqus' Privacy Policy](#) [Privacy Policy](#) [Privacy Policy](#) [Privacy Policy](#)

LATEST BLOG POSTS

- StackOverflow tags with Dask (/blog/stackpverflow-tags-with-dask/)
- Dask - a new elephant in the room (/blog/dask-introduction/)
- Scala (and Java) Spark UDFs in Python (/blog/scala-spark-udfs-in-python/)
- Using language-detector aka large not serializable objects in Spark (/blog/using-language-detector-aka-large-not-serializable-objects-in-spark/)
- Text analysis in Pandas with some TF-IDF (again) (/blog/text-analysis-in-pandas/)
- Why SQL? Notes from (big) data analysis practice (/blog/why-sql/)
- Word count is Spark SQL with a pinch of TF-IDF (continued) (/blog/word-count-in-spark-with-a-pinch-of-tf-idf-continued/)
- Word count is Spark SQL with a pinch of TF-IDF (/blog/word-count-in-spark-with-a-pinch-of-tf-idf/)
- Docker for Data Science (/blog/docker-for-data-science/)
- How to install PySpark locally (/blog/how-to-install-pyspark-locally/)



- Power BI - Self-service Business Intelligence tool (/blog/power-bi-self-service-business-intelligence-tool/)
- Drools far beyond Hello World (/blog/drools-far-beyond-hellow-world/)
- Big Data in the Maturity Stage (/blog/big-data-in-the-maturity-stage/)

CONTACT

How can we help?

Drop us a line and we'll respond as soon as possible.

DROP US A LINE

Title

Email *

Your message

Submit



Powered by Cognito Forms. (<https://www.cognitoforms.com/?crs=cmVmchVibGljOjpTYWdlczE=>)

CALL US

(+48) 22 203 56 00

Nowogrodzka 62C, Warsaw, Poland

office@sigdelta.com

UTC / GMT +1



GET SOCIAL

(<https://www.facebook.com/SigDelta-1786388271624386>)

(<https://twitter.com/sigdeltacom>)

(<https://www.linkedin.com/organization/15185369>)