requirements:

numpy, pandas, nltk, scikit-learn, matplotlib, seaborn

In [1]:
```
#import libraries
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import names
from nltk.stem import WordNetLemmatizer

import glob
import os
import numpy as np
```

In [2]:
```python
#Test raw data
file_path = 'enron1/ham/0007.1999-12-14.farmer.ham.txt'
with open(file_path, 'r') as infile:
    ham_sample = infile.read()
print(ham_sample)

file_path = 'enron1/spam/0058.2003-12-21.GP.spam.txt'
with open(file_path, 'r') as infile:
    spam_sample = infile.read()
print(spam_sample)
```

```
Subject: mcmullen gas for 11 / 99
jackie ,
since the inlet to 3 river plant is shut in on 10 / 19 / 99 ( the last day of
flow ) :
at what meter is the mcmullen gas being diverted to ?
at what meter is hpl buying the residue gas ? ( this is the gas from teco ,
vastar , vintage , tejones , and swift )
i still see active deals at meter 3405 in path manager for teco , vastar ,
vintage , tejones , and swift
i also see gas scheduled in pops at meter 3404 and 3405 .
please advice . we need to resolve this as soon as possible so settlement
can send out payments .
thanks
Subject: stacey automated system generating 8 k per week parallelogram
people are
getting rich using this system ! now it ' s your
turn !
we ' ve
cracked the code and will show you . . . .
this is the
only system that does everything for you , so you can make
money
. . . . . . . .
because your
success is . . . completely automated !
let me show
you how !
click
here
to opt out click here % random _ text
```

In [3]:
```
#import all data files
emails, labels = [], []

file_path = 'enron1/spam/'
for filename in glob.glob(os.path.join(file_path, '*.txt')):
    with open(filename, 'r', encoding = "ISO-8859-1") as infile:
        emails.append(infile.read())
        labels.append(1)

file_path = 'enron1/ham/'
for filename in glob.glob(os.path.join(file_path, '*.txt')):
    with open(filename, 'r', encoding = "ISO-8859-1") as infile:
        emails.append(infile.read())
        labels.append(0)
```

In [4]:
```
# preprocess and clean the raw text data, includes:
# 1) Number and punctuation removal
# 2) Human name removal (optional)
# 3) Stop words removal
# 4) Lemmatization
def letters_only(astr):
    return astr.isalpha()


all_names = set(names.words())
lemmatizer = WordNetLemmatizer()


def clean_text(docs):
    cleaned_docs = []
    for doc in docs:
        cleaned_docs.append(' '.join([lemmatizer.lemmatize(word.lower())
                                        for word in doc.split()
                                        if letters_only(word)
                                        and word not in all_names]))
    return cleaned_docs


cv = CountVectorizer(stop_words="english", max_features=500)

cleaned_emails = clean_text(emails)
term_docs = cv.fit_transform(cleaned_emails)
print(term_docs [0])
```

```
  (0, 481)	1
  (0, 357)	1
  (0, 69)	1
  (0, 285)	1
  (0, 424)	1
  (0, 250)	1
  (0, 345)	1
  (0, 445)	1
  (0, 231)	1
  (0, 497)	1
  (0, 47)	1
  (0, 178)	2
  (0, 125)	2
```

In [5]:
```
feature_mapping = cv.vocabulary
feature_names = cv.get_feature_names()

def get_label_index(labels):
    from collections import defaultdict
    label_index = defaultdict(list)
    for index, label in enumerate(labels):
        label_index[label].append(index)
    return label_index


def get_prior(label_index):
    """ Compute prior based on training samples
    Args:
        label_index (grouped sample indices by class)
    Returns:
        dictionary, with class label as key, corresponding prior as the value
    """
    prior = {label: len(index) for label, index in label_index.items()}
    total_count = sum(prior.values())
    for label in prior:
        prior[label] /= float(total_count)
    return prior


def get_likelihood(term_document_matrix, label_index, smoothing=0):
    """ Compute likelihood based on training samples
    Args:
        term_document_matrix (sparse matrix)
        label_index (grouped sample indices by class)
        smoothing (integer, additive Laplace smoothing parameter)
    Returns:
        dictionary, with class as key, corresponding conditional probability P
(feature|class) vector as value
    """
    likelihood = {}
    for label, index in label_index.items():
        likelihood[label] = term_document_matrix[index, :].sum(axis=0) + smoot
hing
        likelihood[label] = np.asarray(likelihood[label])[0]
        total_count = likelihood[label].sum()
        likelihood[label] = likelihood[label] / float(total_count)
    return likelihood

feature_names[:5]
```

Out[5]:
```
['able', 'access', 'account', 'accounting', 'act']
```

In [6]:
```
def get_posterior(term_document_matrix, prior, likelihood):
    """ Compute posterior of testing samples, based on prior and likelihood
    Args:
        term_document_matrix (sparse matrix)
        prior (dictionary, with class label as key, corresponding prior as the
 value)
        likelihood (dictionary, with class label as key, corresponding conditi
onal probability vector as value)
    Returns:
        dictionary, with class label as key, corresponding posterior as value
    """
    num_docs = term_document_matrix.shape[0]
    posteriors = []
    for i in range(num_docs):
        # posterior is proportional to prior * likelihood
        # = exp(log(prior * likelihood))
        # = exp(log(prior) + log(likelihood))
        posterior = {key: np.log(prior_label) for key, prior_label in prior.it
ems()}
        for label, likelihood_label in likelihood.items():
            term_document_vector = term_document_matrix.getrow(i)
            counts = term_document_vector.data
            indices = term_document_vector.indices
            for count, index in zip(counts, indices):
                posterior[label] += np.log(likelihood_label[index]) * count
        # exp(-1000):exp(-999) will cause zero division error,
        # however it equates to exp(0):exp(1)
        min_log_posterior = min(posterior.values())
        for label in posterior:
            try:
                posterior[label] = np.exp(posterior[label] - min_log_posterior
)
            except:
                # if one's log value is excessively large, assign it infinity
                posterior[label] = float('inf')
        # normalize so that all sums up to 1
        sum_posterior = sum(posterior.values())
        for label in posterior:
            if posterior[label] == float('inf'):
                posterior[label] = 1.0
            else:
                posterior[label] /= sum_posterior
        posteriors.append(posterior.copy())
    return posteriors


label_index = get_label_index(labels)
prior = get_prior(label_index)

smoothing = 1
likelihood = get_likelihood(term_docs, label_index, smoothing)
```

In [7]:
```
emails_test = [
    '''Subject: flat screens
    hello ,
    please call or contact regarding the other flat screens requested .
    trisha tlapek - eb 3132 b
    michael sergeev - eb 3132 a
    also the sun blocker that was taken away from eb 3131 a .
    trisha should two monitors also michael .
    thanks
    kevin moore''',
    '''Subject: having problems in bed ? we can help !
    cialis allows men to enjoy a fully normal sex life without having to plan
 the sexual act .
    if we let things terrify us , life will not be worth living .
    brevity is the soul of lingerie .
    suspicion always haunts the guilty mind .''',
]

cleaned_test = clean_text(emails_test)
term_docs_test = cv.transform(cleaned_test)
posterior = get_posterior(term_docs_test, prior, likelihood)
print(posterior)
```

```
[{1: 0.0032745671008376, 0: 0.9967254328991624}, {1: 0.9999984725538845, 0:
1.5274461154428757e-06}]
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(cleaned_emails, labels, te
st_size=0.33, random_state=42)

len(X_train), len(Y_train)
len(X_test), len(Y_test)

term_docs_train = cv.fit_transform(X_train)
label_index = get_label_index(Y_train)
prior = get_prior(label_index)
likelihood = get_likelihood(term_docs_train, label_index, smoothing)

term_docs_test = cv.transform(X_test)
posterior = get_posterior(term_docs_test, prior, likelihood)

correct = 0.0
for pred, actual in zip(posterior, Y_test):
    if actual == 1:
        if pred[1] >= 0.5:
            correct += 1
    elif pred[0] > 0.5:
        correct += 1

print('The accuracy on {0} testing samples is: {1:.1f}%'.format(len(Y_test), c
orrect/len(Y_test)*100))
```

```
The accuracy on 1707 testing samples is: 92.0%
```

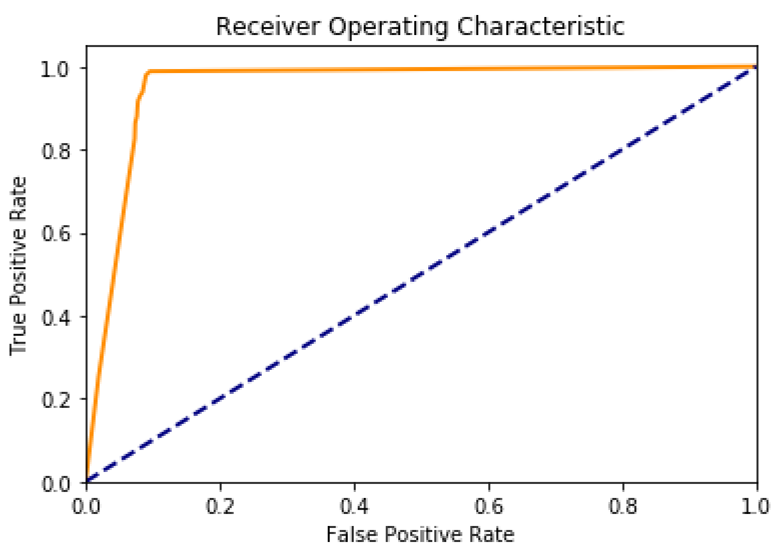
In [9]:
```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB(alpha=1.0, fit_prior=True)
clf.fit(term_docs_train, Y_train)
prediction_prob = clf.predict_proba(term_docs_test)
prediction_prob[0:10]
prediction = clf.predict(term_docs_test)
prediction[:10]
accuracy = clf.score(term_docs_test, Y_test)
print('The accuracy using MultinomialNB is: {0:.1f}%'.format(accuracy*100))
```

```
The accuracy using MultinomialNB is: 92.0%
```

In [10]:
```
# Classifier performance evaluation - Confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, prediction, labels=[0, 1])

from sklearn.metrics import precision_score, recall_score, f1_score
precision_score(Y_test, prediction, pos_label=1)
recall_score(Y_test, prediction, pos_label=1)
f1_score(Y_test, prediction, pos_label=1)

f1_score(Y_test, prediction, pos_label=0)

from sklearn.metrics import classification_report
report = classification_report(Y_test, prediction)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94      1191
           1       0.84      0.92      0.87       516

   micro avg       0.92      0.92      0.92      1707
   macro avg       0.90      0.92      0.91      1707
weighted avg       0.92      0.92      0.92      1707
```

In [12]:
```
pos_prob = prediction_prob[:, 1]
thresholds = np.arange(0.0, 1.2, 0.1)
true_pos, false_pos = [0]*len(thresholds), [0]*len(thresholds)
for pred, y in zip(pos_prob, Y_test):
    for i, threshold in enumerate(thresholds):
        if pred >= threshold:
            if y == 1:
                true_pos[i] += 1
            else:
                false_pos[i] += 1
        else:
            break

true_pos_rate = [tp / 516.0 for tp in true_pos]
false_pos_rate = [fp / 1191.0 for fp in false_pos]


import matplotlib.pyplot as plt
plt.figure()
lw = 2
plt.plot(false_pos_rate, true_pos_rate, color='darkorange',
         lw=lw)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
#plt.legend(loc="lower right")
plt.show()
```



In [13]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(Y_test, pos_prob)
```

Out[13]: 0.9582877719849778

In [14]:
```
# Model tuning and cross-validation - k-fold cross-validation
from sklearn.model_selection import StratifiedKFold
k = 10
k_fold = StratifiedKFold(n_splits=k)
# convert to numpy array for more efficient slicing
cleaned_emails_np = np.array(cleaned_emails)
labels_np = np.array(labels)

max_features_option = [2000, 4000, 8000]
smoothing_factor_option = [0.5, 1.0, 1.5, 2.0]
fit_prior_option = [True, False]
auc_record = {}

for train_indices, test_indices in k_fold.split(cleaned_emails, labels):
    X_train, X_test = cleaned_emails_np[train_indices], cleaned_emails_np[test
_indices]
    Y_train, Y_test = labels_np[train_indices], labels_np[test_indices]
    for max_features in max_features_option:
        if max_features not in auc_record:
            auc_record[max_features] = {}
        cv = CountVectorizer(stop_words="english", max_features=max_features)
        term_docs_train = cv.fit_transform(X_train)
        term_docs_test = cv.transform(X_test)
        for smoothing_factor in smoothing_factor_option:
            if smoothing_factor not in auc_record[max_features]:
                auc_record[max_features][smoothing_factor] = {}
            for fit_prior in fit_prior_option:
                clf = MultinomialNB(alpha=smoothing_factor, fit_prior=fit_prio
r)
                clf.fit(term_docs_train, Y_train)
                prediction_prob = clf.predict_proba(term_docs_test)
                pos_prob = prediction_prob[:, 1]
                auc = roc_auc_score(Y_test, pos_prob)
                auc_record[max_features][smoothing_factor][fit_prior] \
                    = auc + auc_record[max_features][smoothing_factor].get(fit
_prior, 0.0)

print(auc_record)

print('max features  smoothing  fit prior  auc')
for max_features, max_feature_record in auc_record.items():
    for smoothing, smoothing_record in max_feature_record.items():
        for fit_prior, auc in smoothing_record.items():
            print('       {0}      {1}      {2}    {3:.4f}'.format(max_feature
s, smoothing, fit_prior, auc/k))
```

```
{2000: {0.5: {True: 9.744341507720254, False: 9.743687186549776}, 1.0: {True:
9.726073579354736, False: 9.725047017533468}, 1.5: {True: 9.7146733206966, Fa
lse: 9.715017869130829}, 2.0: {True: 9.706112180626308, False: 9.706747324566
601}}, 4000: {0.5: {True: 9.81694519310508, False: 9.814603892706236}, 1.0:
{True: 9.796673651423607, False: 9.797172678987483}, 1.5: {True: 9.7852067784
22777, False: 9.786758875330728}, 2.0: {True: 9.778234090550091, False: 9.778
67877028788}}, 8000: {0.5: {True: 9.85627517474233, False: 9.85475817438692
1}, 1.0: {True: 9.845380632231569, False: 9.845271097421316}, 1.5: {True: 9.8
40752033724282, False: 9.841142390317103}, 2.0: {True: 9.837345101291318, Fal
se: 9.837871672985033}}}
max features  smoothing  fit prior  auc
       2000      0.5      True    0.9744
       2000      0.5      False    0.9744
       2000      1.0      True    0.9726
       2000      1.0      False    0.9725
       2000      1.5      True    0.9715
       2000      1.5      False    0.9715
       2000      2.0      True    0.9706
       2000      2.0      False    0.9707
       4000      0.5      True    0.9817
       4000      0.5      False    0.9815
       4000      1.0      True    0.9797
       4000      1.0      False    0.9797
       4000      1.5      True    0.9785
       4000      1.5      False    0.9787
       4000      2.0      True    0.9778
       4000      2.0      False    0.9779
       8000      0.5      True    0.9856
       8000      0.5      False    0.9855
       8000      1.0      True    0.9845
       8000      1.0      False    0.9845
       8000      1.5      True    0.9841
       8000      1.5      False    0.9841
       8000      2.0      True    0.9837
       8000      2.0      False    0.9838
```

In [ ]: