Project

Twitter API Pull

In [1]:

```
# pulling in twitter data, and cleaning
import tweepy
import pandas as pd
import numpy as np
from collections import Counter, defaultdict
from nltk.corpus import stopwords
from string import punctuation
import re

sw = stopwords.words("english")

# modeling
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, ENGLISH_STOP_WORDS
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier

# topic modeling
from sklearn.decomposition import LatentDirichletAllocation
import pyLDAvis
import pyLDAvis.sklearn
import pyLDAvis.gensim_models
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import NMF, TruncatedSVD, LatentDirichletAllocation

# remove warnings
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [4]:

```
# bring in secret api keys
auth = tweepy.AppAuthHandler(api_key, api_key_secret)
api = tweepy.API(auth,wait_on_rate_limit = True)
```

In [5]:

```
# create tweet_data dictionary
tweet_data = {}
```

In [6]:

```
# bring in briaankempga tweet data
tweets = api.user_timeline(screen_name="briankempga", count=200, tweet_mode='extended')
all_tweets = []
all_tweets.extend(tweets)
oldest_id = tweets[-1].id
while True:
    tweets = api.user_timeline(screen_name="briankempga", count=200,
     max_id = oldest_id - 1,
     tweet_mode='extended')
    if len(tweets) == 0:
        break
    oldest_id = tweets[-1].id
    all_tweets.extend(tweets)
tweet_data["briankempga"] = all_tweets
```

In [7]:

```
#  bring in gavinnewsom tweet data
tweets = api.user_timeline(screen_name="gavinnewsom", count=200, tweet_mode='extended')
all_tweets = []
all_tweets.extend(tweets)
oldest_id = tweets[-1].id
while True:
    tweets = api.user_timeline(screen_name="gavinnewsom", count=200,
     max_id = oldest_id - 1,
     tweet_mode='extended')
    if len(tweets) == 0:
        break
    oldest_id = tweets[-1].id
    all_tweets.extend(tweets)
tweet_data["gavinnewsom"] = all_tweets
```

In [8]:
```
# place briankempga and gavinnewsom tweet_data dictionary into dataframe
res = []
for key, val in tweet_data.items():
    for item in val:
        res.append([key, item.full_text])
df = pd.DataFrame(res, columns=['id', 'text'])
```

First we extract the last 3000 tweets of two governers. Then we use a classification model to predict the owner of tweet from the tweet by using NLP and modeling

In [9]:
```
df.head()
```

Out[9]:

	id	text
0	briankempga	For years, Stacey Abrams has been aligned with...
1	briankempga	We had a great crowd at our lunch in Winder to...
2	briankempga	RT @GovKemp: https://t.co/YMNGbJKgQq
3	briankempga	RT @TeamKempGA: ICYMI: Watch the latest ad fro...
4	briankempga	Stacey Abrams was paid $52,500 to serve on the...

In [10]:
```
# create a function to clean text data column
def clean_text(txt):
    if type(txt) != str:
        return ""
    txt = txt.lower()
    txt = re.sub('@[A-Za-z0-9_]+', '', txt)
    txt = re.sub('#[A-Za-z0-9_]+', '', txt)
    txt = re.sub(r'http\S+', '', txt)
    txt = txt.replace('\n', ' ')
    txt = re.sub('\s+', ' ', txt)
    txt = re.sub('[^a-z\s]','', txt)
    return txt
```

In [11]:
```
df['clean_text'] = df['text'].apply(clean_text)
```

In [12]:
```
df.head()
```

Out[12]:

	id	text	clean_text
0	briankempga	For years, Stacey Abrams has been aligned with...	for years stacey abrams has been aligned with ...
1	briankempga	We had a great crowd at our lunch in Winder to...	we had a great crowd at our lunch in winder to...
2	briankempga	RT @GovKemp: https://t.co/YMNGbJKgQq	rt
3	briankempga	RT @TeamKempGA: ICYMI: Watch the latest ad fro...	rt icymi watch the latest ad from our campaig...
4	briankempga	Stacey Abrams was paid $52,500 to serve on the...	stacey abrams was paid to serve on the board ...

In [13]:
```
# view value counts of gavinnewsom and briankempga
df['id'].value_counts()
```

Out[13]:
```
gavinnewsom    3249
briankempga    3247
Name: id, dtype: int64
```

In [14]:
```
# split data into train and test sets
train, test = train_test_split(df[['id', 'clean_text']], test_size=0.3,shuffle=True, random_state=123)
train.shape, test.shape
```

Out[14]:
```
((4547, 2), (1949, 2))
```

In [15]:
```
# remove the word which can leak some information about the tweet owner
mystopword = ENGLISH_STOP_WORDS.union(['ca', 'georgians','california', 'georgia', 'rt', 'ga'])
```

In [16]:
```
vec = TfidfVectorizer(stop_words=mystopword)
vec.fit(train['clean_text'])
```

```
X_train = vec.transform(train['clean_text'])
y_train = train['id']
X_test = vec.transform(test['clean_text'])
y_test = test['id']
X_train.shape
```

Out[16]: (4547, 7715)

Descriptive Statistics

In [17]:

```
import string
from string import punctuation

def descriptive_stats(tokens, top_num_tokens = 5, verbose=True) :
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity (https://en.wikipedia.org/wiki/Lexical_diversity),
        and num_tokens most common tokens. Return a list with the number of tokens, number
        of unique tokens, lexical diversity, and number of characters.
    """

    # Fill in the correct values here.

    num_tokens = len(tokens)
    num_unique_tokens = len(set(tokens))
    lexical_diversity = num_unique_tokens/num_tokens
    num_characters = len("".join(tokens))

    if verbose :
        print(f"There are {num_tokens} tokens in the data.")
        print(f"There are {num_unique_tokens} unique tokens in the data.")
        print(f"There are {num_characters} characters in the data.")
        print(f"The lexical diversity is {lexical_diversity:.3f} in the data.")
```

In [18]:

```
descriptive_stats(df.loc[df['id'] == 'gavinnewsom']['clean_text'], verbose = True)
```

```
There are 3249 tokens in the data.
There are 3001 unique tokens in the data.
There are 392152 characters in the data.
The lexical diversity is 0.924 in the data.
```

In [19]:

```
descriptive_stats(df.loc[df['id'] == 'briankempga']['clean_text'], verbose = True)
```

```
There are 3247 tokens in the data.
There are 3212 unique tokens in the data.
There are 446019 characters in the data.
The lexical diversity is 0.989 in the data.
```

Random Forest Model

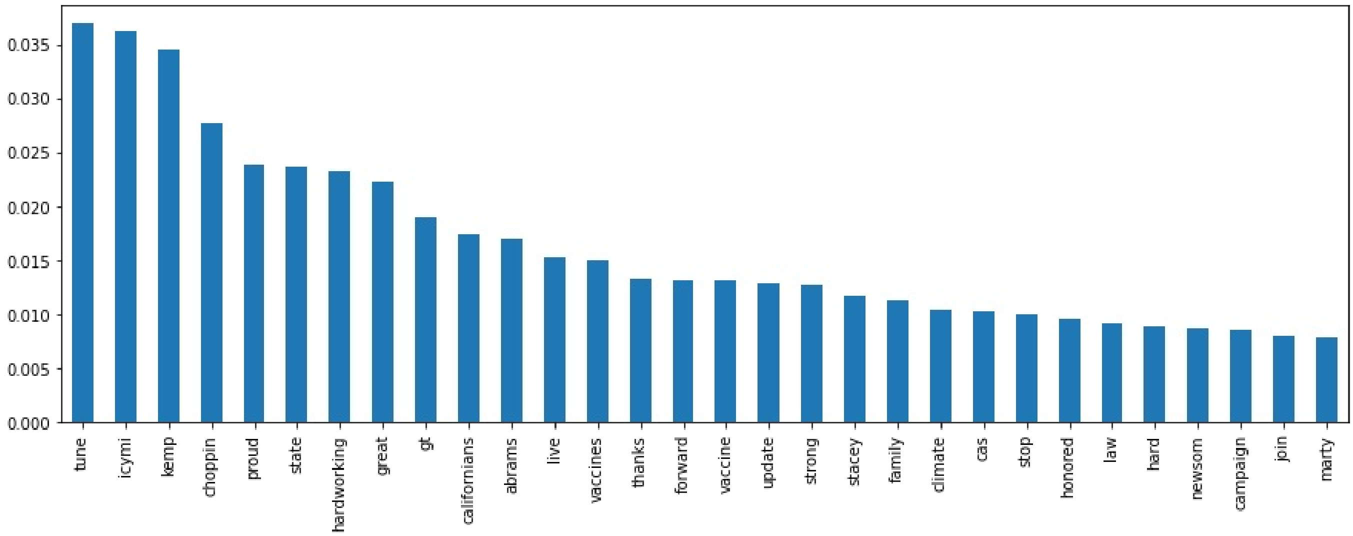
In [15]:
```
rf = RandomForestClassifier(max_depth=8)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

 briankempga       0.87      0.76      0.81       979
gavinnewsom       0.78      0.88      0.83       970

    accuracy                           0.82      1949
   macro avg       0.82      0.82      0.82      1949
weighted avg       0.82      0.82      0.82      1949
```

In [16]:
```
pd.Series(rf.feature_importances_, index=vec.get_feature_names()).\
    sort_values(ascending=False).head(30).plot(kind='bar', figsize=(15,5));
```



XGB

In [17]:
```
xgb = XGBClassifier(max_depth=12, n_estimators=200)
xgb.fit(X_train, 1*(y_train == 'briankempga'))
y_pred = xgb.predict(X_test)
y_pred = ['gavinnewsom' if x == 0 else 'briankempga' for x in y_pred]
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

 briankempga       0.86      0.84      0.85       979
gavinnewsom       0.84      0.86      0.85       970

    accuracy                           0.85      1949
   macro avg       0.85      0.85      0.85      1949
weighted avg       0.85      0.85      0.85      1949
```

Logistic Regression

In [18]:
```
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

Out[18]: LogisticRegression()

In [19]:
```
y_pred = lr.predict(X_test)
```

```
print('Accuracy Score - ', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy Score -  0.8902001026167266
              precision    recall  f1-score   support

 briankempga       0.90      0.88      0.89       979
 gavinnewsom       0.88      0.91      0.89       970

    accuracy                           0.89      1949
   macro avg       0.89      0.89      0.89      1949
weighted avg       0.89      0.89      0.89      1949
```

Naive Bayes Model

In [20]:
```
# Vectorize text reviews to numbers
nb = GaussianNB()
nb.fit(X_train.todense(), y_train)
```

Out[20]:
```
GaussianNB()
```

In [21]:
```
y_pred = nb.predict(X_test.todense())
print('Accuracy Score - ', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy Score -  0.7788609543355567
              precision    recall  f1-score   support

 briankempga       0.73      0.88      0.80       979
 gavinnewsom       0.85      0.68      0.75       970

    accuracy                           0.78      1949
   macro avg       0.79      0.78      0.78      1949
weighted avg       0.79      0.78      0.78      1949
```

Linear SVC Model

In [22]:
```
# HyperTuning Parameters with Validation Set
from sklearn.svm import LinearSVC
lsvc = LinearSVC(verbose=0)
```

In [23]:
```
lsvc.fit(X_train, y_train)
```

Out[23]:
```
LinearSVC()
```

In [24]:
```
y_pred = lsvc.predict(X_test)
print('Accuracy Score - ', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy Score -  0.8907131862493587
              precision    recall  f1-score   support

 briankempga       0.90      0.89      0.89       979
 gavinnewsom       0.89      0.90      0.89       970

    accuracy                           0.89      1949
   macro avg       0.89      0.89      0.89      1949
weighted avg       0.89      0.89      0.89      1949
```

Support Vector Machines

In [25]:

```
svm = svm.SVC(kernel = 'linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print('Accuracy Score -', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy Score - 0.8953309389430477
              precision    recall  f1-score   support

 briankempga       0.91      0.88      0.89       979
 gavinnewsom       0.89      0.91      0.90       970

    accuracy                           0.90      1949
   macro avg       0.90      0.90      0.90      1949
weighted avg       0.90      0.90      0.90      1949
```

K Nearest Neighbors

In [26]:

```
knn_r_acc = []
for i in range(1,17,1):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    test_score = knn.score(X_test, y_test)
    train_score = knn.score(X_train, y_train)
    knn_r_acc.append((i, test_score ,train_score))
df = pd.DataFrame(knn_r_acc, columns=['K','Test Score','Train Score'])
print(df)
```

```
     K  Test Score  Train Score
0    1    0.583376     0.996041
1    2    0.585428     0.991203
2    3    0.539764     0.585661
3    4    0.565418     0.745547
4    5    0.542329     0.621729
5    6    0.552591     0.652078
6    7    0.524885     0.574225
7    8    0.537199     0.593798
8    9    0.520780     0.545854
9   10    0.523858     0.560589
10  11    0.515136     0.528700
11  12    0.516162     0.535518
12  13    0.511544     0.519243
13  14    0.512057     0.521883
14  15    0.508466     0.513086
15  16    0.508466     0.515505
```

In [27]:

```
# the best performing number of neighbors is 1
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)
print('Accuracy Score -', accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy Score - 0.5833760903027193
              precision    recall  f1-score   support

 briankempga       0.90      0.19      0.32       979
 gavinnewsom       0.55      0.98      0.70       970

    accuracy                           0.58      1949
   macro avg       0.72      0.59      0.51      1949
weighted avg       0.73      0.58      0.51      1949
```

Topic Modeling

LDA

In [28]:

```
def display_topics(model, features, no_top_words=5):
    for topic, words in enumerate(model.components_):
        total = words.sum()
```

```
        largest = words.argsort()[::-1] # invert sort order
        print("\nTopic %02d" % topic)
        for i in range(0, no_top_words):
            print("  %s (%2.2f)" % (features[largest[i]], abs(words[largest[i]]*100.0/total)))
```

In [29]:

```
tweet_data = {}
tweets = api.user_timeline(screen_name="briankempga", count=200, tweet_mode='extended')
all_tweets = []
all_tweets.extend(tweets)
oldest_id = tweets[-1].id
while True:
    tweets = api.user_timeline(screen_name="briankempga", count=200,
     max_id = oldest_id - 1,
     tweet_mode='extended')
    if len(tweets) == 0:
        break
    oldest_id = tweets[-1].id
    all_tweets.extend(tweets)
tweet_data["briankempga"] = all_tweets

res = []
for key, val in tweet_data.items():
    for item in val:
        res.append([key, item.full_text])
df = pd.DataFrame(res, columns=['id', 'text'])
```

In [30]:

```
def clean_text(txt):
    if type(txt) != str:
        return ""
    txt = txt.lower()
    txt = re.sub('@[A-Za-z0-9_]+', '', txt)
    txt = re.sub('#[A-Za-z0-9_]+', '', txt)
    txt = re.sub(r'http\S+', '', txt)
    txt = txt.replace('\n', ' ')
    txt = re.sub('\s+', ' ', txt)
    txt = re.sub('[^a-z\s]','', txt)
    return txt

df['clean_text'] = df['text'].apply(clean_text)

# remove the word which can leak some information about the tweet owner
mystopword = ENGLISH_STOP_WORDS.union(['ca', 'georgians','california', 'georgia', 'rt', 'ga'])
vec = TfidfVectorizer(stop_words=mystopword)
vec.fit(df['clean_text'])
tfidf_vals = vec.transform(df['clean_text'])
```

In [31]:

```
lda_text_model = LatentDirichletAllocation(n_components=2, random_state=123)
lda_top = lda_text_model.fit_transform(tfidf_vals)
```

In [32]:

```
display_topics(lda_text_model, vec.get_feature_names())
```

```
Topic 00
  state (0.60)
  amp (0.58)
  choppin (0.42)
  proud (0.37)
  work (0.35)

Topic 01
  great (0.71)
  vote (0.66)
  kemp (0.51)
  choppin (0.45)
  lets (0.43)
```

In [33]:

```
df['topic'] = lda_top.argmax(1)
pd.crosstab(df['topic'], df['id'])
```

Out[33]:

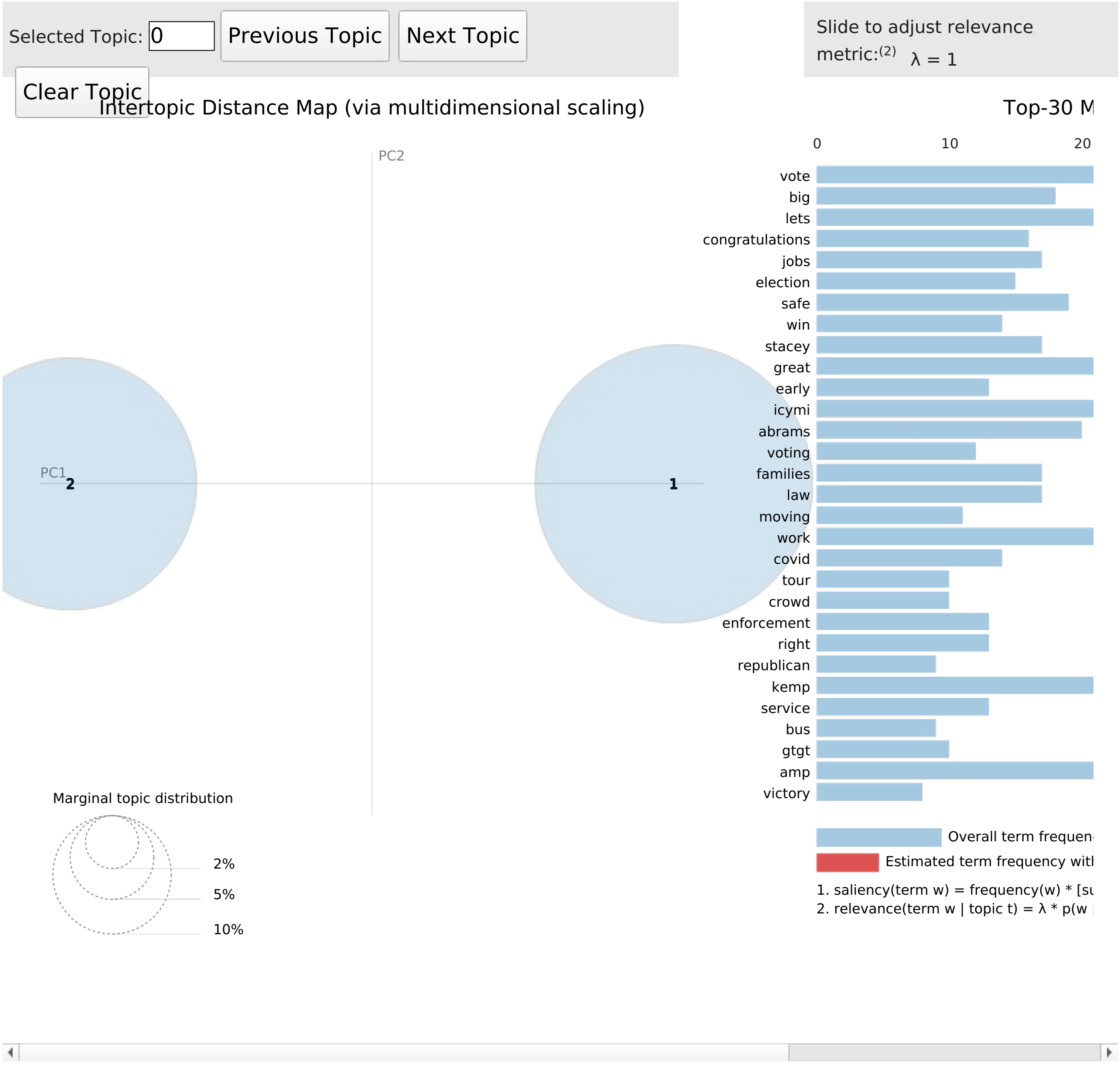
id	briankempga
topic	
0	1741
1	1506

In [34]:

```
lda_display = pyLDAvis.sklearn.prepare(lda_text_model, tfidf_vals, vec, sort_topics=False)
```

```
lda_display = pyLDAvis.sklearn.prepare(lda_text_model, tfidf_vals, vec, sort_topics=False)
pyLDAvis.display(lda_display)
```

Out[34]:



LSA

In [35]:

```
svd_text_model = TruncatedSVD(n_components = 5, random_state=42)
W_svd_text_matrix = svd_text_model.fit_transform(tfidf_vals)
H_svd_text_matrix = svd_text_model.components_
```

In [36]:

```
# call display_topics on your model
display_topics(svd_text_model, vec.get_feature_names())
```

```
Topic 00
  choppin (1.70)
  great (0.98)
  state (0.98)
  support (0.84)
  vote (0.81)

Topic 01
  choppin (6.83)
  lets (1.61)
  thanks (0.85)
  dawgs (0.61)
  wood (0.59)

Topic 02
  vote (13.45)
  kemp (12.63)
```

```
  icymi (6.69)
  governor (6.65)
  brian (5.89)

Topic 03
  support (6.62)
  great (5.59)
  thanks (5.23)
  strong (5.09)
  appreciate (3.24)

Topic 04
  vote (82.73)
  lets (44.08)
  amp (28.84)
  election (25.10)
  early (23.30)
```

Non-Negative Matrix Factorization Model

In [37]:

```
nmf_text_model = NMF(n_components=5, random_state=314)
W_text_matrix = nmf_text_model.fit_transform(tfidf_vals)
H_text_matrix = nmf_text_model.components_

warnings.filterwarnings("ignore", category= FutureWarning)
```

In [38]:

```
display_topics(nmf_text_model, vec.get_feature_names())
```

```
Topic 00
  state (1.70)
  amp (1.49)
  fight (0.77)
  safe (0.72)
  hardworking (0.64)

Topic 01
  choppin (29.23)
  lets (6.53)
  dawgs (2.90)
  wood (2.65)
  congratulations (2.07)

Topic 02
  vote (7.02)
  lets (2.34)
  early (2.25)
  governor (2.15)
  election (2.14)

Topic 03
  support (4.21)
  great (3.93)
  thanks (3.41)
  strong (3.14)
  appreciate (2.10)

Topic 04
  kemp (5.45)
  icymi (5.06)
  gt (3.36)
  brian (3.16)
  gtgt (2.59)
```