Authorship of the Federalist Papers

March 1, 2019

1 Authorship of the Federalist Papers

The *Federalist Papers* were a set of 85 essays published between 1787 and 1788 to promote the ratification of the United States Constitution. They were originally published under the pseudonym "Publius". Although the identity of the authors was a closely guarded secret at the time, most of the papers have since been conclusively attributed to one of Hamilton, Jay, or Madison. The known authorships can be found in `/data301/data/federalist/authorship.csv`.

For 15 of the papers, however, the authorships remain disputed. (These papers can be identified from the `authorship.csv` file because the "Author" field is blank.) In this analysis, you will train a classifier on the papers with known authorships and use your classifier to predict the authorships of the disputed papers. The text of each paper can be found in the `/data301/data/federalist/` directory. The name of the file indicates the number of the paper.

1.1 Question 1

When analyzing an author's style, common words like "the" and "on" are actually more useful than rare words like "hostilities". That is because rare words typically signify context. Context is useful if you are trying to find documents about similar topics, but not so useful if you are trying to identify an author's style because different authors can write about the same topic. For example, both Dr. Seuss and Charles Dickens used rare words like "chimney" and "stockings" in *How the Grinch Stole Christmas* and *A Christmas Carol*, respectively. But they used common words very differently: Dickens used the word "upon" over 100 times, while Dr. Seuss did not use "upon" at all.

Read in the Federalist Papers. Convert each one into a vector of term frequencies. In order to restrict to common words, include only the top 50 words. Then, train a k-nearest neighbors model on the documents with known authorship. Determine an optimal value of k (it's up to you to decide what's "optimal").

Report an estimate of the test accuracy, precision, and recall of your model.

```
In [1]: %matplotlib inline
        import pandas as pd

        papers = pd.read_csv("/data301/data/federalist/authorship.csv")
        papers.head()
```

```
Out[1]:    Paper    Author
        0      1  Hamilton
```

```
1       2       Jay
2       3       Jay
3       4       Jay
4       5       Jay
```

In [2]:
```python
import re
from collections import Counter

texts = []

for i in range (1, 86):
    file_name = "/data301/data/federalist/" + str(i) + ".txt"
    file = open(file_name)
    text = file.read()
    texts.extend([text])

papers.loc[:, "Text"] = texts
papers.head()
```

Out[2]:
```
   Paper    Author                                               Text
0      1  Hamilton  To the People of the State of New York:\n\nAFT...
1      2       Jay  To the People of the State of New York:\n\nWHE...
2      3       Jay  To the People of the State of New York:\n\nIT ...
3      4       Jay  To the People of the State of New York:\n\nMY ...
4      5       Jay  To the People of the State of New York:\n\nQUE...
```

In [3]:
```python
from collections import Counter

bow = (
    papers.loc[:, "Text"].
    str.lower().
    str.replace("[^A-Za-z\s]", "").
    str.split()
).apply(Counter)

papers.loc[:, "bow"] = bow
papers.head()
```

Out[3]:
```
   Paper    Author                                               Text  \
0      1  Hamilton  To the People of the State of New York:\n\nAFT...
1      2       Jay  To the People of the State of New York:\n\nWHE...
2      3       Jay  To the People of the State of New York:\n\nIT ...
3      4       Jay  To the People of the State of New York:\n\nMY ...
4      5       Jay  To the People of the State of New York:\n\nQUE...

                                                 bow
0  {'to': 72, 'the': 132, 'people': 6, 'of': 106,...
1  {'to': 53, 'the': 107, 'people': 22, 'of': 83,...
2  {'to': 56, 'the': 93, 'people': 8, 'of': 62, '...
```

```
3  {'to': 51, 'the': 86, 'people': 8, 'of': 72, '...
4  {'to': 45, 'the': 66, 'people': 3, 'of': 53, '...
```

In [4]:
```python
from sklearn.feature_extraction.text import CountVectorizer

tf = pd.DataFrame(list(bow))
tf.fillna(0, inplace=True)

count_vec = CountVectorizer(max_features=50)
tf_sparse = count_vec.fit_transform(papers["Text"])
tf_sparse.todense()
```

Out[4]:
```
matrix([[  9,  11,  40, ...,  25,   6,   2],
        [  4,   1,  83, ...,   2,  13,   5],
        [  4,   3,  60, ...,  24,  10,   2],
        ...,
        [ 28,  20, 121, ...,  24,  30,  48],
        [ 14,  15,  89, ...,  38,  14,  18],
        [ 13,  20,  73, ...,  16,   8,   6]], dtype=int64)
```

In [5]:
```python
tf_df = pd.DataFrame(tf_sparse.todense(),columns=count_vec.vocabulary_)
master = pd.concat([papers, tf_df], axis=1).set_index("Paper")
master.head()
```

Out[5]:
```
         Author                                              Text  \
Paper
1      Hamilton  To the People of the State of New York:\n\nAFT...
2           Jay  To the People of the State of New York:\n\nWHE...
3           Jay  To the People of the State of New York:\n\nIT ...
4           Jay  To the People of the State of New York:\n\nMY ...
5           Jay  To the People of the State of New York:\n\nQUE...

                                                     bow  to  the  people  of  \
Paper
1      {'to': 72, 'the': 132, 'people': 6, 'of': 106,...   9   11      40   6
2      {'to': 53, 'the': 107, 'people': 22, 'of': 83,...   4    1      83   1
3      {'to': 56, 'the': 93, 'people': 8, 'of': 62, '...   4    3      60   5
4      {'to': 51, 'the': 86, 'people': 8, 'of': 72, '...   4    3      90   5
5      {'to': 45, 'the': 66, 'people': 3, 'of': 53, '...   4    4      72   3

       state  an  government ...  all  but  more  power  one  would  them  \
Paper                        ...
1         12  10           8 ...    2    6    14      9   72      8    18
2          6  16          10 ...    4   22    14      2   53      5    11
3          8  24           1 ...    8    5     6      6   56      0    11
4         11  20           2 ...   12   17     1      4   51     10    10
5          3   3           4 ...   11   11     6      9   45      5    10

       other  can  no
```

```
Paper
1           25    6   2
2            2   13   5
3           24   10   2
4           15   12  17
5            7   11  37

[5 rows x 53 columns]
```

In [6]:
```python
tf_y_train = master.Author.loc[master.Author.notnull()]

tf_x_train = (master.drop(["Author", "Text", "bow"], axis=1).
              loc[master.Author.notnull()])
```

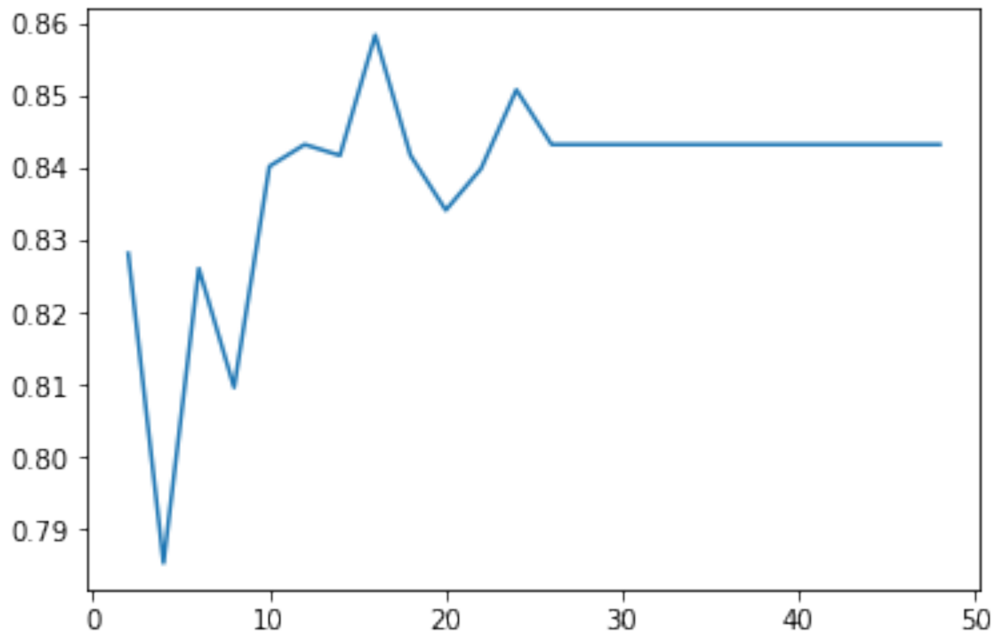
In [7]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score

def calc_f1_for_hamilton(k):
    pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("fit", KNeighborsClassifier(n_neighbors=k))])
    return cross_val_score(pipeline, tf_x_train.astype(float),
                           (tf_y_train == "Hamilton"), cv=10,
                           scoring="f1").mean()
```

In [8]:
```python
ks = pd.Series(range(2, 50, 2))
ks.index = range(2, 50, 2)

hamilton_errs = ks.apply(calc_f1_for_hamilton)
hamilton_errs.plot.line()
hamilton_errs.sort_values(ascending=False)[:5]
```

Out[8]:
```
16    0.858442
24    0.850866
32    0.843290
12    0.843290
46    0.843290
dtype: float64
```




```
In [9]: model = KNeighborsClassifier(n_neighbors=16)
        pipeline = Pipeline([("scaler", StandardScaler()), ("fit", model)])
        pipeline.fit(tf_x_train.astype(float), tf_y_train)
        tf_y_pred = pipeline.predict(tf_x_train.astype(float))
```

```
In [10]: from sklearn.metrics import accuracy_score, precision_score, recall_score

         (accuracy_score(tf_y_train, tf_y_pred),
          precision_score(tf_y_train, tf_y_pred, average=None),
          recall_score(tf_y_train, tf_y_pred, average=None)
         )
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricW
  'precision', 'predicted', average, warn_for)
```

```
Out[10]: (0.74285714285714288,
          array([ 0.74626866,  0.        ,  0.66666667]),
          array([ 0.98039216,  0.        ,  0.14285714]))
```

```
In [11]: tf_y_pred
```

```
Out[11]: array(['Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
```

```
'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
'Madison', 'Hamilton', 'Hamilton', 'Hamilton', 'Madison',
'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
'Hamilton', 'Hamilton', 'Hamilton', 'Madison', 'Hamilton'], dtype=object)
```

Process Notes: I evaluated the best value of k based on the F1 score of Hamilton, since he comprises over half the dataset.

This model achieves about 83% accuracy for predicting the authors. The precision and recall scores for Jay are both 0 because he was never predicted as an author; presumably, the training data for him was too sparse to construct a reliable understanding of his writing style. Furthermore, we can deduce from the fact that Hamilton's precision score is lower than Madison's and Hamilton's recall score is higher than Madison's that this model overpredicted for Hamilton. The value counts of the predictions support this conclusion.

1.2 Question 2

What if we used TF-IDF on the top 50 words instead of the term frequencies? Repeat Question 1, using TF-IDF instead of TF. Which approach is better: TF-IDF or TF?

In [12]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf_vec = TfidfVectorizer(max_features=50)
tf_idf_sparse = tf_idf_vec.fit_transform(papers["Text"])
tf_idf_sparse.todense()
```

Out[12]:
```
matrix([[ 0.0435466 ,  0.05322362,  0.19354043, ...,  0.12096277,
          0.02903106,  0.00967702],
        [ 0.02085212,  0.00521303,  0.4326814 , ...,  0.01042606,
          0.06776938,  0.02606514],
        [ 0.02457136,  0.01842852,  0.36857033, ...,  0.14742813,
          0.06142839,  0.01228568],
        ...,
        [ 0.03787873,  0.02705624,  0.16369023, ...,  0.03246748,
          0.04058435,  0.06493497],
        [ 0.02471767,  0.02648322,  0.15713376, ...,  0.06709082,
          0.02471767,  0.03177986],
        [ 0.03645784,  0.05608899,  0.2047248 , ...,  0.04487119,
          0.02243559,  0.0168267 ]])
```

In [13]:
```python
tf_idf_df = pd.DataFrame(tf_idf_sparse.todense(),columns=tf_idf_vec.vocabulary_)
tf_idf_train_df = pd.concat([papers, tf_idf_df], axis=1).set_index("Paper")
tf_idf_train_df.head()
```

Out[13]:
```
          Author                                               Text  \
Paper
1       Hamilton  To the People of the State of New York:\n\nAFT...
2            Jay  To the People of the State of New York:\n\nWHE...
3            Jay  To the People of the State of New York:\n\nIT ...
4            Jay  To the People of the State of New York:\n\nMY ...
5            Jay  To the People of the State of New York:\n\nQUE...

                                                     bow        to       the  \
Paper
1       {'to': 72, 'the': 132, 'people': 6, 'of': 106,...  0.043547  0.053224
2       {'to': 53, 'the': 107, 'people': 22, 'of': 83,...  0.020852  0.005213
3       {'to': 56, 'the': 93, 'people': 8, 'of': 62, '...  0.024571  0.018429
4       {'to': 51, 'the': 86, 'people': 8, 'of': 72, '...  0.022780  0.017085
5       {'to': 45, 'the': 66, 'people': 3, 'of': 53, '...  0.028122  0.028122

          people        of     state        an  government    ...          all  \
Paper                                                         ...
1       0.193540  0.029714  0.058062  0.048385    0.038708    ...     0.009790
2       0.432681  0.005336  0.031278  0.083408    0.052130    ...     0.021096
3       0.368570  0.031437  0.049143  0.147428    0.006143    ...     0.049717
4       0.512543  0.029145  0.062644  0.113898    0.011390    ...     0.069138
5       0.506198  0.021588  0.021092  0.021092    0.028122    ...     0.078240

             but      more     power       one     would      them     other  \
Paper
1       0.029031  0.067739  0.045621  0.348373  0.040082  0.087093  0.120963
2       0.114687  0.072982  0.010923  0.276291  0.026991  0.057343  0.010426
3       0.030714  0.036857  0.038612  0.343999  0.000000  0.067571  0.147428
4       0.096814  0.005695  0.023865  0.290441  0.058971  0.056949  0.085424
5       0.077336  0.042183  0.066288  0.316374  0.036401  0.070305  0.049214

             can        no
Paper
1       0.029031  0.009677
2       0.067769  0.026065
3       0.061428  0.012286
4       0.068339  0.096814
5       0.077336  0.260130

[5 rows x 53 columns]
```

In [14]:
```
idf_y_train = tf_idf_train_df.Author.loc[tf_idf_train_df.Author.notnull()]

idf_x_train = (tf_idf_train_df.drop(["Author", "Text", "bow"], axis=1)
               .loc[tf_idf_train_df.Author.notnull()])
```

In [15]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

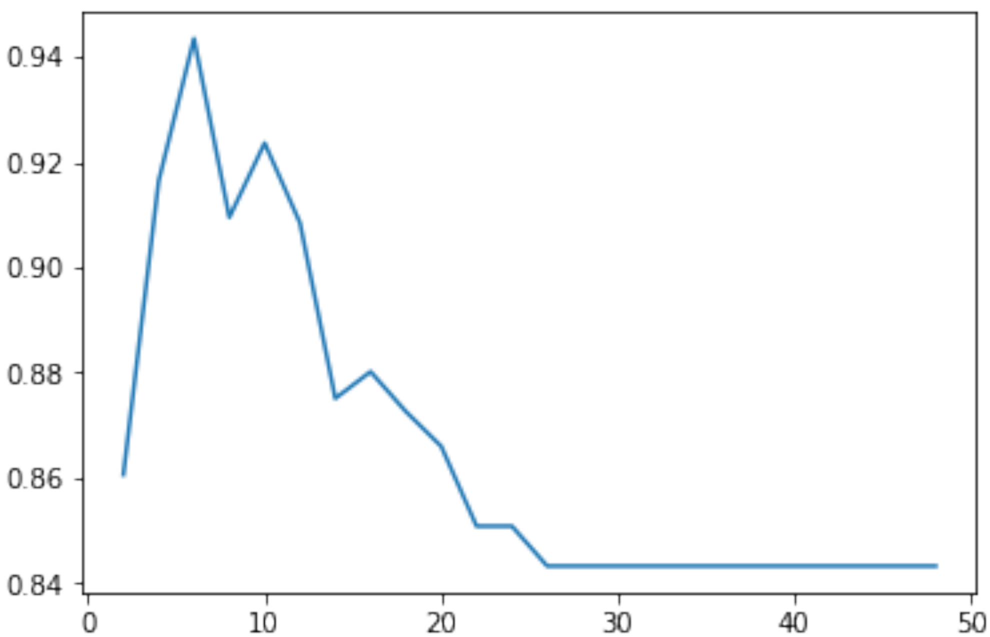
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score

def calc_f1_for_hamilton(k):
    pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("fit", KNeighborsClassifier(n_neighbors=k))])
    return cross_val_score(pipeline, idf_x_train.astype(float),
                           (idf_y_train == "Hamilton"), cv=10,
                           scoring="f1").mean()

ks = pd.Series(range(2, 50, 2))
ks.index = range(2, 50, 2)

hamilton_errs = ks.apply(calc_f1_for_hamilton)
hamilton_errs.plot.line()
hamilton_errs.sort_values(ascending=False)[:5]
```

```
Out[15]: 6     0.943434
         10    0.923593
         4     0.916414
         8     0.909479
         12    0.908442
         dtype: float64
```



In [16]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

model = KNeighborsClassifier(n_neighbors=6)
pipeline = Pipeline([("scaler", StandardScaler()), ("fit", model)])
pipeline.fit(idf_x_train.astype(float), idf_y_train)
idf_y_pred = pipeline.predict(idf_x_train.astype(float))

(accuracy_score(idf_y_train, idf_y_pred),
 precision_score(idf_y_train, idf_y_pred, average=None),
 recall_score(idf_y_train, idf_y_pred, average=None)
)
```

Out[16]:
```
(0.88571428571428568,
 array([ 0.87931034,  1.        ,  0.90909091]),
 array([ 1.        ,  0.2       ,  0.71428571]))
```

Process Notes: I evaluated the best value of *k* based on the F1 score of Hamilton, since he comprises over half the dataset.

This model is about 5% more accurate than the previous model based on term frequencies. From these results, we see that precision scores across the three categories are all quite high. Jay's precision score is extremely high, because he is only predicted once; his recall score is rather low, also because he is only predicted once, even though he authored five papers. Because Hamilton's precision is lower than Madison's precision and Hamilton's recall is higher than Madison's recall, we can conclude that the model overpredicted for Hamilton. We observed this same problem in the previous model.

Process Notes: I attempted 2 approaches to this section. The first technique limits the top 50 words using the `max_features` parameter of the `TfidfVectorizer`. However, I later saw a post on Piazza that pointed out that this parameter limited the data to the 50 most frequently occuring words, rather than the 50 highest ranked words. I tried to obtain the 50 highest-ranked TF-IDF words in my second appraoch. The segment above this comment describes Process 1, and the segment below describes Process 2. The results for Process 2 seemed a little funky to me, so I completed the remainder of the lab using Process 1's results.

In [51]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf_vec = TfidfVectorizer()
tf_idf_vec.fit(papers["Text"])
tf_idf_sorted = sorted(tf_idf_vec.vocabulary_.items(), key=lambda x: x[1])
tf_idf_last_50 = tf_idf_sorted[-50:]
tf_idf_last_50
```

Out[51]:
```
[('workings', 8566),
 ('works', 8567),
 ('world', 8568),
 ('worn', 8569),
 ('worse', 8570),
 ('worst', 8571),
 ('worth', 8572),
 ('worthy', 8573),
 ('would', 8574),
 ('wound', 8575),
 ('wounded', 8576),
 ('wreaked', 8577),
 ('wreck', 8578),
 ('wrest', 8579),
 ('wretched', 8580),
 ('writ', 8581),
 ('write', 8582),
 ('writer', 8583),
 ('writers', 8584),
 ('writing', 8585),
 ('writings', 8586),
 ('written', 8587),
 ('wrong', 8588),
 ('wrongs', 8589),
 ('wrought', 8590),
 ('wyoming', 8591),
 ('xerxes', 8592),
 ('xiv', 8593),
 ('xv', 8594),
 ('yards', 8595),
 ('yates', 8596),
 ('year', 8597),
 ('yearly', 8598),
 ('years', 8599),
 ('yeomanry', 8600),
 ('yes', 8601),
 ('yet', 8602),
```

 ('yield', 8603),
 ('yielding', 8604),
 ('yoke', 8605),
 ('yokes', 8606),
 ('york', 8607),
 ('you', 8608),
 ('young', 8609),
 ('your', 8610),
 ('yourselves', 8611),
 ('zaleucus', 8612),
 ('zeal', 8613),
 ('zealand', 8614),
 ('zealous', 8615)]

In [52]:
```
tf_idf_50_vec = TfidfVectorizer(vocabulary=[i[0] for i in tf_idf_last_50])
tf_idf_sparse = tf_idf_50_vec.fit_transform(papers["Text"])
tf_idf_sparse.todense()
```

Out[52]:
```
matrix([[ 0.        ,  0.        ,  0.0522586 , ...,  0.16292634,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.23953191, ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.2827031 , ...,  0.        ,
          0.        ,  0.        ],
        ...,
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.20838566,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.11873826,
          0.        ,  0.36093022]])
```

In [56]:
```
tf_idf_df = pd.DataFrame(tf_idf_sparse.todense(),columns=tf_idf_50_vec.vocabulary)
tf_idf_train_df = pd.concat([papers, tf_idf_df], axis=1).set_index("Paper")
tf_idf_train_df.head()
```

Out[56]:
```
          Author                                              Text  \
Paper
1       Hamilton  To the People of the State of New York:\n\nAFT...
2            Jay  To the People of the State of New York:\n\nWHE...
3            Jay  To the People of the State of New York:\n\nIT ...
4            Jay  To the People of the State of New York:\n\nMY ...
5            Jay  To the People of the State of New York:\n\nQUE...

                                                     bow  workings  works  \
Paper
1       {'to': 72, 'the': 132, 'people': 6, 'of': 106,...       0.0    0.0
2       {'to': 53, 'the': 107, 'people': 22, 'of': 83,...       0.0    0.0
3       {'to': 56, 'the': 93, 'people': 8, 'of': 62, '...       0.0    0.0
```

```
4      {'to': 51, 'the': 86, 'people': 8, 'of': 72, '...       0.0    0.0
5      {'to': 45, 'the': 66, 'people': 3, 'of': 53, '...       0.0    0.0

          world  worn  worse  worst  worth    ...     yokes      york  \
Paper                                         ...
1      0.052259   0.0    0.0    0.0    0.0    ...       0.0  0.022536
2      0.239532   0.0    0.0    0.0    0.0    ...       0.0  0.103298
3      0.282703   0.0    0.0    0.0    0.0    ...       0.0  0.121915
4      0.000000   0.0    0.0    0.0    0.0    ...       0.0  0.056492
5      0.000000   0.0    0.0    0.0    0.0    ...       0.0  0.025217

            you     young      your  yourselves  zaleucus      zeal  zealand  \
Paper
1      0.497206  0.000000  0.825415    0.000000       0.0  0.162926      0.0
2      0.000000  0.000000  0.000000    0.000000       0.0  0.000000      0.0
3      0.000000  0.000000  0.000000    0.000000       0.0  0.000000      0.0
4      0.178049  0.000000  0.000000    0.000000       0.0  0.000000      0.0
5      0.079478  0.102584  0.277077    0.120063       0.0  0.000000      0.0

       zealous
Paper
1          0.0
2          0.0
3          0.0
4          0.0
5          0.0

[5 rows x 53 columns]
```

In [57]:
```
idf_y_train = tf_idf_train_df.Author.loc[tf_idf_train_df.Author.notnull()]

idf_x_train = (tf_idf_train_df.drop(["Author", "Text", "bow"], axis=1)
               .loc[tf_idf_train_df.Author.notnull()])
```

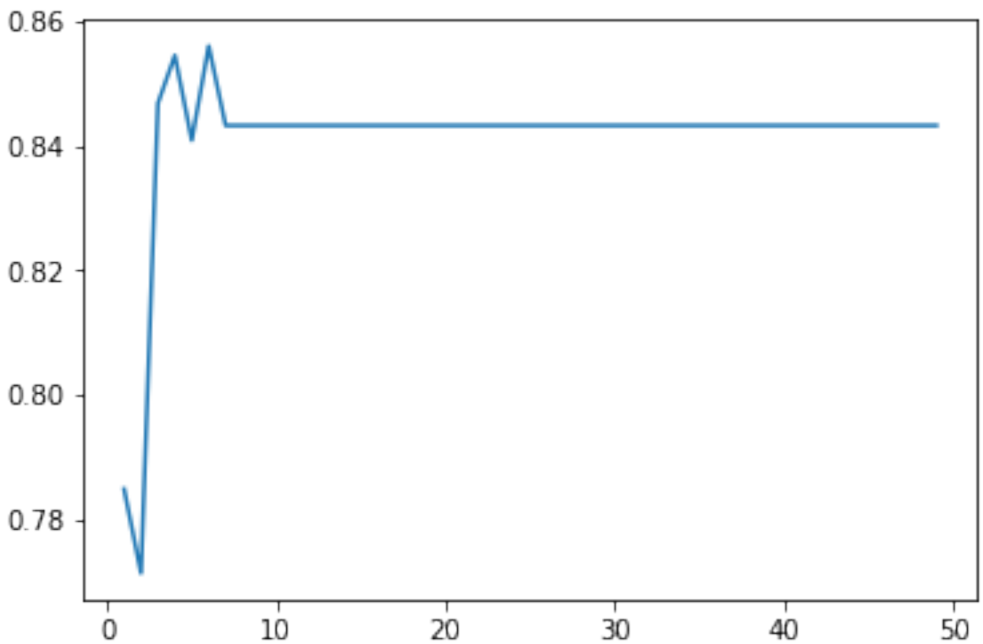
In [58]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score

def calc_f1_for_hamilton(k):
    pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("fit", KNeighborsClassifier(n_neighbors=k))])
    return cross_val_score(pipeline, idf_x_train.astype(float),
                           (idf_y_train == "Hamilton"), cv=10,
                           scoring="f1").mean()
```

```
ks = pd.Series(range(1, 50, 1))
ks.index = range(1, 50, 1)

hamilton_errs = ks.apply(calc_f1_for_hamilton)
hamilton_errs.plot.line()
hamilton_errs.sort_values(ascending=False)[:5]
```

Out[58]:
```
6     0.856061
4     0.854545
3     0.846853
49    0.843290
15    0.843290
dtype: float64
```



In [59]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

model = KNeighborsClassifier(n_neighbors=6)
pipeline = Pipeline([("scaler", StandardScaler()), ("fit", model)])
pipeline.fit(idf_x_train.astype(float), idf_y_train)
idf_y_pred = pipeline.predict(idf_x_train.astype(float))

(accuracy_score(idf_y_train, idf_y_pred),
 precision_score(idf_y_train, idf_y_pred, average=None),
 recall_score(idf_y_train, idf_y_pred, average=None)
)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricW
  'precision', 'predicted', average, warn_for)
```

```
Out[59]: (0.72857142857142854,
          array([ 0.72857143,  0.        ,  0.        ]),
          array([ 1.,  0.,  0.]))
```

```
In [18]: idf_y_pred
```

```
Out[18]: array(['Hamilton', 'Madison', 'Hamilton', 'Jay', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Madison', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Madison', 'Hamilton', 'Madison', 'Madison',
                'Madison', 'Madison', 'Madison', 'Madison', 'Hamilton', 'Hamilton',
                'Madison', 'Madison', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton', 'Hamilton',
                'Hamilton', 'Hamilton'], dtype=object)
```

1.3 Question 3

Using the model that you determined to be best in Questions 1 and 2, fit a *k*-nearest neighbors model to all 70 documents with known authorship. Create a confusion matrix for your model that shows how often you predicted Hamilton, Jay, or Madison, and how often it actually was Hamilton, Jay, or Madison (on the training data, of course).

From your confusion matrix, you should be able to calculate the (training) precision and recall of your model for predicting Hamilton. What is it?

In [17]:
```
model = KNeighborsClassifier(n_neighbors=6)
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("fit", model)])
pipeline.fit(idf_x_train.astype(float), idf_y_train)
idf_y_pred = pipeline.predict(idf_x_train.astype(float))
```

In [18]:
```
from sklearn.metrics import confusion_matrix
pd.DataFrame(confusion_matrix(idf_y_pred, idf_y_train),
             columns=["Hamilton", "Jay", "Madison"],
             index=["Hamilton", "Jay", "Madison"])
```

Out[18]:
```
          Hamilton  Jay  Madison
Hamilton        51    3        4
```

```
Jay               0    1      0
Madison           0    1     10
```

Each row represents the distribution of predictions, whereas each column represents the truth.
Hamilton - Precision: (51+3+4)/59 = 0.8793 - Recall: 51/51 = 1.000

1.4 Question 4

Finally, use the model you trained in Question 3 to predict the authorships of the 15 documents with unknown authors. Summarize what you find.

In [19]:
```
tf_idf_train_df.head()
```

Out[19]:
```
          Author                                               Text  \
Paper
1       Hamilton  To the People of the State of New York:\n\nAFT...
2            Jay  To the People of the State of New York:\n\nWHE...
3            Jay  To the People of the State of New York:\n\nIT ...
4            Jay  To the People of the State of New York:\n\nMY ...
5            Jay  To the People of the State of New York:\n\nQUE...

                                                      bow        to       the  \
Paper
1       {'to': 72, 'the': 132, 'people': 6, 'of': 106,...  0.043547  0.053224
2       {'to': 53, 'the': 107, 'people': 22, 'of': 83,...  0.020852  0.005213
3       {'to': 56, 'the': 93, 'people': 8, 'of': 62, '...  0.024571  0.018429
4       {'to': 51, 'the': 86, 'people': 8, 'of': 72, '...  0.022780  0.017085
5       {'to': 45, 'the': 66, 'people': 3, 'of': 53, '...  0.028122  0.028122

          people        of     state        an  government    ...          all  \
Paper                                                         ...
1       0.193540  0.029714  0.058062  0.048385    0.038708    ...     0.009790
2       0.432681  0.005336  0.031278  0.083408    0.052130    ...     0.021096
3       0.368570  0.031437  0.049143  0.147428    0.006143    ...     0.049717
4       0.512543  0.029145  0.062644  0.113898    0.011390    ...     0.069138
5       0.506198  0.021588  0.021092  0.021092    0.028122    ...     0.078240

             but      more     power       one     would      them     other  \
Paper
1       0.029031  0.067739  0.045621  0.348373  0.040082  0.087093  0.120963
2       0.114687  0.072982  0.010923  0.276291  0.026991  0.057343  0.010426
3       0.030714  0.036857  0.038612  0.343999  0.000000  0.067571  0.147428
4       0.096814  0.005695  0.023865  0.290441  0.058971  0.056949  0.085424
5       0.077336  0.042183  0.066288  0.316374  0.036401  0.070305  0.049214

             can        no
Paper
1       0.029031  0.009677
2       0.067769  0.026065
```

```
3      0.061428  0.012286
4      0.068339  0.096814
5      0.077336  0.260130

[5 rows x 53 columns]
```

In [20]:
```
no_authors = tf_idf_train_df.loc[tf_idf_train_df.Author.isnull()]
final_x_train = no_authors.drop(["Author", "Text", "bow"], axis=1)
final_y_pred = pipeline.predict(final_x_train)

author_pred = pd.concat([pd.Series(no_authors.index),
                         pd.Series(final_y_pred)],
                        axis=1).set_index("Paper")
author_pred.columns = ["Author"]
author_pred
```

Out[20]:
```
         Author
Paper
18     Hamilton
19     Hamilton
20     Hamilton
49     Hamilton
50     Hamilton
51      Madison
52     Hamilton
53      Madison
54      Madison
55     Hamilton
56     Hamilton
57      Madison
58      Madison
62     Hamilton
63     Hamilton
```

According to the TF-IDF model, Hamilton wrote 10 of the remaining papers, and Madison wrote 5 of them.

2 Submission Instructions

Once you are finished, follow these steps:

1. Restart the kernel and re-run this notebook from beginning to end by going to `Kernel > Restart Kernel and Run All Cells`.
2. If this process stops halfway through, that means there was an error. Correct the error and repeat Step 1 until the notebook runs from beginning to end.
3. Double check that there is a number next to each code cell and that these numbers are in order.

Then, submit your lab as follows:

1. Go to `File > Export Notebook As > PDF`.
2. Double check that the entire notebook, from beginning to end, is in this PDF file. (If the notebook is cut off, try first exporting the notebook to HTML and printing to PDF.)
3. Upload the PDF to PolyLearn.