Sentiment Analysis Movie Review

For this analysis we'll be using a dataset of 50,000 movie reviews taken from IMDb. The data was compiled by Andrew Maas and can be found here: IMDb Reviews. The data is split evenly with 25k reviews intended for training and 25k for testing your classifier. Moreover, each set has 12.5k positive and 12.5k negative reviews. IMDb lets users rate movies on a scale from 1 to 10. To label these reviews the curator of the data labeled anything with ≤ 4 stars as negative and anything with ≥ 7 stars as positive. Reviews with 5 or 6 stars were left out.

The dataset is available at http://ai.stanford.edu/~amaas/data/sentiment/.

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# load_files module loads text files with categories as subfolder names.
# The folder names are used as supervised signal label names.
# The individual file names are not important.
from sklearn.datasets import load_files
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
import os
import random
np.set_printoptions(precision=3, suppress=True)
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale, StandardScaler

import matplotlib.pyplot as plt
%matplotlib inline
#plt.rcParams["figure.dpi"] = 300
plt.rcParams["savefig.dpi"] = 300
plt.rcParams["savefig.bbox"] = "tight"
```

You may not always find the dataset available for you to download from Sklearn, Keras type of platforms. Most of the times, you will have to find the dataset yourself and read it to your work space from some file. Below, you will find how you can read the IMDB text files from a folder where files are arranged by the label names.

In [3]:
```
def load_imdb_sentiment_analysis_dataset(data_path, seed=123):
    """Loads the IMDb movie reviews sentiment analysis dataset.

    # Arguments
        data_path: string, path to the data directory.
```

```
        seed: int, seed for randomizer.

    # Returns
        A tuple of training and validation data.
        Number of training samples: 25000
        Number of test samples: 25000
        Number of categories: 2 (0 - negative, 1 - positive)

    # References
        Mass et al., http://www.aclweb.org/anthology/P11-1015

        Download and uncompress archive from:
        http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
    """
    imdb_data_path = os.path.join(data_path, 'aclImdb')

    # Load the training data
    train_texts = []
    train_labels = []
    for category in ['pos', 'neg']:
        train_path = os.path.join(imdb_data_path, 'train', category)
        for fname in sorted(os.listdir(train_path)):
            if fname.endswith('.txt'):
                with open(os.path.join(train_path, fname)) as f:
                    z=f.read()
                    train_texts.append(str(z))
                train_labels.append(0 if category == 'neg' else 1)

    # Load the validation data.
    test_texts = []
    test_labels = []
    for category in ['pos', 'neg']:
        test_path = os.path.join(imdb_data_path, 'test', category)
        for fname in sorted(os.listdir(test_path)):
            if fname.endswith('.txt'):
                with open(os.path.join(test_path, fname)) as f:
                    z=f.read()
                    test_texts.append(str(z))
                test_labels.append(0 if category == 'neg' else 1)

    # Shuffle the training data and labels.
    random.seed(seed)
    random.shuffle(train_texts)
    random.seed(seed)
    random.shuffle(train_labels)

    return (train_texts, np.array(train_labels),
            test_texts, np.array(test_labels))
```

Load the train dataset

- Load the dataset and divide data into trainval and test datasets.
- Print the type of the trainval dataset
- Print the length of the trainval dataset
- Print the first document in your trainval dataset

In [4]:
```
#reviews_train = load_files("aclImdb/train")
#text_train, y_train = reviews_train.data, reviews_train.target
text_trainval, y_trainval, text_test, y_test = \
load_imdb_sentiment_analysis_dataset("", seed=123)
print("type of text train: {}".format(type(text_trainval)))
print("length of text_train: {}".format(len(text_trainval)))
print("text_train[1]:\n{}".format(text_trainval[1]))
```

```
type of text train: <class 'list'>
length of text_train: 25000
text_train[1]:
The long list of "big" names in this flick (including the ubiquitous John Mill
s) didn't bowl me over to the extent that I couldn't judge the film on its act
ual merits. It is FULL of stereotypes, caricatures, and standard, set scenes,
from the humble air-ace hero to the loud-mouthed yank flyer. The music track w
as such that at one point, about an hour before the end, I thought the film wa
s over: loud, rising crescendo, grand flourish and finish then silence, but th
en the movie continued! I found no real storyline, haphazard writing, but smar
tly-pressed uniforms and the pretty Jean Simmons (pre-nose job) with a rousing
little ditty. I cannot say that this picture has any of the ingredients which
make a film great. I found it maudlin, mawkish and minor.
```

- Print the type of your labels 'y'
- Print the lnegth of your labels
- Print the first three labels

In [5]:
```
print("type of y train: {}\n".format(type(y_trainval)))
print("length of y train: {}\n".format(len(y_trainval)))
print("y_train[0]:{}".format(y_trainval[0]))
print("y_train[1]:{}".format(y_trainval[1]))
print("y_train[2]:{}".format(y_trainval[2]))
```

```
type of y train: <class 'numpy.ndarray'>

length of y train: 25000

y_train[0]:0
y_train[1]:0
y_train[2]:1
```

In [6]:
```
a=(text_trainval[1])
print(a)
a[0]
a.replace("<br />", " ")
```

```
The long list of "big" names in this flick (including the ubiquitous John Mill
s) didn't bowl me over to the extent that I couldn't judge the film on its act
ual merits. It is FULL of stereotypes, caricatures, and standard, set scenes,
from the humble air-ace hero to the loud-mouthed yank flyer. The music track w
as such that at one point, about an hour before the end, I thought the film wa
s over: loud, rising crescendo, grand flourish and finish then silence, but th
en the movie continued! I found no real storyline, haphazard writing, but smar
tly-pressed uniforms and the pretty Jean Simmons (pre-nose job) with a rousing
little ditty. I cannot say that this picture has any of the ingredients which
make a film great. I found it maudlin, mawkish and minor.
```

Out[6]:
```
'The long list of "big" names in this flick (including the ubiquitous John Mil
ls) didn\'t bowl me over to the extent that I couldn\'t judge the film on its
actual merits. It is FULL of stereotypes, caricatures, and standard, set scene
s, from the humble air-ace hero to the loud-mouthed yank flyer. The music trac
k was such that at one point, about an hour before the end, I thought the film
was over: loud, rising crescendo, grand flourish and finish then silence, but
then the movie continued! I found no real storyline, haphazard writing, but sm
artly-pressed uniforms and the pretty Jean Simmons (pre-nose job) with a rousi
ng little ditty. I cannot say that this picture has any of the ingredients whi
ch make a film great. I found it maudlin, mawkish and minor.'
```

- If you go over the reviews in the directory structure, you will notice that many of the reviews contain some html line breaks.
- Let's clean this. text_trainval is a list containing type bytes(bytes represents a binary encoding of the string data in Python 3.
- after cleaning your dataset, divide your trainval dataset into train and validation datasets

In [7]:
```
text_trainval = [doc.replace("<br />", " ") \
                 for doc in text_trainval]
print("\nSamples per class (training): {}\n".\
      format(np.bincount(y_trainval)))
```

```
Samples per class (training): [12500 12500]
```

In [8]:
```
len(np.unique(text_trainval))
```

Out[8]:
```
24904
```

In [9]:
```
np.unique(y_trainval)
```

Out[9]:
```
array([0, 1])
```

In [10]:
```
len(np.unique(y_trainval))
```

Out[10]: 2

In [11]:
```
print("text_train[1]:\n{}".format(text_trainval[1]))
```

```
text_train[1]:
The long list of "big" names in this flick (including the ubiquitous John Mill
s) didn't bowl me over to the extent that I couldn't judge the film on its act
ual merits. It is FULL of stereotypes, caricatures, and standard, set scenes,
from the humble air-ace hero to the loud-mouthed yank flyer. The music track w
as such that at one point, about an hour before the end, I thought the film wa
s over: loud, rising crescendo, grand flourish and finish then silence, but th
en the movie continued! I found no real storyline, haphazard writing, but smar
tly-pressed uniforms and the pretty Jean Simmons (pre-nose job) with a rousing
little ditty. I cannot say that this picture has any of the ingredients which
make a film great. I found it maudlin, mawkish and minor.
```

In [12]:
```
print("y_train[1]:\n{}".format(y_trainval[1]))
```

```
y_train[1]:
0
```

In [13]:
```
text_train, text_val, y_train, y_val =\
train_test_split(text_trainval,y_trainval,\
                 stratify=y_trainval, random_state=0)
```

Latent Semantic Analysis (LSA)

LSA is basically just the same as PCA. The idea is to reduce the dimensionality of the data to some semantically meaningful components. The thing is, we can't easily do PCA here, because we can’t remove the mean. Remember, this is a sparse dataset and so the zero is sort of meaningful and if we shift the zero, like by trying to subtract the mean, we will get a density dataset that won't have many zeros anymore and so we won’t be able to even store it.

There are ways to still compute PCA without explicitly creating a big dense matrix, but in the traditional LSA, we just ignore that and do a singular value decomposition (SVD) of the data. This is exactly the same computation as PCA, only, we don't subtract the mean.

In [14]:
```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer(stop_words="english", min_df=4)
X_train = vect.fit_transform(text_train)
```

In [15]:
```
X_train.shape
```

Out[15]: (18750, 26604)

Below, we are doing a truncated SVD and extract 100 components. And so the components we extract are 100 X number of features. So we have 100 vectors, where each feature corresponds to one of the words in the vocabulary.

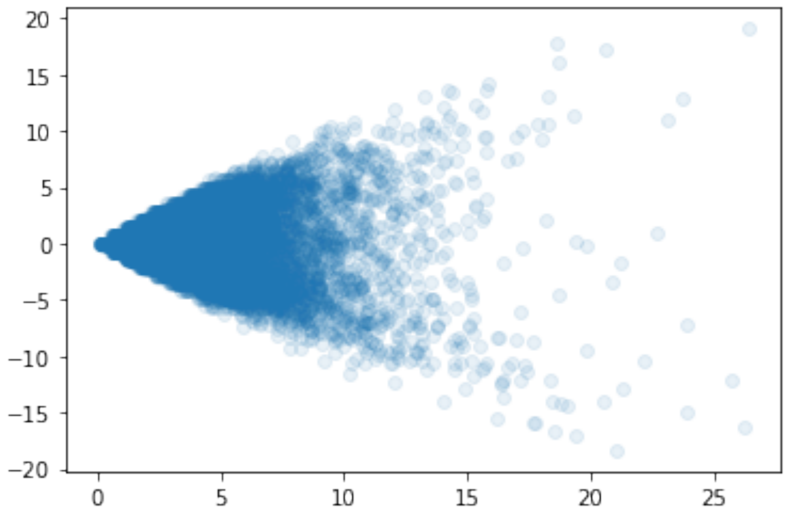
In [16]:
```
from sklearn.decomposition import TruncatedSVD
lsa = TruncatedSVD(n_components=100)
X_lsa = lsa.fit_transform(X_train)
```

In [17]:
```
lsa.components_.shape
```

Out[17]:
```
(100, 26604)
```

In [18]:
```
plt.scatter(X_lsa[:, 0], X_lsa[:, 1], alpha=.1)
```

Out[18]:
```
<matplotlib.collections.PathCollection at 0x7fa3a23e4b20>
```

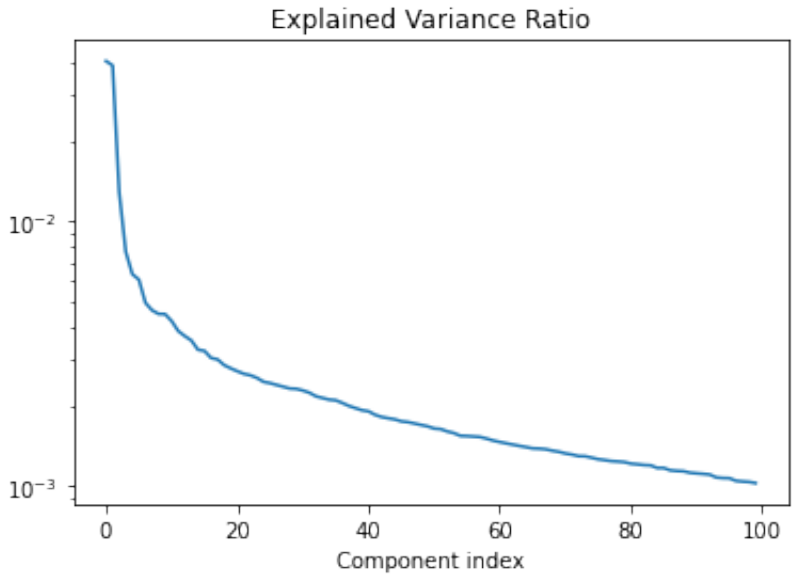


Here's is a semi-log scale. And you can see that the first one or two explain a lot and then rapidly decreases. A lot is captured in the first 10, and then it goes down.

But still at 100, there's still a lot of variances left, so it'll just probably go down logarithmically as it goes on.

In [19]:
```
plt.semilogy(lsa.explained_variance_ratio_)
plt.title("Explained Variance Ratio")
plt.xlabel("Component index")
#plt.savefig("images/lsa_truncated_svd_plot.png")
```

Out[19]: Text(0.5, 0, 'Component index')



In [21]:
```
lsa.components_.shape
```

Out[21]: (100, 26604)

In [33]:
```
coef2=lsa.components_[:5,:8]
print(coef2)
np.argsort(np.abs(coef2))[-3:]
```

```
[[ 0.001  0.003  0.     0.     0.     0.     0.     0.   ]
 [ 0.    -0.001  0.    -0.    -0.     0.    -0.    -0.   ]
 [ 0.001  0.004  0.     0.     0.001 -0.     0.    -0.   ]
 [ 0.001  0.002  0.    -0.     0.001  0.     0.     0.   ]
 [-0.     0.    -0.     0.    -0.     0.    -0.     0.   ]]
```

Out[33]:
```
array([[7, 5, 3, 6, 2, 4, 0, 1],
       [2, 3, 6, 5, 7, 4, 0, 1],
       [3, 7, 0, 6, 5, 4, 1, 2]])
```

In [45]:
```
inds = np.argsort(np.abs(coef2))[2]
important_coefs = coef2[2,inds]
sort_sign = np.argsort(important_coefs)
myrange = range(len(inds))
combined_inds = inds[sort_sign]
print(np.abs(coef2)[2])
print(inds)
print(important_coefs)
print(sort_sign)
print(combined_inds)
print(coef2[2,combined_inds])
```

```
[0.001 0.004 0.    0.    0.001 0.    0.    0.   ]
[7 5 3 6 2 4 0 1]
[-0.    -0.     0.     0.     0.     0.001  0.001  0.004]
[1 0 2 3 4 5 6 7]
[5 7 3 6 2 4 0 1]
[-0.    -0.     0.     0.     0.     0.001  0.001  0.004]
```

In [23]:

```python
coef=lsa.components_
np.argsort(np.abs(coef))[-5:]
```

Out[23]:

```
array([[10712,  8966,   322, ..., 11374, 21092, 25751],
       [15713,  5314, 26552, ..., 14076, 25751, 24612],
       [14286, 16432, 14931, ..., 25751, 24612, 14448],
       [23090, 20019,  3221, ...,  9991, 23871, 17848],
       [24997,  3029, 18003, ..., 23997,  6045,  1034]])
```

In [21]:

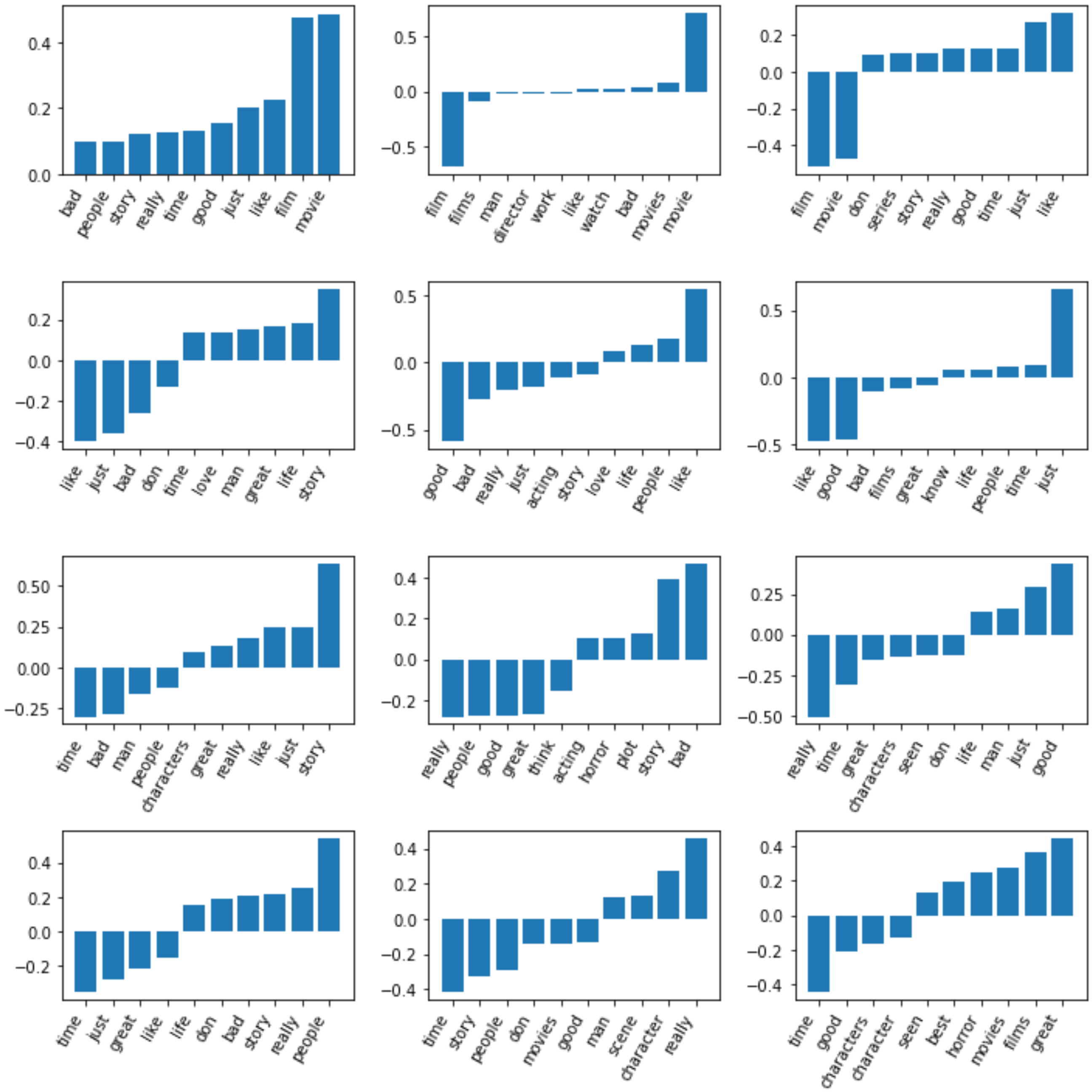
```python
def plot_important_features(coef, feature_names, top_n=20, ax=None):
    if ax is None:
        ax = plt.gca()
    inds = np.argsort(np.abs(coef))[-top_n:]
    important_coefs = coef[inds]
    sort_sign = np.argsort(important_coefs)
    myrange = range(len(inds))
    combined_inds = inds[sort_sign]

    ax.bar(myrange, coef[combined_inds])
    ax.set_xticks(myrange)
    ax.set_xticklabels(feature_names[combined_inds], rotation=60, ha="right")
```

Let's look at the eigenvectors with the highest singular values. So the first one is sort of what you would usually get. They're all positive because all entries in the data matrix are positive since its bag of words. This is sort of, in a sense, just translating into the middle of the data, somewhat similar to trying to model the mean. ‘Movie’ and ‘film’ are obviously the most common words.

The second eigenvector is whether someone uses the word ‘movies’ or ‘films’. You can see that either someone uses ‘film’ and ‘films’ or ‘movie’ and ‘movies’. Basically, people don't usually use both of them in the same comment.

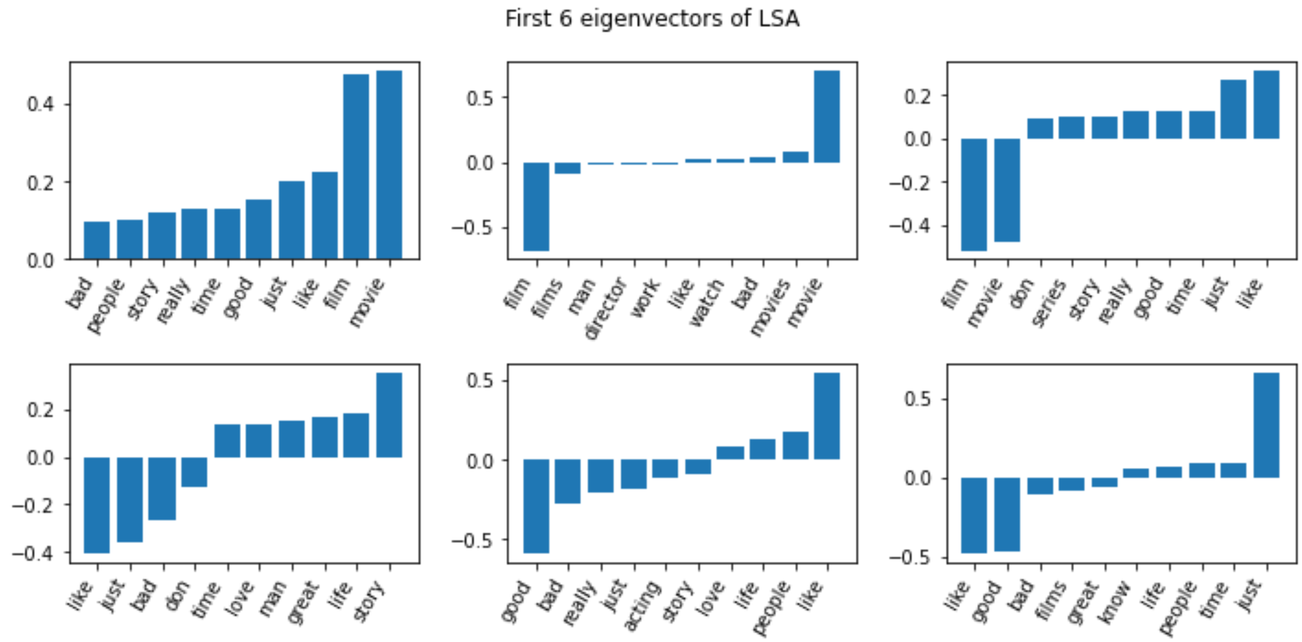
In [22]:

```python
feature_names = np.array(vect.get_feature_names())
n_components = 10
fig, axes = plt.subplots(n_components // 3 + 1, 3, figsize=(10, 10))
for ax, coef in zip(axes.ravel(), lsa.components_):
    plot_important_features(coef, feature_names, top_n=10, ax=ax)

plt.tight_layout()
```



In [23]:

```
# Only six first
n_components = 6
fig, axes = plt.subplots(n_components // 3, 3, figsize=(10, 5))
for ax, coef in zip(axes.ravel(), lsa.components_):
    plot_important_features(coef, feature_names, top_n=10, ax=ax)
plt.suptitle("First 6 eigenvectors of LSA")
plt.tight_layout()
plt.subplots_adjust(top=.9)
#plt.savefig("images/lsa_six_eigvec.png")
```



In [ ]:

Trying normalizer

MaxAbsScaler performs the same way as a standard scaler, only it doesn't subtract the mean. So it just scales everything to have the same maximum absolute value. And so in this sense, I'm basically scaling down film and movie to have the same importance as the other words. `Movie` and `Film` were dominating first couple of components. Try to get rid of that effect.

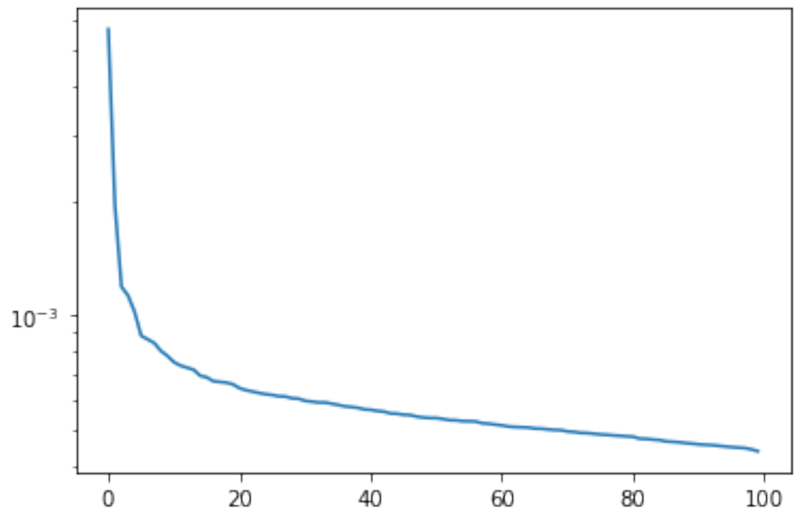
In [27]:

```
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
X_scaled = scaler.fit_transform(X_train)

lsa_scaled = TruncatedSVD(n_components=100)
X_lsa_scaled = lsa_scaled.fit_transform(X_scaled)
```

In [28]:

```
plt.semilogy(lsa_scaled.explained_variance_ratio_)
```

Out[28]:
```
[<matplotlib.lines.Line2D at 0x7fb50109dac0>]
```

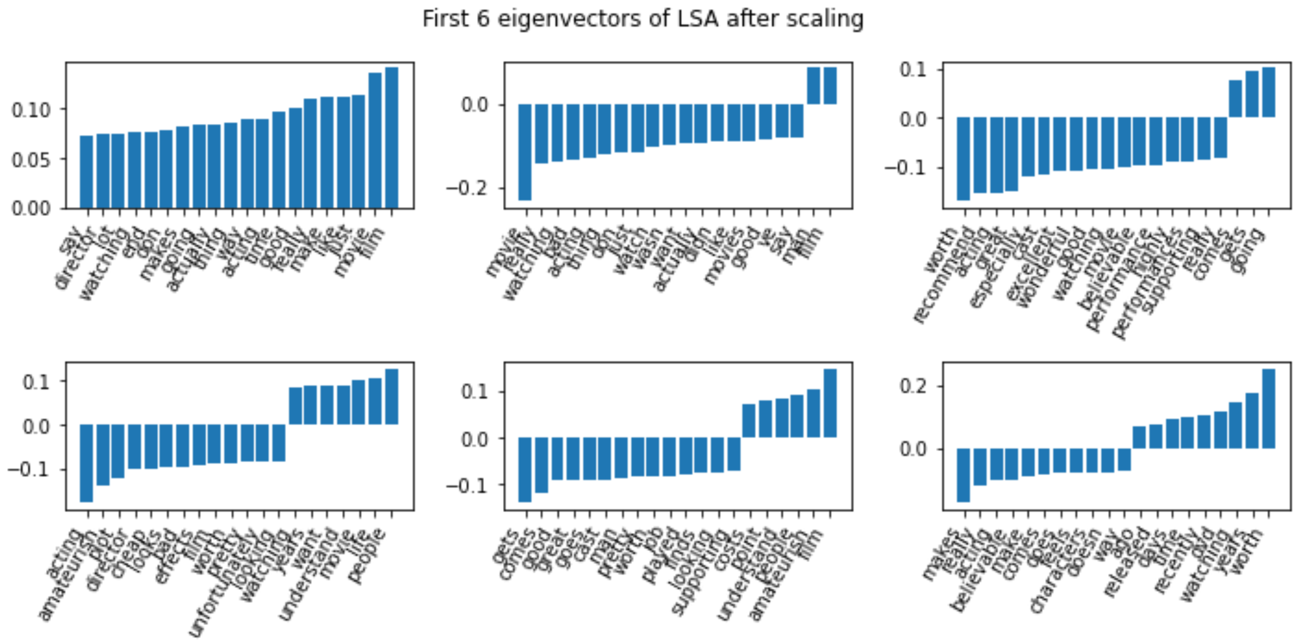


This is the first six eigenvectors of LSA after scaling. The first one still points towards the direction of the mean. So I found some components like the third one here, which are interesting because I interpreted this as like a writer sophistication. So there are some people that use a lot of very short words and some people use a lot of very long words. So people that use words like ‘cinematography’ also use words like ‘performance’ and ‘excellent’. This also comes out from some of the other methods.

And so I looked at a couple of these components and it turned out that the component 1 and the component 3 are actually related to the good and bad review.

Movie and film still important, but not that dominant any more.

In [30]:
```
n_components = 6
fig, axes = plt.subplots(n_components // 3, 3, figsize=(10, 5))
for ax, coef in zip(axes.ravel(), lsa_scaled.components_):
    plot_important_features(coef, feature_names, top_n=20, ax=ax)

plt.suptitle("First 6 eigenvectors of LSA after scaling")
plt.tight_layout()
plt.subplots_adjust(top=.9)
#plt.savefig("images/lsa_six_eigvec_scaled.png")
```



Now let us use logistic regression on both regular dataset and scaled LSA transformed dataset.

In [32]:
```
#reviews_test = load_files("../data/aclImdb/test/")
#text_test, y_test = reviews_test.data, reviews_test.target
#text_test = [doc.replace(b"<br />", b" ") for doc in text_test]
X_test = vect.transform(text_test)
```

In [33]:
```
X_test_lsa_scaled = lsa_scaled.transform(scaler.transform(X_test))
```

In [34]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=.1).fit(X_train, y_train)
lr.score(X_test, y_test)
```

Out[34]: 0.8718

In [35]:
```
lr_lsa = LogisticRegression(C=100).fit(X_lsa_scaled[:, :10], y_train)
lr_lsa.score(X_test_lsa_scaled[:, :10], y_test)
```

Out[35]: 0.83092

In [36]:
```
lr_lsa.score(X_lsa_scaled[:, :10], y_train)
```

Out[36]: 0.8350933333333334

In [37]:
```
lsa_scaled1k = TruncatedSVD(n_components=1000)
X_lsa_scaled1k = lsa_scaled1k.fit_transform(X_scaled)
```

In [38]:
```
X_test_lsa_scaled1k = lsa_scaled1k.transform(scaler.transform(X_test))
```

In [39]:
```
lr_lsa = LogisticRegression().fit(X_lsa_scaled1k, y_train)
lr_lsa.score(X_test_lsa_scaled1k, y_test)
```

Out[39]: 0.87316

In [40]:
```
lr_lsa.score(X_lsa_scaled1k, y_train)
```

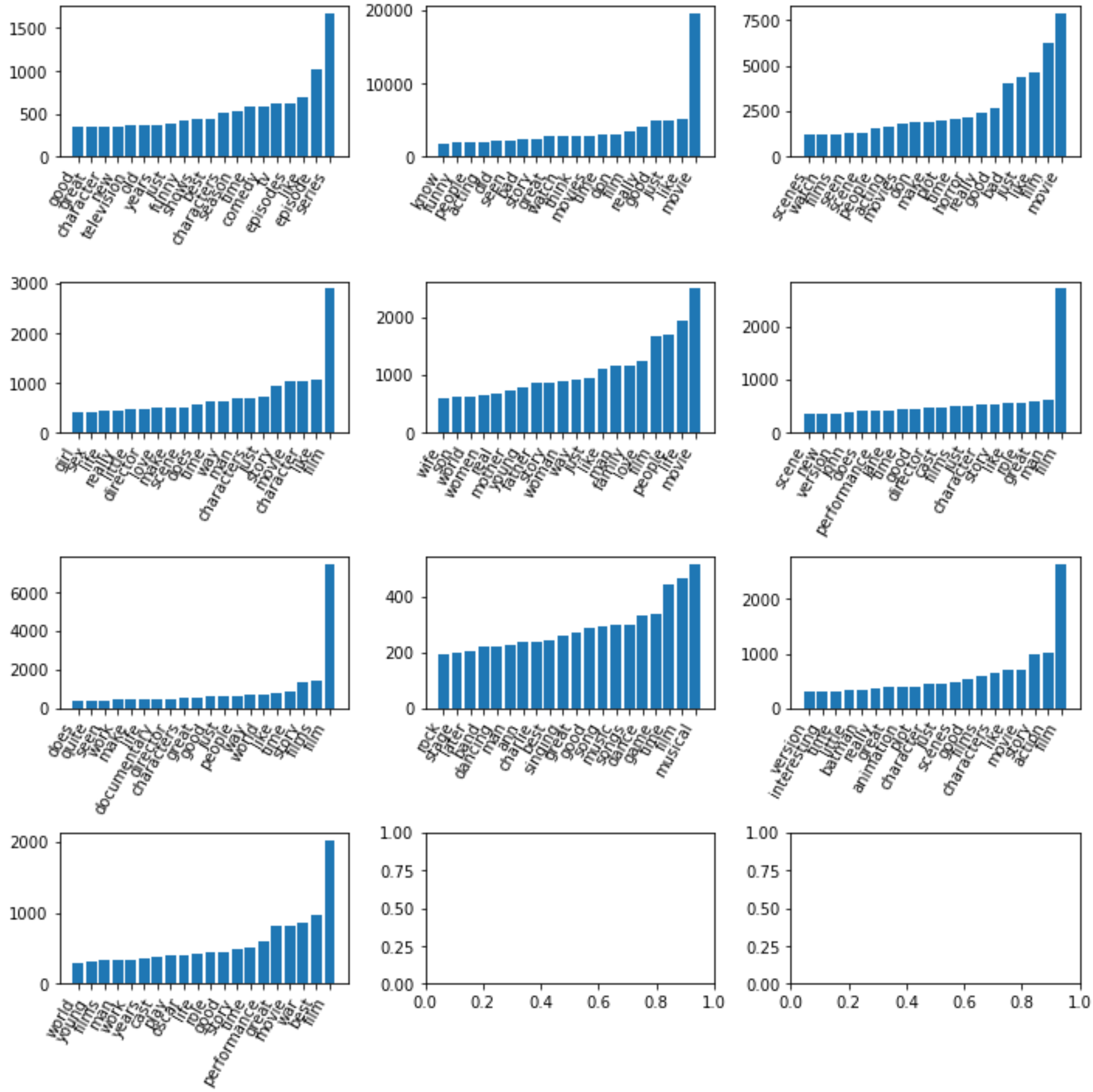
Out[40]: 0.8969066666666666

Topic Modeling

- Each document is created as a mixture of topics
- Topics are distributions over words
- Learn topics and composition of documents simultaneously
- Unsupervised

In [41]:
```
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=10, learning_method="batch")
X_lda = lda.fit_transform(X_train)
```

In [42]:
```
feature_names = np.array(vect.get_feature_names())
n_components = 10
fig, axes = plt.subplots(n_components // 3 + 1, 3, figsize=(10, 10))
for ax, coef in zip(axes.ravel(), lda.components_):
    plot_important_features(coef, feature_names, top_n=20, ax=ax)

plt.tight_layout()
```

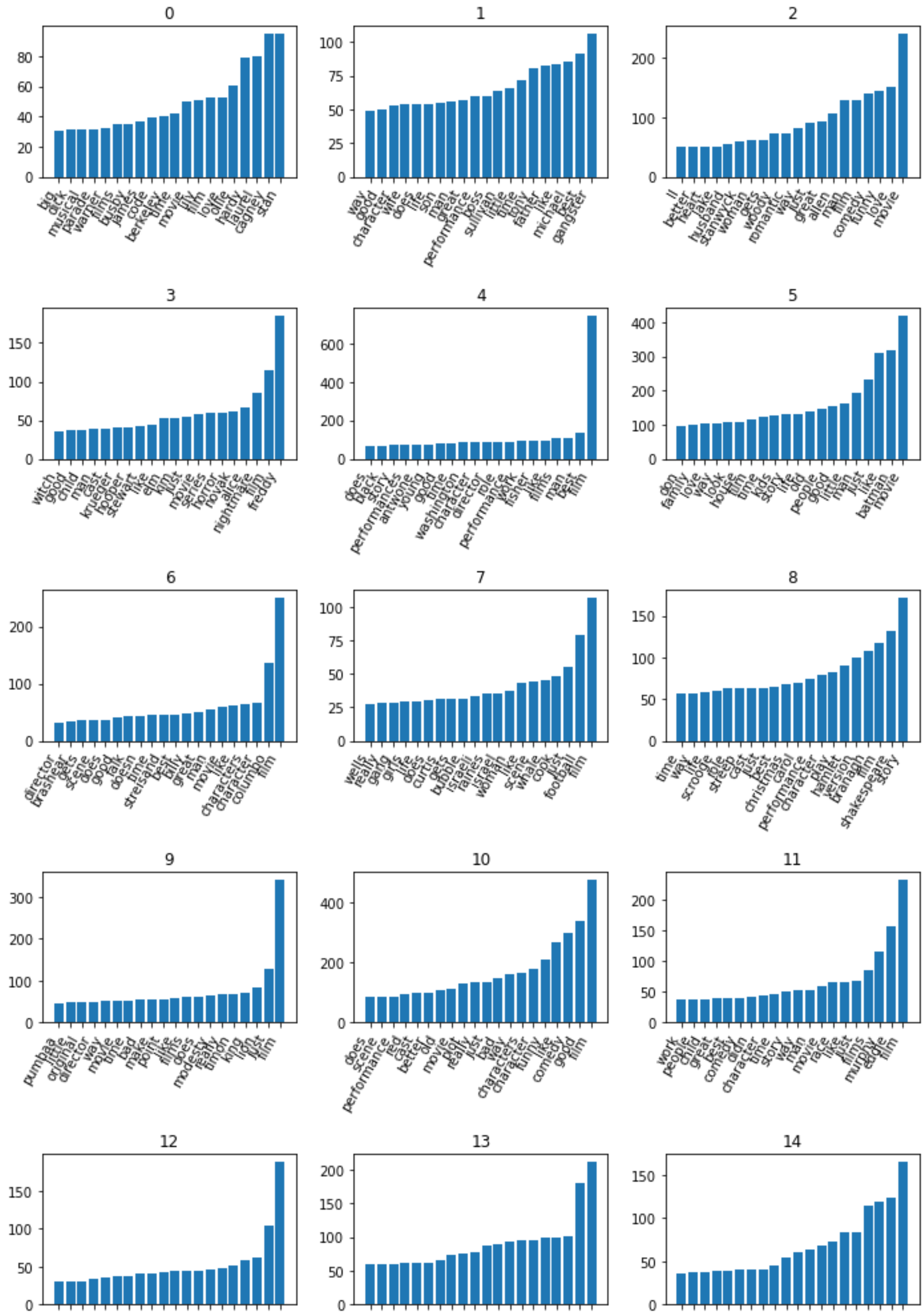


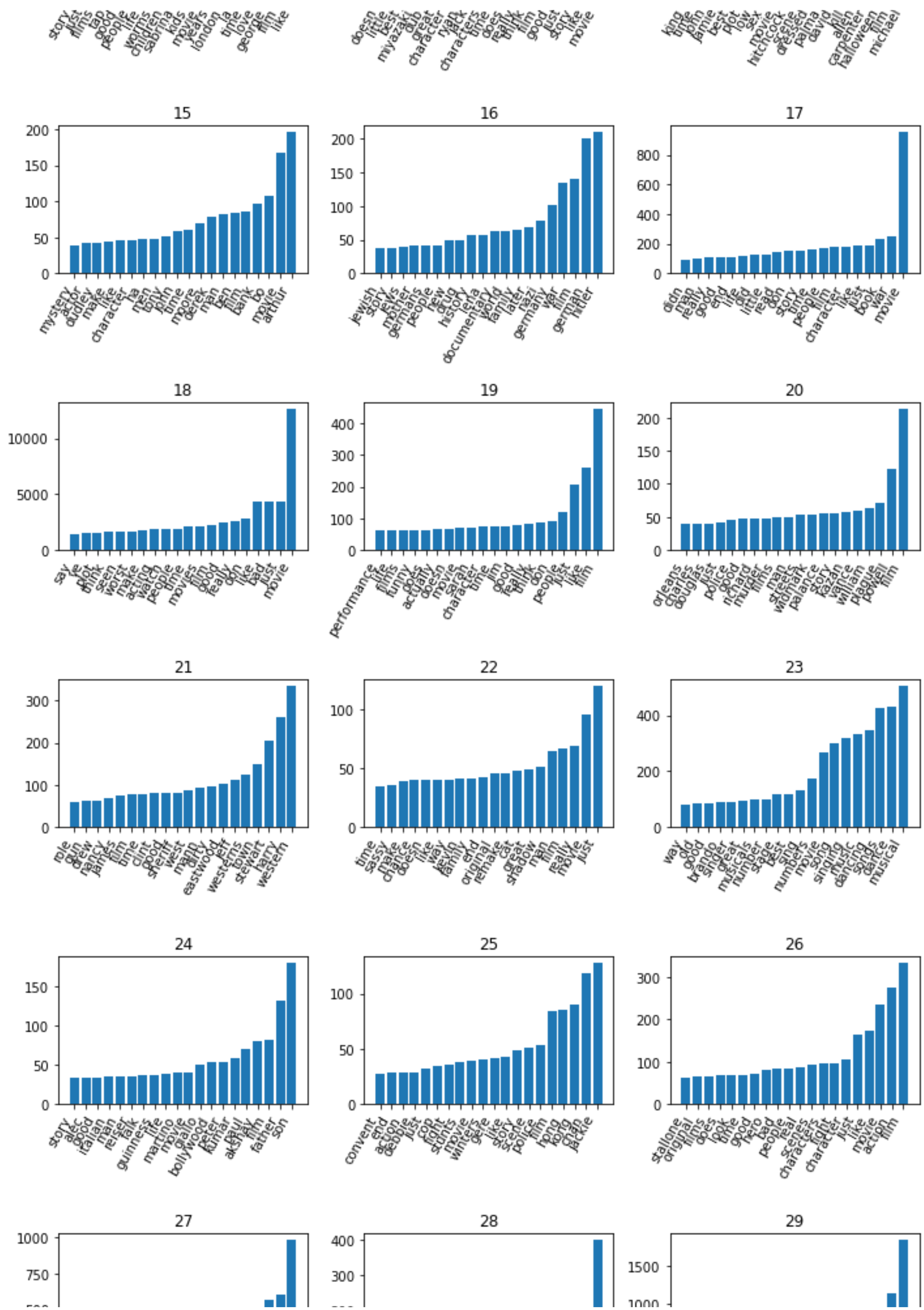
In [43]:

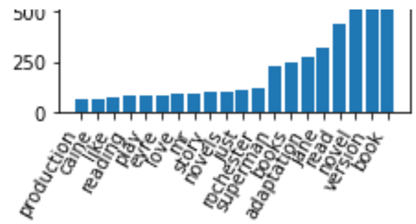
```
lda100 = LatentDirichletAllocation(n_components=100, learning_method="batch")
X_lda100 = lda100.fit_transform(X_train)
```

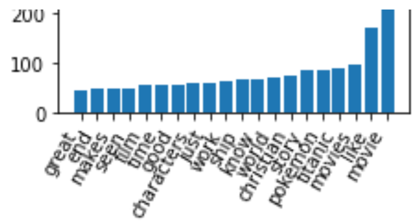
In [44]:

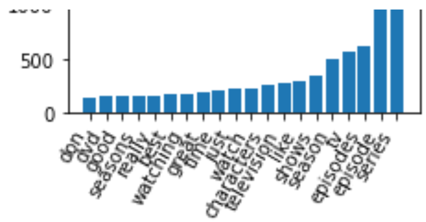
```
large_comp_inds = np.argsort(X_lda100.mean(axis=0))[-50:][::-1]

feature_names = np.array(vect.get_feature_names())
n_components = 50
fig, axes = plt.subplots(n_components // 3 + 1, 3, figsize=(10, 50))
for i, (ax, coef) in enumerate(zip(axes.ravel(), lda100.components_[:, :])):
    plot_important_features(coef, feature_names, top_n=20, ax=ax)
    ax.set_title(i)

plt.tight_layout()
```

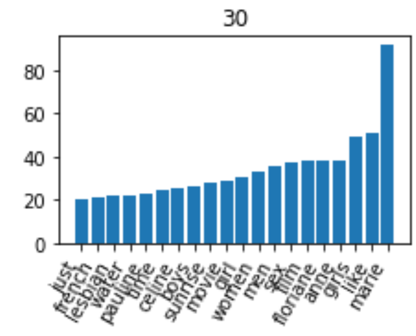

0
big dick musical parade warner films busby james code berkeley time movie lily film love ollie hardy laurel cagney stan
1
way good character wife does life son man great performance boss sullivan little time tony father like michael best gangster
2
ll better heart jake husband stanwyck woman gets woody romantic way just great allen man film comedy funny love movie
3
witch good child man cast krueger hooper stewart like elm film just movie series horror novak alice nightmare film freddy
4
does black story performances antwone young good time washington character director role performance work fisher like films man best film
5
don family love way look house film time kids story life old people good little man just like batman movie
6
director brashear gets scene does good talk doesn time streisand best billy great man movie like characters character columbo film
7
wells really gang girls life does curtis gets bubble israeli raines israel woman like scene whale cook just football film
8
time way life scrooge love streep cast just best christmas carol performance character play hamlet version branagh film shakespeare story
9
pumbaa little original director way movie time bad make point like films does modesty really timon king lion just film
10
does scene performance red cast better old movie plot really just bad way characters character funny like comedy good film
11
work people child great best comedy didn character time story way man movie race like just films murphy eddie film
12
13
14

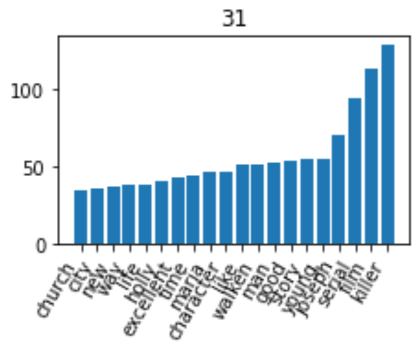

15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

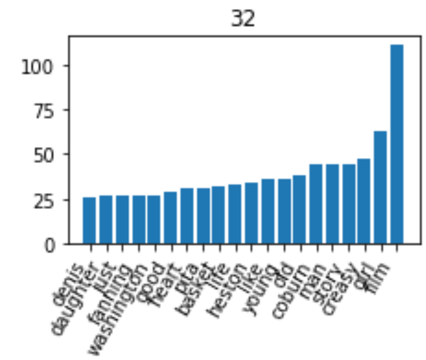

0
250
500
production
caine
like
reading
play
eyre
love
mr
story
novels
just
rochester
superman
books
adaptation
jane
read
novel
version
book

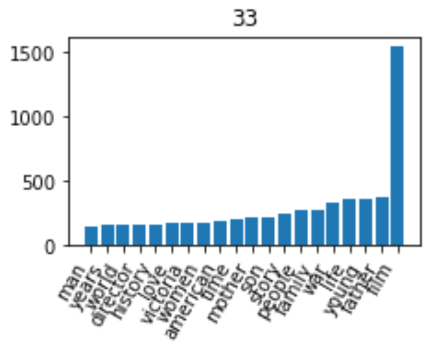

0
100
200
great
end
makes
seen
film
time
good
characters
just
work
ship
know
world
christian
story
pokemon
titanic
movies
like
movie

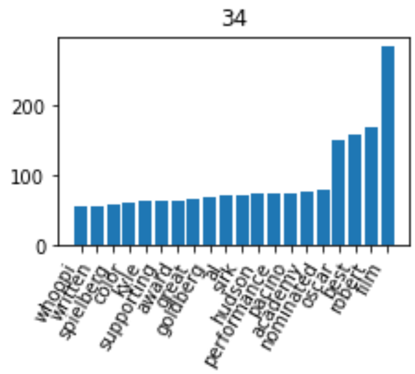

0
500
don
dvd
good
seasons
really
best
watching
great
time
just
watch
characters
television
like
shows
season
tv
episodes
episode
series

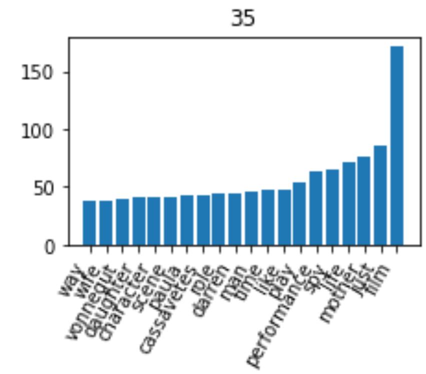

30
0
20
40
60
80
just
french
lesbian
walter
pauline
time
celine
boys
sunrise
movie
girl
women
men
sex
film
floriane
anne
girls
like
marie

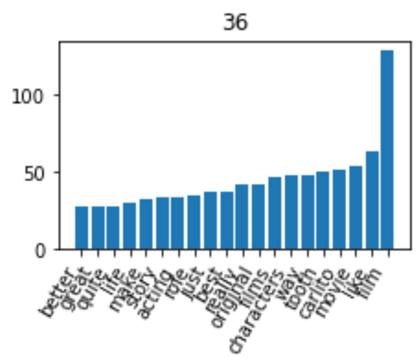

31
0
50
100
church
city
new
way
life
holly
excellent
time
maria
character
like
walken
man
good
story
young
joseph
serial
film
killer

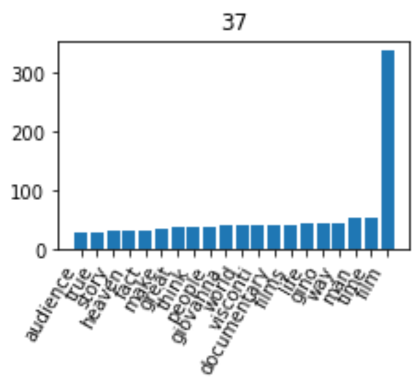

32
0
25
50
75
100
denis
daughter
just
fanning
washington
good
heart
rita
basket
life
heston
like
young
old
coburn
man
story
creasy
girl
film

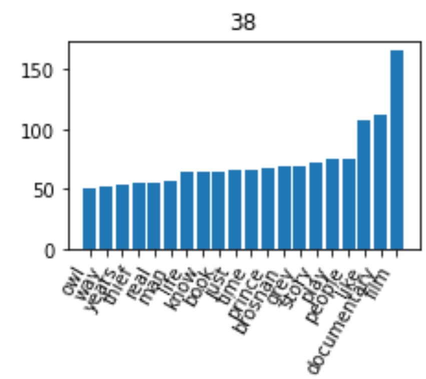

33
0
500
1000
1500
man
years
world
director
history
love
victoria
women
american
time
mother
son
story
people
family
war
life
young
father
film

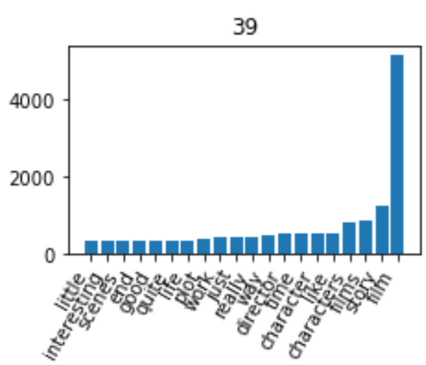

34
0
100
200
whoopi
written
spielberg
color
kyle
supporting
award
great
goldberg
al
sirk
hudson
performance
pacino
academy
nominated
oscar
best
robert
film

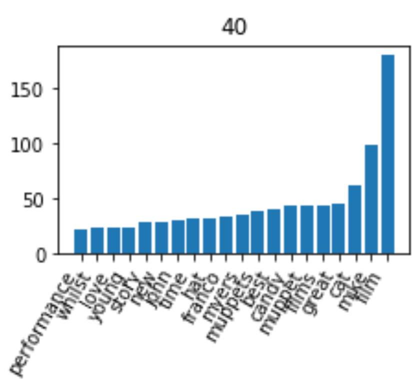

35
0
50
100
150
way
wife
vonnegut
daughter
character
scene
paula
cassavetes
role
darren
man
time
like
play
performance
son
life
mother
just
film

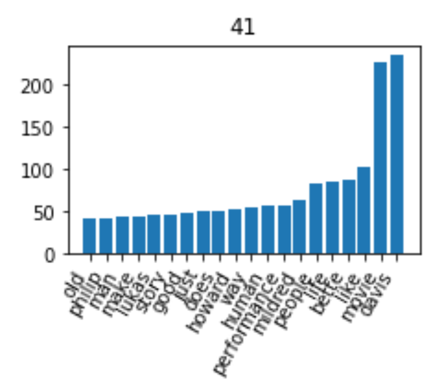

36
0
50
100
better
great
quite
life
make
story
acting
role
just
best
really
original
films
characters
way
tooth
carlito
movie
like
film

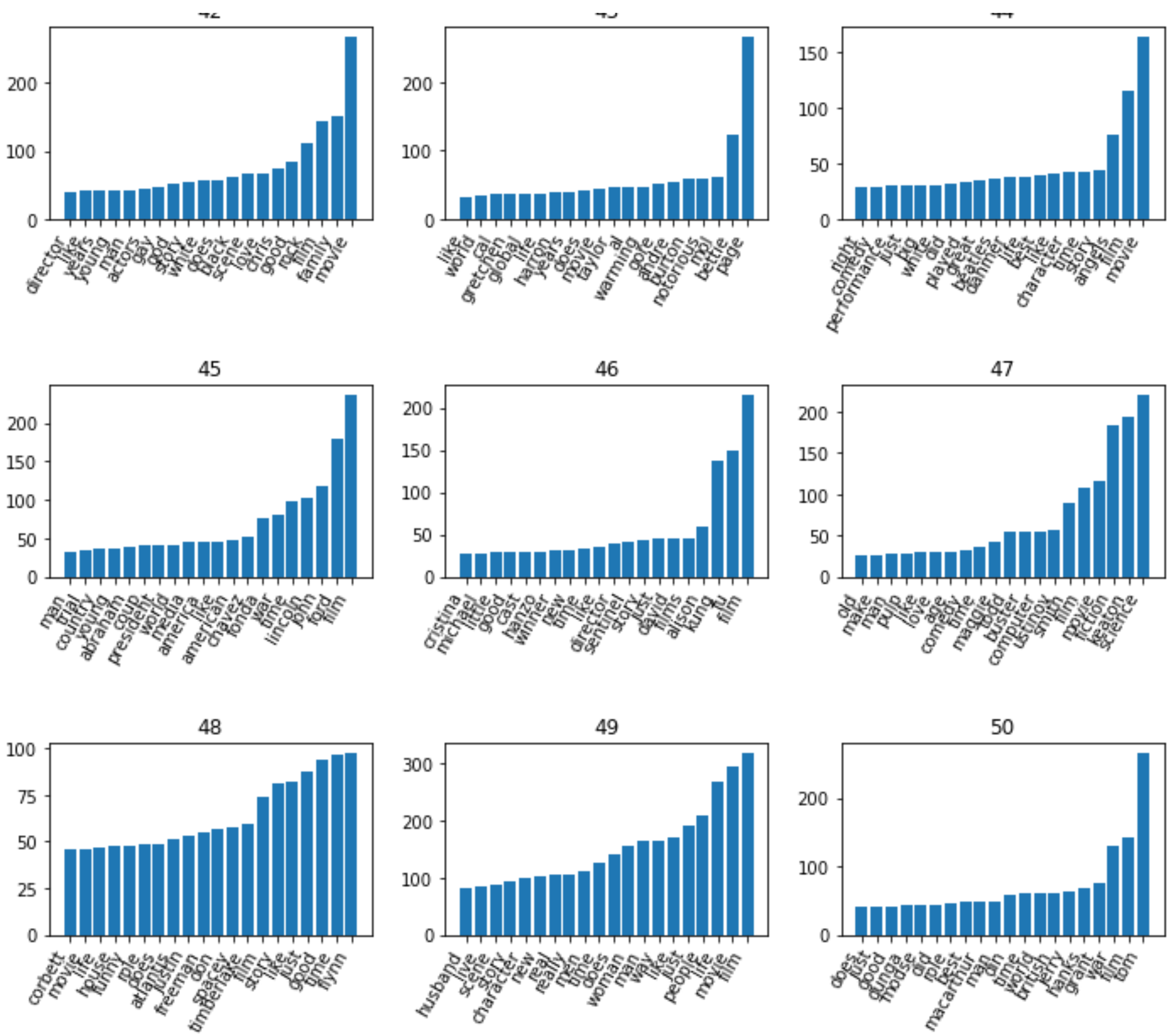

37
0
100
200
300
audience
true
story
heaven
fact
make
great
think
people
giovanna
world
visconti
documentary
films
life
gino
way
man
time
film


38
0
50
100
150
owl
way
years
thief
real
man
life
know
book
just
time
prince
brosnan
grey
story
play
people
like
documentary
film


39
0
2000
4000
little
interesting
scenes
end
good
quite
life
plot
work
just
really
way
director
time
character
like
characters
films
story
film


40
0
50
100
150
performance
whilst
love
young
story
new
john
time
hat
franco
myers
muppets
best
candy
muppet
films
great
cat
like
film


41
0
50
100
150
200
old
philip
man
make
lukas
story
good
just
does
howard
way
human
performance
mildred
people
life
bette
like
movie
davis


director like years young man actors gay god story white does black scene love chris good rock film family movie
like world cal gretchen global life harron years does movie taylor al warming gore andre burton notorious mpi bettie page
right comedy performance just big white did played great beatles dahmer life best like character time story angels film movie
45
man trial country young abraham coup president world media america like american chavez fonda war time lincoln john ford film
46
cristina michael little good cast hanzo winner new time like director sentinel story just david films alison kung fu film
47
old make man pulp like love age comedy time maggie todd buster computer ustinov smith film movie fiction keaton science
48
corbett movie life house funny role does atlantis justin freeman don spacey timberlake film story like just good time flynn
49
husband live scene story character new real really men time does woman man way like just people life movie film
50
does just good gunga moose did role best macarthur man din time world british jerry hanks grant war film tom

In [45]:
```
def print_topics(topics, feature_names, components, topics_per_chunk=6,
                 n_words=20):
    sorting = np.argsort(components, axis=1)[:, ::-1]
    for i in range(0, len(topics), topics_per_chunk):
        # for each chunk:
        these_topics = topics[i: i + topics_per_chunk]
        # maybe we have less than topics_per_chunk left
        len_this_chunk = len(these_topics)
        # print topic headers
        print(("topic {:<8}" * len_this_chunk).format(*these_topics))
        print(("-------- {0:<5}" * len_this_chunk).format(""))
        # print top n_words frequent words
        for i in range(n_words):
            try:
                print(("{:<14}" * len_this_chunk).format(
                    *feature_names[sorting[these_topics, i]]))
            except:
                pass
        print("\n")
```

In [48]:
```
large_comp_inds = np.argsort(X_lda100.mean(axis=0))[-12:][::-1]

feature_names = np.array(vect.get_feature_names())
sorting = np.argsort(lda100.components_, axis=1)[:, ::-1]
print_topics(large_comp_inds, feature_names, sorting)
```

```
topic 18      topic 62      topic 69      topic 39      topic 57      topic 33
--------      --------      --------      --------      --------      --------
master        luck          encyclopedia  cooperation   crippling     milder
suggests      suspected     shy           veronika      target        sweeter
purity        proceedings   recognisable  emerge        caged         rapidly
puritanical   proceeding    reclusive     hah           prosecutor    rapid
puritan       proceeded     recluse       greenlight    prosecution   rapes
purists       proceed       reclaim       governments   prosecuting   raped
purist        procedures    reckoning     grams         prosecuted    rape
puri          procedure     reckoned      hams          prose         rapaport
purgatory     procedural    reckon        guardians     prosaic       rap
purest        problems      recklessness  gunfights     pros          raoul
purely        problematic   blossoms      censorship    props         rao
pure          probing       reciting      fulfilling    propriety     rants
purchasing    priyanka      receptacle    fun           proposition   rantings
purchases     probe         recites       globalization prophet       ranting
purchased     probation     recite        goer          proposing     rant
berkley       probably      bilal         georges       proposes      ransom
purchase      probable      console       gino          proposed      ranma
pups          probability   butter        gimmick       propose       ranks
puppy         pro           recital       humiliation   proposal      rankings
audience      prizes        135           html          proportions   ranking


topic 61      topic 29      topic 10      topic 17      topic 5       topic 49
--------      --------      --------      --------      --------      --------
melodies      cave          filter        finance       mein          minute
swimsuit      suspended     styles        succeed       sutton        shuts
radha         procedural    pyle          pygmies       racing        ravenous
radar         problems      pygmies       pyewacket     racially      raven
danielle      problematic   adelaide      pyar          racial        raved
rad           problem       pyewacket     pvt           rachel        contract
ual
racy          probing       pyar          puzzling      rachael       rave
racks         probe         carroll       puzzles       racetrack     ravages
racking       probation     pvt           puzzled       blockbusters  ravaged
racket        probably      puzzling      puzzle        races         ravage
racked        probable      puzzles       putting       racer         raunchy
rack          pro           puzzled       assumption    race          raucous
brutish       privileged    comparison    puts          raccoons      rational
ization
racist        prizes        puzzle        putrid        raccoon       rattling
racism        prized        putting       putain        rabies        rattlesn
ake
cutouts       19            bon           pussy         cottage       rattle
racing        prize         puts          puss          rabid         ratso
racially      priyanka      putrid        pushy         rabble        rats
racial        priyadarshan  putain        pushing       rabbits       rationin
g
rachel        priya         pussy         pushes        rabbit        rational
ly
```

In [ ]: