Fuzzy_Systems_Twitter_V2

April 27, 2021

1 Computational Intelligence Project: Sentiment Analysis on IMDB dataset Using Fuzzy System on Twitter

Musab - 19030008

Sentiment analysis is one of the key areas of research in NLP and Sequence modelling. I will be using fuzzy systems to predict two classes - positive or negative sentiment.

[ ]:

```
import re
import os
import pandas as pd
import numpy as np
!pip install -U scikit-fuzzy
import skfuzzy as fuzz
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import time
from keras.datasets import imdb
import matplotlib.pyplot as plt
```

```
Collecting scikit-fuzzy
  Downloading https://files.pythonhosted.org/packages/6c/f0/5eb5dbe0fd8dfe
7d4651a8f4e591a196623a22b9e5339101e559695b4f6c/scikit-fuzzy-0.4.2.tar.gz (993kB)
     || 1.0MB 11.2MB/s
Requirement already satisfied, skipping upgrade: numpy>=1.6.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-fuzzy) (1.19.5)
Requirement already satisfied, skipping upgrade: scipy>=0.9.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-fuzzy) (1.4.1)
Requirement already satisfied, skipping upgrade: networkx>=1.9.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-fuzzy) (2.5)
Requirement already satisfied, skipping upgrade: decorator>=4.3.0 in
/usr/local/lib/python3.7/dist-packages (from networkx>=1.9.0->scikit-fuzzy)
(4.4.2)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-cp37-none-any.whl
size=894069
```

```
sha256=03c75ed98061b6196825bfa4796a5d61908425a62c604aad2fee1370563d3581
  Stored in directory: /root/.cache/pip/wheels/b9/4e/77/da79b16f64ef1738d95486e2
731eea09d73e90a72465096600
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...

/usr/local/lib/python3.7/dist-packages/nltk/twitter/__init__.py:20: UserWarning:
The twython library has not been installed. Some functionality from the twitter
package will not be available.
  warnings.warn("The twython library has not been installed. "
```

1.1 Loading Dataset

[ ]:

```
# from google.colab import drive
# drive.mount('/content/drive')

# dataset = "twitter-sanders-apple3.csv"
# drive_path = '/content/drive/My Drive/Computational Intelligence/'
# path = os.path.join(drive_path, dataset)
df = pd.read_csv("twitter-sanders-apple3.csv")
```

[ ]:

```
start = time.time()

traindata = df
doc = traindata.text
print(len(doc))
sentidoc = traindata['class']
```

```
988
```

[ ]:

```
df.head
```

[ ]:

```
<bound method NDFrame.head of        class
text
0        Pos  Now all @Apple has to do is get swype on the i...
1        Pos  @Apple will be adding more carrier support to ...
2        Pos  Hilarious @youtube video - guy does a duet wit...
3        Pos  @RIM you made it too easy for me to switch to ...
4        Pos  I just realized that the reason I got into twi...
..       ...                                                ...
983  Neutral  @vlingo is a POOR substitute for Siri!! Yo @AP...
984  Neutral                                @Apple Scrapple. (:
985  Neutral  @tvnewschick @apple Oh no! Why not?! I want it...
986  Neutral  One of the great #entrepreneurs has died. #Ste...
987  Neutral  @fashionNOGuilt haha! tomorrow should be less ...
```

```
[988 rows x 2 columns]>
```

[ ]:

```
# Generate variables
x_p = np.arange(0, 1, 0.1)
x_n = np.arange(0, 1, 0.1)
x_op = np.arange(0, 10, 1)
```

1.2 Membership Functions

[ ]:

```
# Generate fuzzy membership functions
p_lo = fuzz.trimf(x_p, [0, 0, 0.5])
p_md = fuzz.trimf(x_p, [0, 0.5, 1])
p_hi = fuzz.trimf(x_p, [0.5, 1, 1])
n_lo = fuzz.trimf(x_n, [0, 0, 0.5])
n_md = fuzz.trimf(x_n, [0, 0.5, 1])
n_hi = fuzz.trimf(x_n, [0.5, 1, 1])
op_Neg = fuzz.trimf(x_op, [0, 0, 5])
op_Neu = fuzz.trimf(x_op, [0, 5, 10])
op_Pos = fuzz.trimf(x_op, [5, 10, 10])
```

1.2.1 Visualization of Memversship Functions

[ ]:

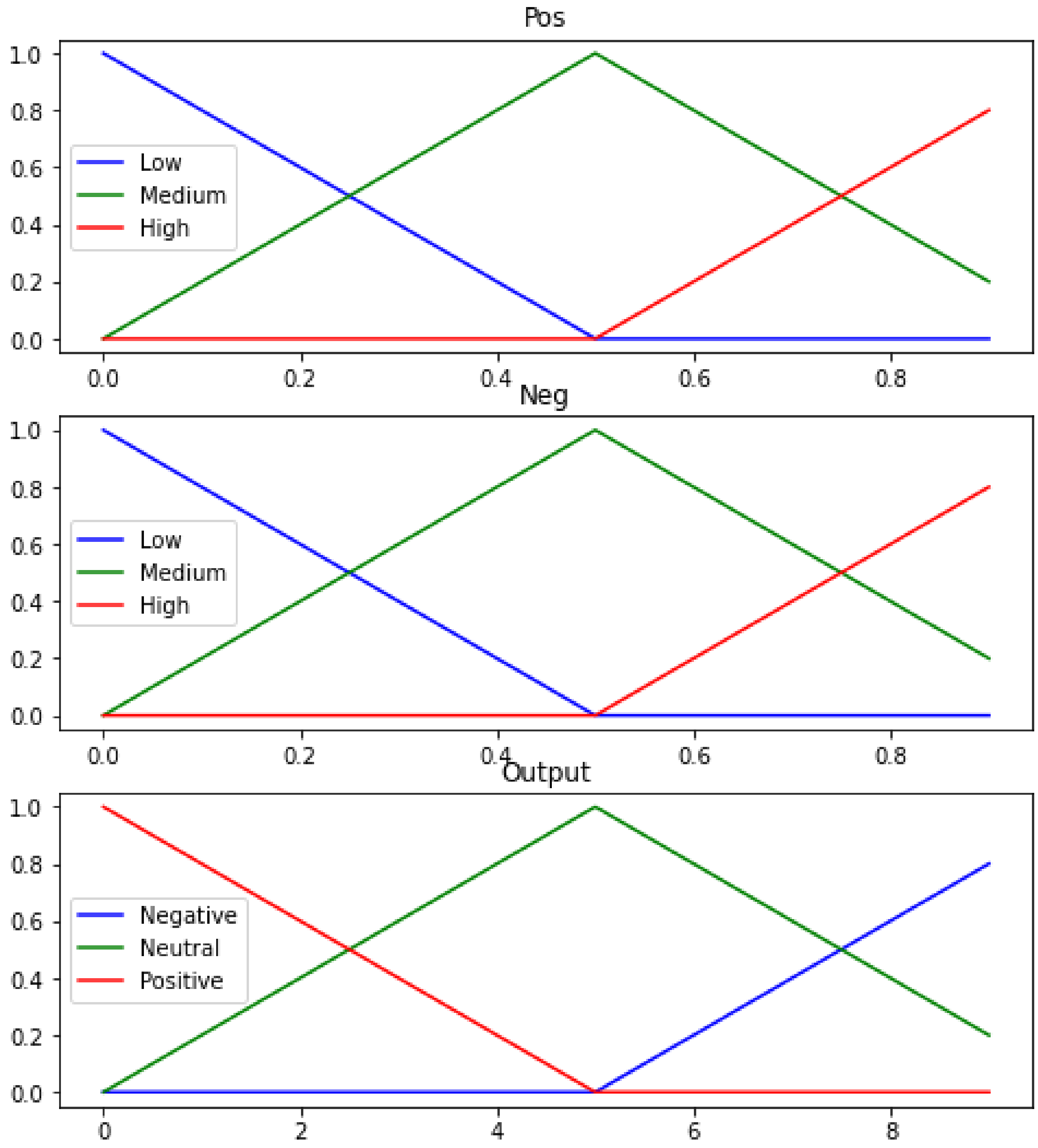
```
# Visualize these universes and membership functions
fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(8, 9))
#
ax0.plot(x_p, p_lo, 'b', linewidth=1.5, label='Low')
ax0.plot(x_p, p_md, 'g', linewidth=1.5, label='Medium')
ax0.plot(x_p, p_hi, 'r', linewidth=1.5, label='High')
ax0.set_title('Pos')
ax0.legend()

ax1.plot(x_n, n_lo, 'b', linewidth=1.5, label='Low')
ax1.plot(x_n, n_md, 'g', linewidth=1.5, label='Medium')
ax1.plot(x_n, n_hi, 'r', linewidth=1.5, label='High')
ax1.set_title('Neg')
ax1.legend()

ax2.plot(x_op, op_Pos, 'b', linewidth=1.5, label='Negative')
ax2.plot(x_op, op_Neu, 'g', linewidth=1.5, label='Neutral')
ax2.plot(x_op, op_Neg, 'r', linewidth=1.5, label='Positive')
ax2.set_title('Output')
ax2.legend()
```

[ ]:

```
<matplotlib.legend.Legend at 0x7f5ceed90fd0>
```



1.3 Preprocessing

[ ]:

```
tweets=[]
senti=[]
sentiment=[]
sentiment_doc=[]

for j in range(len(doc)):
    str1=traindata.text[j]
    str2=str1.lower()
    tweets.append(str2)    # converted into lower case
    senti.append(traindata['class'][j])

def decontracted(phrase):    # text pre-processing
        # specific
        phrase = re.sub(r"won't", "will not", phrase)
        phrase = re.sub(r"can\'t", "can not", phrase)
        phrase = re.sub(r"@", "" , phrase)          # removal of @
        phrase =  re.sub(r"http\S+", "", phrase)    # removal of URLs
        phrase = re.sub(r"#", "", phrase)           # hashtag processing

        # general
        phrase = re.sub(r"n\'t", " not", phrase)
        phrase = re.sub(r"\'re", " are", phrase)
        phrase = re.sub(r"\'s", " is", phrase)
        phrase = re.sub(r"\'d", " would", phrase)
        phrase = re.sub(r"\'ll", " will", phrase)
        phrase = re.sub(r"\'t", " not", phrase)
        phrase = re.sub(r"\'ve", " have", phrase)
        phrase = re.sub(r"\'m", " am", phrase)
        return phrase

for k in range(len(doc)):
    tweets[k]=decontracted(tweets[k])

sid = SentimentIntensityAnalyzer()
```

[ ]:

```
len(sentiment_doc), len(sentiment)
```

[ ]:

```
(0, 0)
```

1.4 Output

[ ]:

```
for j in range(len(doc)):
  sentiment_doc.append(senti[j])
  ss = sid.polarity_scores(tweets[j])
  posscore=ss['pos']
  negscore=ss['neg']
  neuscore=ss['neu']
  compoundscore=ss['compound']

  print(str(j+1)+" {:-<65} {}".format(tweets[j], str(ss)))

  print("\nPositive Score for each  tweet :")
  if (posscore==1):
    posscore=0.9
```

```python
else:
  posscore=round(posscore,1)
print(posscore)

print("\nNegative Score for each  tweet :")
if (negscore==1):
  negscore=0.9
else:
  negscore=round(negscore,1)
print(negscore)

# We need the activation of our fuzzy membership functions at these values.
p_level_lo = fuzz.interp_membership(x_p, p_lo, posscore)
p_level_md = fuzz.interp_membership(x_p, p_md, posscore)
p_level_hi = fuzz.interp_membership(x_p, p_hi, posscore)

n_level_lo = fuzz.interp_membership(x_n, n_lo, negscore)
n_level_md = fuzz.interp_membership(x_n, n_md, negscore)
n_level_hi = fuzz.interp_membership(x_n, n_hi, negscore)

# Now we take our rules and apply them. Rule 1 concerns bad food OR nice.
# The OR operator means we take the maximum of these two.
active_rule1 = np.fmin(p_level_lo, n_level_lo)
active_rule2 = np.fmin(p_level_md, n_level_lo)
active_rule3 = np.fmin(p_level_hi, n_level_lo)
active_rule4 = np.fmin(p_level_lo, n_level_md)
active_rule5 = np.fmin(p_level_md, n_level_md)
active_rule6 = np.fmin(p_level_hi, n_level_md)
active_rule7 = np.fmin(p_level_lo, n_level_hi)
active_rule8 = np.fmin(p_level_md, n_level_hi)
active_rule9 = np.fmin(p_level_hi, n_level_hi)

# Now we apply this by clipping the top off the corresponding output
# membership function with `np.fmin`

n1=np.fmax(active_rule4,active_rule7)
n2=np.fmax(n1,active_rule8)
op_activation_lo = np.fmin(n2,op_Neg)

neu1=np.fmax(active_rule1,active_rule5)
neu2=np.fmax(neu1,active_rule9)
op_activation_md = np.fmin(neu2,op_Neu)

p1=np.fmax(active_rule2,active_rule3)
p2=np.fmax(p1,active_rule6)
op_activation_hi = np.fmin(p2,op_Pos)
```

```
  op0 = np.zeros_like(x_op)

  # Aggregate all three output membership functions together
  aggregated = np.fmax(op_activation_lo,
                       np.fmax(op_activation_md, op_activation_hi))

  # Calculate defuzzified result
  op = fuzz.defuzz(x_op, aggregated, 'centroid')
  output=round(op,2)

  op_activation = fuzz.interp_membership(x_op, aggregated, op)  # for plot

  # Visualize Aggregated Membership
  fig, ax0 = plt.subplots(figsize=(8, 3))

  ax0.plot(x_op, op_Neg, 'b', linewidth=0.5, linestyle='--',label= 'Negative')
  ax0.plot(x_op, op_Pos, 'r', linewidth=0.5, linestyle='--',label= 'Positive')
  ax0.fill_between(x_op, op0, aggregated, facecolor='Orange', alpha=0.7)
  ax0.plot([op, op], [0, op_activation], 'k', linewidth=1.5, alpha=0.9)
  ax0.set_title('Aggregated membership and result (line)')
  ax0.legend()

  # Turn off top/right axes
  for ax in (ax0,):
      ax.spines['top'].set_visible(False)
      ax.spines['right'].set_visible(False)
      ax.get_xaxis().tick_bottom()
      ax.get_yaxis().tick_left()

  plt.tight_layout()

  # Visualize Output Membership
  fig, ax0 = plt.subplots(figsize=(8, 3))

  ax0.fill_between(x_op, op0, op_activation_lo, facecolor='b', alpha=0.7)
  ax0.plot(x_op, op_Neg, 'b', linewidth=0.5, linestyle='--',label= 'Negative' )
  ax0.fill_between(x_op, op0, op_activation_md, facecolor='g', alpha=0.7)
  ax0.plot(x_op, op_Neu, 'g', linewidth=0.5, linestyle='--', label='Neutral')
  ax0.fill_between(x_op, op0, op_activation_hi, facecolor='r', alpha=0.7)
  ax0.plot(x_op, op_Pos, 'r', linewidth=0.5, linestyle='--', label='Positive')
  ax0.plot([op, op], [0, op_activation], 'k', linewidth=1.5, alpha=0.9)
  ax0.set_title('Output membership activity')
  ax0.legend()

#    # Turn off top/right axes
#    for ax in (ax0,):
#        ax.spines['top'].set_visible(False)
#        ax.spines['right'].set_visible(False)
#        ax.get_xaxis().tick_bottom()
#        ax.get_yaxis().tick_left()
#
#    plt.tight_layout()

  print("\nFiring Strength of Negative (wneg): "+str(round(n2,4)))
  print("Firing Strength of Neutral (wneu): "+str(round(neu2,4)))
  print("Firing Strength of Positive (wpos): "+str(round(p2,4)))

  print("\nResultant consequents MFs:" )
  print("op_activation_low: "+str(op_activation_lo))
  print("op_activation_med: "+str(op_activation_md))
  print("op_activation_high: "+str(op_activation_hi))

  print("\nAggregated Output: "+str(aggregated))

  print("\nDefuzzified Output: "+str(output))

# Scale : Neg Neu Pos
  if 0<(output)<3.33:    # R
      print("\nOutput after Defuzzification: Negative")
      sentiment.append("Negative")

  elif 3.34<(output)<6.66:
      print("\nOutput after Defuzzification: Neutral")
      sentiment.append("Neutral")

  elif 6.67<(output)<10:
      print("\nOutput after Defuzzification: Positive")
      sentiment.append("Positive")

  print("Doc sentiment: " +str(senti[j])+"\n")
```

```
Output hidden; open in https://colab.research.google.com to view.
```

1.5 Evaluation

[ ]:

```
count=0
for k in range(len(doc)):
    if(sentiment_doc[k]==sentiment[k]):
        count=count+1
print("Accuracy is: "+ str(round(count/len(doc)*100,2)))

from sklearn.metrics import f1_score, precision_score, recall_score
```

```
y_true = sentiment_doc
y_pred = sentiment

p1=precision_score(y_true, y_pred, average='macro')

print("Precision score (MACRO): " + str(round((p1*100),2)))

r1=recall_score(y_true, y_pred, average='macro')

print("Recall score (MACRO): " + str(round((r1*100),2)))

f1=f1_score(y_true, y_pred, average='macro')
f2=f1_score(y_true, y_pred, average='micro')

print("F1 score (MACRO): " + str(round((f1*100),2)))
print("F1 score (MICRO): "+ str(round((f2*100),2)))

end = time.time()
print("Execution Time: "+str(round((end - start),3))+" secs")
```

```
Accuracy is: 50.3
Precision score (MACRO): 10.78
Recall score (MACRO): 19.53
F1 score (MACRO): 13.89
F1 score (MICRO): 50.3
Execution Time: 832.217 secs
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels
with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
accuracy_score(y_true, y_pred)
```

[ ]:

```
0.5030364372469636
```

[1]:

```
%cd drive/My\ Drive/
```

```
/content/drive/My Drive
```

[2]:

```
!pwd
```

```
/content/drive/My Drive
```

[ ]:

```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
 ↪texlive-generic-recommended
```

[6]:

```
!jupyter nbconvert --to pdf Fuzzy_Systems_Twitter_V2.ipynb
```

```
[NbConvertApp] WARNING | pattern u'Fuzzy_Systems_Twitter_V2.ipynb' matched no
files
This application is used to convert notebook files (*.ipynb) to various other
formats.

WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
-------

Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

--execute
    Execute the notebook prior to export.
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--stdout
    Write notebook output to stdout instead of files.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
-y
    Answer yes to any questions instead of prompting.
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--debug
    set log level to logging.DEBUG (maximize logging output)
--no-prompt
    Exclude input and output prompts from converted document.
--generate-config
```

```
    generate default config file
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    Choices: [1, 2, 3, 4]
    The nbformat version to write. Use this to downgrade notebooks.
--output-dir=<Unicode> (FilesWriter.build_directory)
    Default: ''
    Directory to write output(s) to. Defaults to output to the directory of each
    notebook. To recover previous default behaviour (outputting to the current
    working directory) use . as the flag value.
--writer=<DottedObjectName> (NbConvertApp.writer_class)
    Default: 'FilesWriter'
    Writer class used to write the  results of the conversion
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL')
    Set the log level by value or name.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
    Default: u''
    The URL prefix for reveal.js (version 3.x). This defaults to the reveal CDN,
    but can be any url pointing to a copy  of reveal.js.
    For speaker notes to work, this must be a relative path to a local  copy of
    reveal.js: e.g., "reveal.js".
    If a relative path is given, it must be a subdirectory of the current
    directory (from which the server is run).
    See the usage documentation
    (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-
    slideshow) for more details.
--to=<Unicode> (NbConvertApp.export_format)
    Default: 'html'
    The export format to be used, either one of the built-in formats
    ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
    'python', 'rst', 'script', 'slides'] or a dotted object name that represents
    the import path for an `Exporter` class
--template=<Unicode> (TemplateExporter.template_file)
    Default: u''
    Name of the template file to use
--output=<Unicode> (NbConvertApp.output_base)
    Default: ''
    overwrite base name use for output files. can only be used when converting
    one notebook at a time.
--post=<DottedOrNone> (NbConvertApp.postprocessor_class)
    Default: u''
    PostProcessor class used to write the results of the conversion
--config=<Unicode> (JupyterApp.config_file)
    Default: u''
    Full path of a config file.
```

```
To see all available configurables, use `--help-all`

Examples
--------

    The simplest way to use nbconvert is

    > jupyter nbconvert mynotebook.ipynb

    which will convert mynotebook.ipynb to the default format (probably HTML).

    You can specify the export format with `--to`.
    Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

    > jupyter nbconvert --to latex mynotebook.ipynb

    Both HTML and LaTeX support multiple output templates. LaTeX includes
    'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
    can specify the flavor of the format used.

    > jupyter nbconvert --to html --template basic mynotebook.ipynb

    You can also pipe the output to stdout, rather than a file

    > jupyter nbconvert mynotebook.ipynb --stdout

    PDF is generated via latex

    > jupyter nbconvert mynotebook.ipynb --to pdf

    You can get (and serve) a Reveal.js-powered slideshow

    > jupyter nbconvert myslides.ipynb --to slides --post serve

    Multiple notebooks can be given at the command line in a couple of
    different ways:

    > jupyter nbconvert notebook*.ipynb
    > jupyter nbconvert notebook1.ipynb notebook2.ipynb

    or you can specify the notebooks list in a config file, containing::

        c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

    > jupyter nbconvert --config mycfg.py
```

[ ]:

```
```