Load the Data from Source

↳ 已隐藏 1 个单元格

Global Imports

↳ 已隐藏 1 个单元格

Load data

↳ 已隐藏 1 个单元格

Split data into training (75%) and testing (25%) sets

↳ 已隐藏 1 个单元格

Build pipeline

↳ 已隐藏 1 个单元格

Grid search

↳ 已隐藏 1 个单元格

Print and plot confusion matrix

↳ 已隐藏 1 个单元格

Explore the scikit-learn TfidVectorizer class

- Run the TfidVectorizer class on the training data above (docs_train).
- Explore the min_df and max_df parameters of TfidVectorizer.
- Explore the ngram_range parameter of TfidVectorizer.

Parameters in tf-idf Vectorizer

- **min_df**: filter all terms with frequency lower than this value in any document
- **max-df**: filter all terms with frequecy greater than this value in any document, used to filter out stop words.
- **n-gram range**: If n-gram range = (m,M), build a vocabulary of ALL n-grams of length m through M

Test tf-idf vectorizer object on training set

```
tfidfv = TfidfVectorizer()
tfidfv = tfidfv.set_params(max_df=0.75, max_features= 5000, use_idf= True, smooth_idf=True, sublinear_tf = True)
```

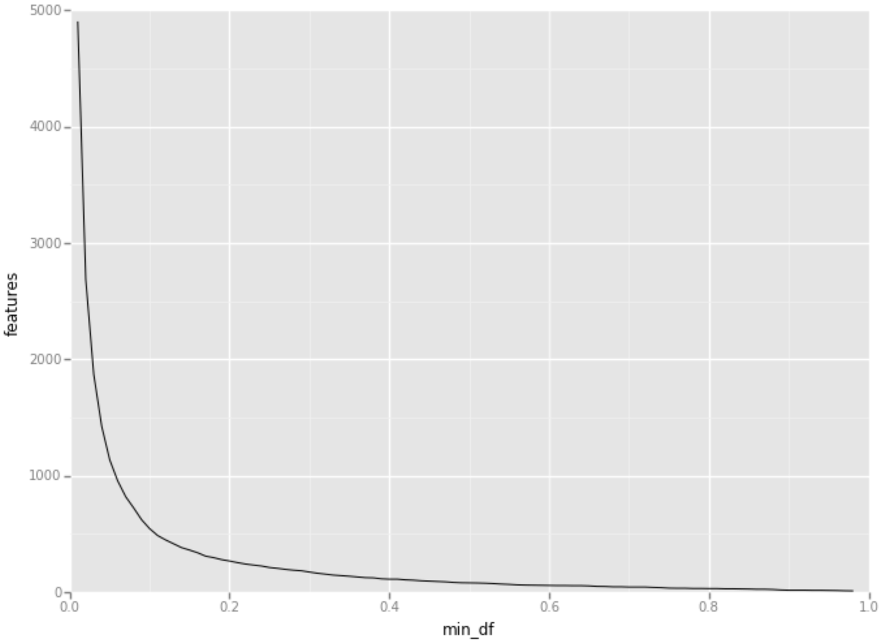
```
t0 = time()
vectors = tfidfv.fit_transform(docs_train)
print("done in %0.3fs" % (time() - t0))
```

```
done in 2.181s
```

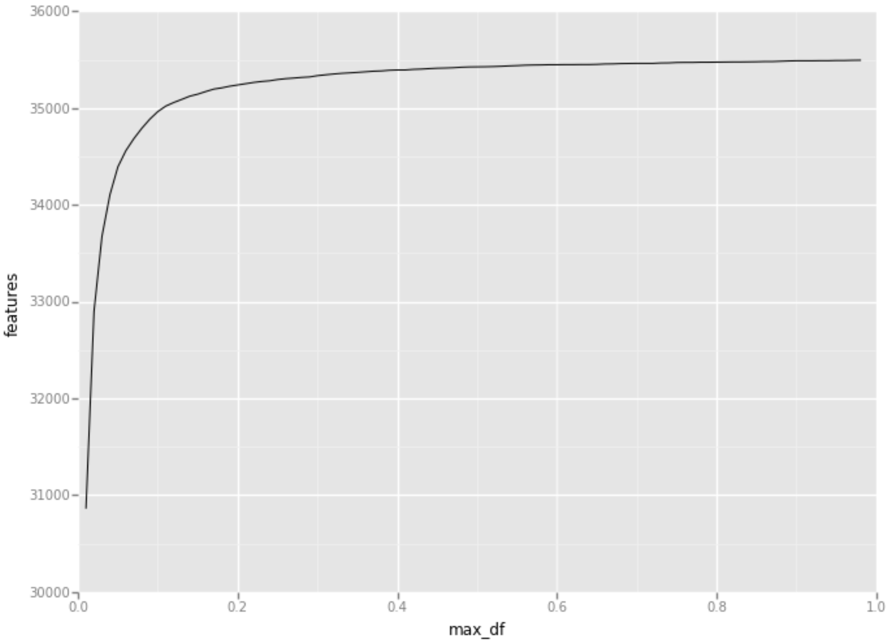
Explore how the min_df and max_df change the number of features we get

```
import numpy as np
value_range=np.arange(0.01,0.99,0.01)

# Calculate the number of features in the library for each value of min_df and max_df in the given range
y1=[TfidfVectorizer(min_df=x).fit_transform(docs_train).shape[1] for x in value_range]
y2=[TfidfVectorizer(max_df=x).fit_transform(docs_train).shape[1] for x in value_range]

# Plot min_df and max_df versus the number of tokens in the vocabulary
from ggplot import *
print qplot(value_range,y=y1,geom='line')+xlab('min_df')+ylab('features')
print qplot(value_range,y=y2,geom='line')+xlab('max_df')+ylab('features')
```

```
C:\Users\Tyler\AppData\Local\Enthought\Canopy\User\lib\site-packages\matplotlib\__init__.py:872: UserWarning: axes.color_cycle is deprecat
  warnings.warn(self.msg_depr % (key, alt_key))
```




```
<ggplot: (24655318)>
```

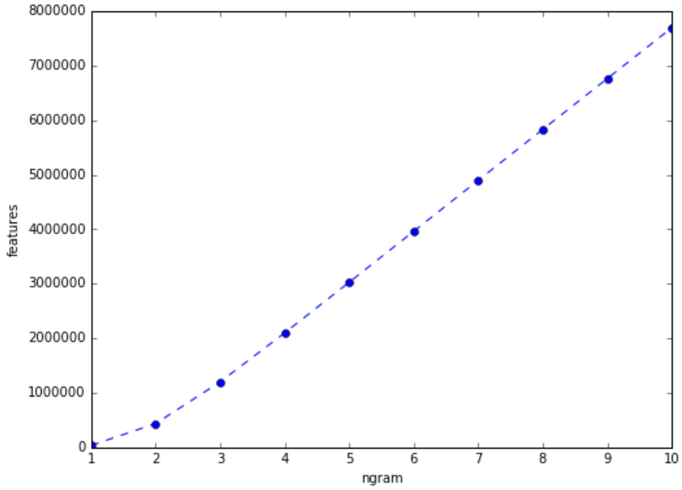



```
<ggplot: (23096818)>
```

Explore how the ngram_range change the number of features we get

```
x=[1 for i in range(10)]
y=np.arange(10)+1

# We allow ngram_range to range in the form (1 , ngram) for ngram between 1 and 10
parameter=zip(x,y)

# Calculate the number of tokens in the vocabulary
y3=[TfidfVectorizer(ngram_range=i).fit_transform(docs_train).shape[1] for i in parameter]

# Plot the number of features verses the (1,ngram) range
fig=plt.figure(figsize=(8,6))
plt.plot([1,2,3,4,5,6,7,8,9,10],y3,'b--o')
plt.xlabel('ngram')
plt.ylabel('features')
```

<matplotlib.text.Text at 0x164a1f28>



Observe how the parameters min_df , max_df, and ngram_range affect the predictiona ccuracy of classification algorithms.

```
#setting max_df and n_gram_range as default, we choose min_df in [1,2,3,4,5] seperately,
#and store the corresponding Xtrain and Xtest into min_df_data array.

min_df_data=[(TfidfVectorizer(min_df=i).fit_transform(docs_train).toarray(),
TfidfVectorizer(min_df=i).fit(docs_train).transform(docs_test).toarray()) for i in [1,3,5,7]]
```

```
#setting min_df and n_gram_range as default, we choose max_df in [0.40,0.5, 0.60, 0.7] seperately,
#and store the corresponding Xtrain and Xtest into max_df_data array.

max_df_data=[(TfidfVectorizer(max_df=i).fit_transform(docs_train).toarray(),
TfidfVectorizer(max_df=i).fit(docs_train).transform(docs_test).toarray()) for i in [0.40,0.5, 0.60, 0.7]]
```

```
#setting min_df and max_df as default, we choose ngram_range in [(1,1),(1,2)] seperately,
#and store the corresponding Xtrain and Xtest into ngram_range_data array.

ngram_range_data=[(TfidfVectorizer(ngram_range=i).fit_transform(docs_train),
TfidfVectorizer(ngram_range=i).fit(docs_train).transform(docs_test)) for i in [(1,1),(1,2)]]
```

```
# explore parameters in tfidf for both linear SVC and KNN
param_grid = [
  {'C': [1]},
   ]
grid_search = GridSearchCV(LinearSVC(), param_grid, n_jobs=1, verbose=1)

# For each XTrain and XTest generated above (for the varying parameters) fit a linear SVC on XTrain and use that to predict
# on X_Test
min_df_fit=[grid_search.fit(i[0],y_train).predict(i[1]) for i in min_df_data ]
max_df_fit=[grid_search.fit(i[0],y_train).predict(i[1]) for i in max_df_data ]
ngram_range_fit=[grid_search.fit(i[0],y_train).predict(i[1]) for i in ngram_range_data]

# Determine the prediction accuracy for each model (separated per-parameter)

min_df_svc_score=[metrics.accuracy_score(min_df_fit[i],y_test) for i in range(4)]
max_df_svc_score=[metrics.accuracy_score(max_df_fit[i],y_test) for i in range(4)]
ngram_range_svc_score=[metrics.accuracy_score(ngram_range_fit[i],y_test) for i in range(2)]
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  2.4min finished
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:   30.8s finished

Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.8s finished
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.0s finished

Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:   47.4s finished
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:   34.3s finished

Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 10.2min finished
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  3.5min finished

Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    2.0s finished
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:   13.3s finished
```

```
from sklearn.neighbors import KNeighborsClassifier
```

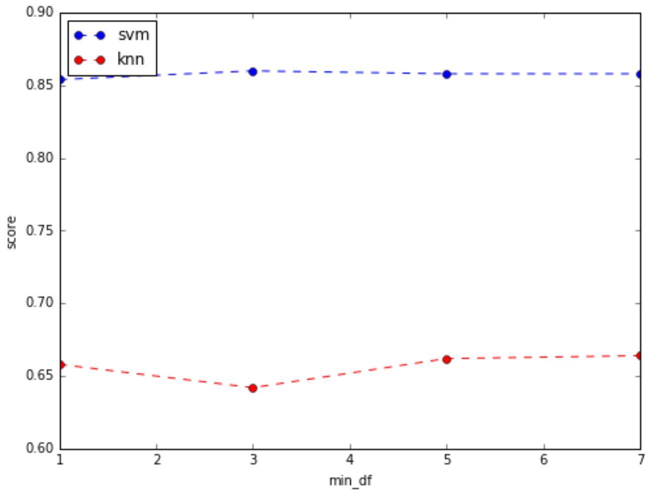
```
param_grid = [
  {'n_neighbors': [1,4]},
 ]
grid_search1 = GridSearchCV(KNeighborsClassifier(), param_grid, n_jobs=1, verbose=1)

# For each XTrain and XTest generated above (for the varying parameters) fit KNN on XTrain and use that to predict
# on X_Test.  We also try K = 1 and 4.

min_df_fit1=[grid_search1.fit(i[0],y_train).predict(i[1]) for i in min_df_data ]
max_df_fit1=[grid_search1.fit(i[0],y_train).predict(i[1]) for i in max_df_data ]
ngram_range_fit1=[grid_search1.fit(i[0],y_train).predict(i[1]) for i in ngram_range_data]
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
Fitting 3 folds for each of 2 candidates, totalling 6 fits[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed: 13.6min finished
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  8.5min finished

Fitting 3 folds for each of 2 candidates, totalling 6 fits
Fitting 3 folds for each of 2 candidates, totalling 6 fits[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  3.1min finished
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  3.1min finished

Fitting 3 folds for each of 2 candidates, totalling 6 fits
Fitting 3 folds for each of 2 candidates, totalling 6 fits[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  9.0min finished
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  8.9min finished

Fitting 3 folds for each of 2 candidates, totalling 6 fits
Fitting 3 folds for each of 2 candidates, totalling 6 fits[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed: 11.8min finished
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:  9.3min finished

Fitting 3 folds for each of 2 candidates, totalling 6 fits
Fitting 3 folds for each of 2 candidates, totalling 6 fits[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    5.6s finished
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    6.7s finished
```

```
# Determine the prediction accuracy for each model (separated per-parameter)

min_df_knn_score=[metrics.accuracy_score(min_df_fit1[i],y_test) for i in range(4)]
max_df_knn_score=[metrics.accuracy_score(max_df_fit1[i],y_test) for i in range(4)]
ngram_range_knn_score=[metrics.accuracy_score(ngram_range_fit1[i],y_test) for i in range(2)]
```

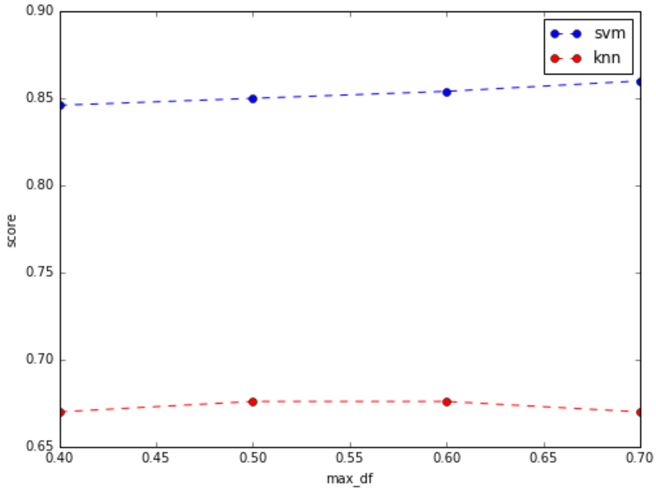
```
import matplotlib.pyplot as plt

# Plot prediction accuracy of KNN and SVC models versus the min_df value.

fig=plt.figure(figsize=(8,6))
plt.plot([1,3,5,7], min_df_svc_score, 'bo--',label='svm')
plt.plot([1,3,5,7], min_df_knn_score, 'ro--',label='knn')
plt.legend(loc='best')
plt.xlabel('min_df')
plt.ylabel('score')
```

```
<matplotlib.text.Text at 0xe1f1400>
```




```
fig=plt.figure(figsize=(8,6))

# Plot prediction accuracy of KNN and SVC models versus the max_df value.

plt.plot([0.40,0.5, 0.60, 0.7], max_df_svc_score, 'bo--',label='svm')
plt.plot([0.40,0.5, 0.60, 0.7], max_df_knn_score, 'ro--',label='knn')
plt.legend(loc='best')
plt.xlabel('max_df')
plt.ylabel('score')
```

<matplotlib.text.Text at 0xe25c7b8>




```
fig=plt.figure(figsize=(8,6))

# Plot prediction accuracy of KNN and SVC models versus the ngram_range.

plt.plot([1,2], ngram_range_svc_score, 'bo--',label='svm')
plt.plot([1,2], ngram_range_knn_score, 'ro--',label='knn')
plt.legend(loc='best')
plt.xlabel('ngram_range = (1,ngram)')
plt.ylabel('score')
```

<matplotlib.text.Text at 0xe26edd8>