In [1]:

```
import demjson
import math
import matplotlib.pyplot as plt
import numpy
import json
import random
import scipy
import sklearn
import string
import tensorflow as tf
from collections import defaultdict # Dictionaries that take a default for missi
from sklearn import linear_model
```

```
2021-12-07 23:22:03.818622: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcuda
rt.so.11.0: cannot open shared object file: No such file or directory
2021-12-07 23:22:03.818648: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
Ignore above cudart dlerror if you do not have a GPU set up on your machine.
```

Data is available at http://cseweb.ucsd.edu/~jmcauley/pml/data/. Download and save to your own directory

In [2]:

```
#dataDir = "/home/jmcauley/pml_data/"
dataDir = "/home/julian/backup/jmcauley/pml_data/"
```

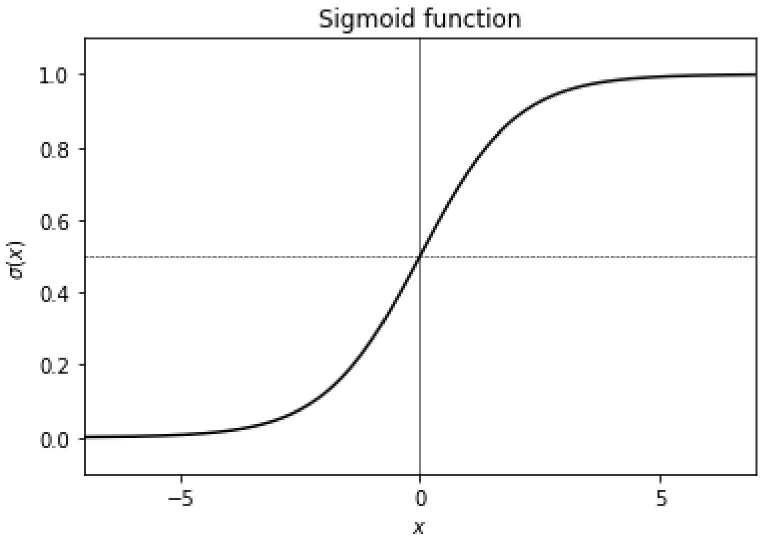
Sigmoid function

In [3]:

```
def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))
```

In [4]:

```
X = numpy.arange(-7,7.1,0.1)
Y = [sigmoid(x) for x in X]
plt.plot(X, Y, color='black')
plt.plot([0,0],[-2,2], color = 'black', linewidth=0.5)
plt.plot([-7,7],[0.5,0.5], color = 'k', linewidth=0.5, linestyle='--')
plt.xlim(-7, 7)
plt.ylim(-0.1, 1.1)
plt.xticks([-5,0,5])
plt.xlabel("$x$")
plt.ylabel(r"$\sigma(x)$")
plt.title("Sigmoid function")
plt.show()
```



Implementing a simple classifier

Read a small dataset of beer reviews

In [5]:

```
path = dataDir + "beer_50000.json"
f = open(path)

data = []

for l in f:
    if 'user/gender' in l: # Discard users who didn't specify gender
        d = eval(l) # demjson is more secure but much slower!
        data.append(d)

f.close()
```

In [6]:

```
data[0]
```

Out[6]:

```
{'review/appearance': 4.0,
 'beer/style': 'American Double / Imperial IPA',
 'review/palate': 4.0,
 'review/taste': 4.5,
 'beer/name': 'Cauldron DIPA',
 'review/timeUnix': 1293735206,
 'user/gender': 'Male',
 'user/birthdayRaw': 'Jun 16, 1901',
 'beer/ABV': 7.7,
 'beer/beerId': '64883',
 'user/birthdayUnix': -2163081600,
 'beer/brewerId': '1075',
 'review/timeStruct': {'isdst': 0,
  'mday': 30,
  'hour': 18,
  'min': 53,
  'sec': 26,
  'mon': 12,
  'year': 2010,
  'yday': 364,
```

```
  'wday': 3},
 'user/ageInSeconds': 3581417047,
 'review/overall': 4.0,
 'review/text': "According to the website, the style for the Caldera Cauldron ch
anges every year. The current release is a DIPA, which frankly is the only cauld
ron I'm familiar with (it was an IPA/DIPA the last time I ordered a cauldron at
the horsebrass several years back). In any event... at the Horse Brass yesterda
y.\t\tThe beer pours an orange copper color with good head retention and lacing.
The nose is all hoppy IPA goodness, showcasing a huge aroma of dry citrus, pine
and sandlewood. The flavor profile replicates the nose pretty closely in this We
st Coast all the way DIPA. This DIPA is not for the faint of heart and is a bit
much even for a hophead like myslf. The finish is quite dry and hoppy, and ther
e's barely enough sweet malt to balance and hold up the avalanche of hoppy bitte
rness in this beer. Mouthfeel is actually fairly light, with a long, persistente
ly bitter finish. Drinkability is good, with the alcohol barely noticeable in th
is well crafted beer. Still, this beer is so hugely hoppy/bitter, it's really ha
rd for me to imagine ordering more than a single glass. Regardless, this is a ve
ry impressive beer from the folks at Caldera.",
 'user/profileName': 'johnmichaelsen',
 'review/aroma': 4.5}
```


Predict the user's gender from the length of their review

In [7]:
```
X = [[1, len(d['review/text'])] for d in data]
y = [d['user/gender'] == 'Female' for d in data]
```

Fit the model

In [8]:
```
mod = sklearn.linear_model.LogisticRegression(fit_intercept=False)
mod.fit(X,y)
```

Out[8]:
```
LogisticRegression()
```

Calculate the accuracy of the model

In [9]:
```
predictions = mod.predict(X) # Binary vector of predictions
correct = predictions == y # Binary vector indicating which predictions were cor
sum(correct) / len(correct)
```

Out[9]:
```
0.9849041807577317
```

Accuracy seems surprisingly high! Check against the number of positive labels...

In [10]:
```
1 - (sum(y) / len(y))
```

Out[10]:
```
0.9849041807577317
```

Accuracy is identical to the proportion of "males" in the data. Confirm that the model is never predicting positive

In [11]:
```
sum(predictions)
```

Out[11]:
```
0
```

Implementing a balanced classifier

Use the class_weight='balanced' option to implement the balanced classifier

In [12]:
```
mod = sklearn.linear_model.LogisticRegression(class_weight='balanced')
mod.fit(X,y)
predictions = mod.predict(X)
```

Simple classification diagnostics

Accuracy

Compute the accuracy of the balanced model

In [13]:
```
correct = predictions == y
sum(correct) / len(correct)
```

Out[13]:
```
0.4225849139832378
```

True positives, False positives (etc.), and balanced error rate (BER)

In [14]:
```
TP = sum([(p and l) for (p,l) in zip(predictions, y)])
FP = sum([(p and not l) for (p,l) in zip(predictions, y)])
TN = sum([(not p and not l) for (p,l) in zip(predictions, y)])
FN = sum([(not p and l) for (p,l) in zip(predictions, y)])
```

In [15]:
```
print("TP = " + str(TP))
print("FP = " + str(FP))
print("TN = " + str(TN))
print("FN = " + str(FN))
```

```
TP = 199
FP = 11672
TN = 8423
FN = 109
```

Can rewrite the accuracy in terms of these metrics

In [16]:
```
(TP + TN) / (TP + FP + TN + FN)
```

Out[16]:
```
0.4225849139832378
```

True positive and true negative rates

In [17]:
```
TPR = TP / (TP + FN)
TNR = TN / (TN + FP)
```

In [18]:
```
TPR,TNR
```

Out[18]:
```
(0.6461038961038961, 0.4191589947748196)
```

Balanced error rate (BER)

In [19]:
```
BER = 1 - 1/2 * (TPR + TNR)
BER
```

Out[19]:
```
0.4673685545606422
```

Precision, recall, and F1 scores

In [20]:
```
precision = TP / (TP + FP)
recall = TP / (TP + FN)
```

In [21]:
```
precision, recall
```

Out[21]:
```
(0.0167635414034201, 0.6461038961038961)
```

F1 score

In [22]:
```
F1 = 2 * (precision*recall) / (precision + recall)
F1
```

Out[22]:
```
0.032679201904918305
```

Significance testing

In [23]:
```
path = dataDir + "beer_500.json"
f = open(path)

data = []

for l in f:
    d = eval(l)
    data.append(d)

f.close()
```

Randomly sort the data (so that train and test are iid)

In [24]:
```
random.seed(0)
random.shuffle(data)
```

Predict overall rating from ABV

In [25]:
```
X1 = [[1] for d in data] # Model *without* the feature
X2 = [[1, d['beer/ABV']] for d in data] # Model *with* the feature
y = [d['review/overall'] for d in data]
```

Fit the two models (with and without the feature)

In [26]:
```
model1 = sklearn.linear_model.LinearRegression(fit_intercept=False)
model1.fit(X1[:250], y[:250]) # Train on first half
residuals1 = model1.predict(X1[250:]) - y[250:] # Test on second half
```

In [27]:
```
model2 = sklearn.linear_model.LinearRegression(fit_intercept=False)
model2.fit(X2[:250], y[:250])
residuals2 = model2.predict(X2[250:]) - y[250:]
```

Residual sum of squares for both models

In [28]:
```
rss1 = sum([r**2 for r in residuals1])
rss2 = sum([r**2 for r in residuals2])
k1,k2 = 1,2 # Number of parameters of each model
n = len(residuals1) # Number of samples
```

F statistic (results may vary for different random splits)

In [29]:
```
F = ((rss1 - rss2) / (k2 - k1)) / (rss2 / (n-k2))
1 - scipy.stats.f.cdf(F,k2-k1,n-k2)
```

Out[29]:
```
1.0
```

Regression in tensorflow

Small dataset of fantasy reviews

In [30]:
```
path = dataDir + "fantasy_100.json"
f = open(path)

data = []

for l in f:
    d = json.loads(l)
    data.append(d)

f.close()
```

Predict rating from review length

In [31]:
```
ratings = [d['rating'] for d in data]
lengths = [len(d['review_text']) for d in data]
```

In [32]:
```
X = numpy.matrix([[1,l] for l in lengths])
y = numpy.matrix(ratings).T
```

First check the coefficients if we fit the model using sklearn

In [33]:
```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, y)
theta = model.coef_
theta
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWa
rning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. P
lease convert to a numpy array with np.asarray. For more information see: http
s://numpy.org/doc/stable/reference/generated/numpy.matrix.html
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWa
rning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. P
lease convert to a numpy array with np.asarray. For more information see: http
s://numpy.org/doc/stable/reference/generated/numpy.matrix.html
  FutureWarning,
```

Out[33]:
```
array([[3.98394783e+00, 1.19363599e-04]])
```

Convert features and labels to tensorflow structures

In [34]:
```
X = tf.constant(X, dtype=tf.float32)
y = tf.constant(y, dtype=tf.float32)
```

```
2021-12-07 23:22:08.609713: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcuda
rt.so.11.0: cannot open shared object file: No such file or directory
2021-12-07 23:22:08.609805: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcublas.so.11'; dlerror: libcubla
s.so.11: cannot open shared object file: No such file or directory
2021-12-07 23:22:08.609867: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcublasLt.so.11'; dlerror: libcubl
asLt.so.11: cannot open shared object file: No such file or directory
2021-12-07 23:22:08.609927: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcufft.so.10'; dlerror: libcufft.s
o.10: cannot open shared object file: No such file or directory
2021-12-07 23:22:08.609988: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcurand.so.10'; dlerror: libcuran
d.so.10: cannot open shared object file: No such file or directory
2021-12-07 23:22:08.610046: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcusolver.so.11'; dlerror: libcuso
lver.so.11: cannot open shared object file: No such file or directory
2021-12-07 23:22:08.610105: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcusparse.so.11'; dlerror: libcusp
arse.so.11: cannot open shared object file: No such file or directory
2021-12-07 23:22:08.610163: W tensorflow/stream_executor/platform/default/dso_lo
ader.cc:64] Could not load dynamic library 'libcudnn.so.8'; dlerror: libcudnn.s
o.8: cannot open shared object file: No such file or directory
2021-12-07 23:22:08.610176: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1
850] Cannot dlopen some GPU libraries. Please make sure the missing libraries me
ntioned above are installed properly if you would like to use GPU. Follow the gu
ide at https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices...
```

Build tensorflow regression class

In [35]:
```
class regressionModel(tf.keras.Model):
    def __init__(self, M, lamb):
        super(regressionModel, self).__init__()
```

```
        # Initialize weights to zero
        self.theta = tf.Variable(tf.constant([0.0]*M, shape=[M,1], dtype=tf.floa
        self.lamb = lamb

    # Prediction (for a matrix of instances)
    def predict(self, X):
        return tf.matmul(X, self.theta)

    # Mean Squared Error
    def MSE(self, X, y):
        return tf.reduce_mean((tf.matmul(X, self.theta) - y)**2)

    # Regularizer
    def reg(self):
        return self.lamb * tf.reduce_sum(self.theta**2)

    # L1 regularizer
    def reg1(self):
        return self.lamb * tf.reduce_sum(tf.abs(self.theta))

    # Loss
    def call(self, X, y):
        return self.MSE(X, y) + self.reg()
```

Initialize the model (lambda = 0)

In [36]:

```
optimizer = tf.keras.optimizers.Adam(0.1)
model = regressionModel(len(X[0]), 0)
```

Train for 1000 iterations of gradient descent (could implement more careful stopping criteria)

In [37]:

```
for iteration in range(1000):
    with tf.GradientTape() as tape:
        loss = model(X, y)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

Confirm that we get a similar result to what we got using sklearn

In [38]:

```
model.theta
```

Out[38]:

```
<tf.Variable 'Variable:0' shape=(2, 1) dtype=float32, numpy=
array([[3.9834061e+00],
       [1.1961416e-04]], dtype=float32)>
```

Make a few predictions using the model

In [39]:

```
model.predict(X)[:10]
```

Out[39]:

```
<tf.Tensor: shape=(10, 1), dtype=float32, numpy=
array([[4.232921 ],
       [4.165339 ],
       [4.1651   ],
       [4.197635 ],
       [4.194166 ],
       [4.0396247],
```

```
        [4.0818486],
        [4.047041 ],
        [4.0570884],
        [4.0489545]], dtype=float32)>
```

Classification in Tensorflow

Predict whether rating is above 4 from length

In [40]:

```
X = numpy.matrix([[1,l*0.0001] for l in lengths]) # Rescale the lengths for cond
y_class = numpy.matrix([[r > 4 for r in ratings]]).T
```

Convert to tensorflow structures

In [41]:

```
X = tf.constant(X, dtype=tf.float32)
y_class = tf.constant(y_class, dtype=tf.float32)
```

Tensorflow classification class

In [42]:

```
class classificationModel(tf.keras.Model):
    def __init__(self, M, lamb):
        super(classificationModel, self).__init__()
        self.theta = tf.Variable(tf.constant([0.0]*M, shape=[M,1], dtype=tf.floa
        self.lamb = lamb

    # Probability (for a matrix of instances)
    def predict(self, X):
        return tf.math.sigmoid(tf.matmul(X, self.theta))

    # Objective
    def obj(self, X, y):
        pred = self.predict(X)
        pos = y*tf.math.log(pred)
        neg = (1.0 - y)*tf.math.log(1.0 - pred)
        return -tf.reduce_mean(pos + neg)

    # Same objective, using tensorflow short-hand
    def obj_short(self, X, y):
        pred = self.predict(X)
        bce = tf.keras.losses.BinaryCrossentropy()
        return tf.reduce_mean(bce(y, pred))

    # Regularizer
    def reg(self):
        return self.lamb * tf.reduce_sum(self.theta**2)

    # Loss
    def call(self, X, y):
        return self.obj(X, y) + self.reg()
```

Initialize the model (lambda = 0)

In [43]:

```
optimizer = tf.keras.optimizers.Adam(0.1)
model = classificationModel(len(X[0]), 0)
```

Run for 1000 iterations

In [44]:
```
for iteration in range(1000):
    with tf.GradientTape() as tape:
        loss = model(X, y_class)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

In [45]:
```
model.theta
```

Out[45]:
```
<tf.Variable 'Variable:0' shape=(2, 1) dtype=float32, numpy=
array([[-0.08591048],
       [ 0.30873907]], dtype=float32)>
```

Model predictions (as probabilities via the sigmoid function)

In [46]:
```
model.predict(X)[:10]
```

Out[46]:
```
<tf.Tensor: shape=(10, 1), dtype=float32, numpy=
array([[0.49462333],
       [0.4902634 ],
       [0.490248  ],
       [0.49234676],
       [0.49212298],
       [0.48215765],
       [0.4848793 ],
       [0.4826356 ],
       [0.4832832 ],
       [0.48275894]], dtype=float32)>
```

Regularization pipeline

In [47]:
```
def parseData(fname):
    for l in open(fname):
        yield eval(l)
```

Just read the first 5000 reviews (deliberately making a model that will overfit if not carefully regularized)

In [48]:
```
data = list(parseData(dataDir + "beer_50000.json"))[:5000]
```

Fit a simple bag-of-words model (see Chapter 8 for more details)

In [49]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data: # Strictly, should just use the *training* data to extract word c
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1

counts = [(wordCount[w], w) for w in wordCount]
```

```
counts.sort()
counts.reverse()
```

1000 most popular words

In [50]:
```
words = [x[1] for x in counts[:1000]]
```

In [51]:
```
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

Bag-of-words features for 1000 most popular words

In [52]:
```
def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation
    for w in r.split():
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) # offset
    return feat
```

In [53]:
```
random.shuffle(data)
```

In [54]:
```
X = [feature(d) for d in data]
y = [d['review/overall'] for d in data]
```

In [55]:
```
Ntrain,Nvalid,Ntest = 4000,500,500
Xtrain,Xvalid,Xtest = X[:Ntrain],X[Ntrain:Ntrain+Nvalid],X[Ntrain+Nvalid:]
ytrain,yvalid,ytest = y[:Ntrain],y[Ntrain:Ntrain+Nvalid],y[Ntrain+Nvalid:]
```

Unregularized model (train on training set, test on test set)

In [56]:
```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(Xtrain, ytrain)
predictions = model.predict(Xtest)
```

In [57]:
```
sum((ytest - predictions)**2)/len(ytest) # Mean squared error
```

Out[57]:
```
0.5621377319894529
```

Regularized model ("ridge regression")

In [58]:
```
model = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
model.fit(Xtrain, ytrain)
predictions = model.predict(Xtest)
```

In [59]:
```
sum((ytest - predictions)**2)/len(ytest)
```

Out[59]:
```
0.5553767774281313
```

Complete regularization pipeline

Track the model which works best on the validation set

In [60]:
```
bestModel = None
bestVal = None
bestLamb = None
```

Train models for different values of lambda (or C). Keep track of the best model on the validation set.

In [61]:
```
ls = [0.01, 0.1, 1, 10, 100, 1000, 10000]
errorTrain = []
errorValid = []

for l in ls:
    model = sklearn.linear_model.Ridge(l)
    model.fit(Xtrain, ytrain)
    predictTrain = model.predict(Xtrain)
    MSEtrain = sum((ytrain - predictTrain)**2)/len(ytrain)
    errorTrain.append(MSEtrain)
    predictValid = model.predict(Xvalid)
    MSEvalid = sum((yvalid - predictValid)**2)/len(yvalid)
    errorValid.append(MSEvalid)
    print("l = " + str(l) + ", validation MSE = " + str(MSEvalid))
    if bestVal == None or MSEvalid < bestVal:
        bestVal = MSEvalid
        bestModel = model
        bestLamb = l
```

```
l = 0.01, validation MSE = 0.4933728029983656
l = 0.1, validation MSE = 0.49292702478338096
l = 1, validation MSE = 0.48865630138060057
l = 10, validation MSE = 0.4585804821929264
l = 100, validation MSE = 0.39865226713206803
l = 1000, validation MSE = 0.413825760476879
l = 10000, validation MSE = 0.5041631912430448
```

Using the best model from the validation set, compute the error on the test set

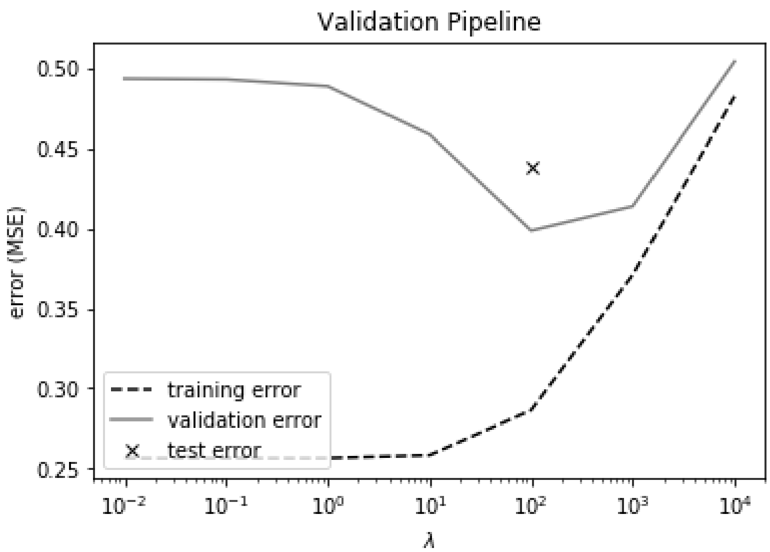
In [62]:
```
predictTest = bestModel.predict(Xtest)
MSEtest = sum((ytest - predictTest)**2)/len(ytest)
MSEtest
```

Out[62]:
```
0.4380927014652703
```

Plot the train/validation/test error associated with this pipeline

In [63]:
```
plt.xticks([])
plt.xlabel(r"$\lambda$")
plt.ylabel(r"error (MSE)")
plt.title(r"Validation Pipeline")
plt.xscale('log')
```

```
plt.plot(ls, errorTrain, color='k', linestyle='--', label='training error')
plt.plot(ls, errorValid, color='grey',zorder=4,label="validation error")
plt.plot([bestLamb], [MSEtest], linestyle='', marker='x', color='k', label="test
plt.legend(loc='lower left')
plt.show()
```



Precision, recall, and ROC curves

Same data as pipeline above, slightly bigger dataset

In [64]:
```
data = list(parseData(dataDir + "beer_50000.json"))[:10000]
```

Simple bag-of-words model (as in pipeline above, and in Chapter 8)

In [65]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1
```

In [66]:
```
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()
```

In [67]:
```
words = [x[1] for x in counts[:1000]]
```

In [68]:
```
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

In [69]:
```
def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation
```

```
    ws = r.split()
    for w in ws:
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) # offset
    return feat
```

Predict whether the ABV is above 6.7 (roughly, above average) from the review text

In [70]:
```
random.shuffle(data)
```

In [71]:
```
X = [feature(d) for d in data]
y = [d['beer/ABV'] > 6.7 for d in data]
```

Train on first 9000 reviews, test on last 1000

In [72]:
```
mod = sklearn.linear_model.LogisticRegression(max_iter=5000)
mod.fit(X[:9000],y[:9000])
predictions = mod.predict(X[9000:]) # Binary vector of predictions
correct = predictions == y[9000:]
```

Accuracy

In [73]:
```
sum(correct) / len(correct)
```

Out[73]:
```
0.832
```

To compute precision and recall, we want the output probabilities (or scores) rather than the predicted labels

In [74]:
```
probs = mod.predict_proba(X[9000:]) # could also use mod.decision_function
```

Build a simple data structure that contains the score, the predicted label, and the actual label (on the test set)

In [75]:
```
probY = list(zip([p[1] for p in probs], [p[1] > 0.5 for p in probs], y[9000:]))
```

For example...

In [76]:
```
probY[:10]
```

Out[76]:
```
[(0.9999998762354996, True, True),
 (0.012720481291453863, False, False),
 (0.997069530448619, True, True),
 (0.00011239818179373418, False, False),
 (0.9506213023816044, True, True),
 (0.8511369245118185, True, False),
 (0.9508087266170231, True, True),
 (0.0019793193061717392, False, True),
 (0.903593342768345, True, True),
 (0.08480983386485495, False, True)]
```

Sort this so that the most confident predictions come first

In [77]:
```
probY.sort(reverse=True)
```

For example...

In [78]:
```
probY[:10]
```

Out[78]:
```
[(1.0, True, True),
 (0.9999999999999885, True, True),
 (0.9999999999999716, True, True),
 (0.9999999999999378, True, True),
 (0.9999999999998981, True, True),
 (0.9999999999980203, True, True),
 (0.999999999996775, True, True),
 (0.9999999999825748, True, True),
 (0.9999999997872742, True, True),
 (0.9999999997680737, True, True)]
```

Receiver operator characteristic (ROC) curve

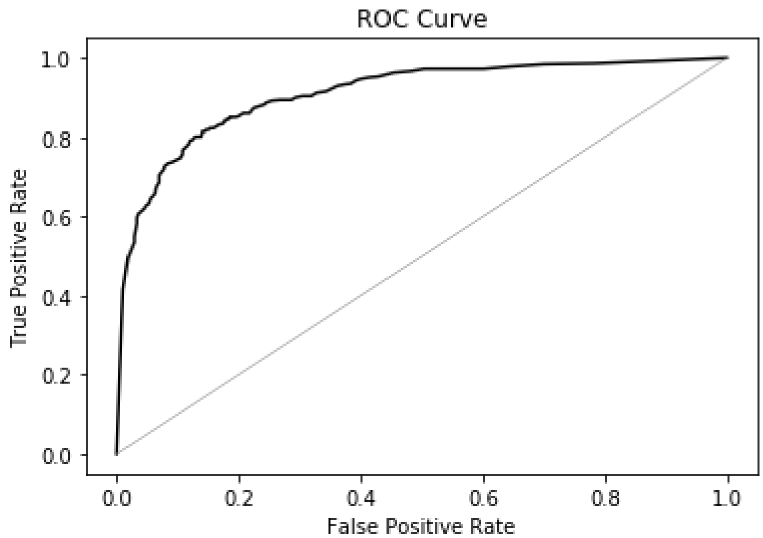
In [79]:
```
xROC = []
yROC = []

for thresh in numpy.arange(0,1.01,0.01):
    preds = [x[0] > thresh for x in probY]
    labs = [x[2] for x in probY]
    if len(labs) == 0:
        continue

    TP = sum([(a and b) for (a,b) in zip(preds,labs)])
    FP = sum([(a and not b) for (a,b) in zip(preds,labs)])
    TN = sum([(not a and not b) for (a,b) in zip(preds,labs)])
    FN = sum([(not a and b) for (a,b) in zip(preds,labs)])

    TPR = TP / (TP + FN) # True positive rate
    FPR = FP / (TN + FP) # False positive rate
    xROC.append(FPR)
    yROC.append(TPR)
```

In [80]:
```
plt.plot(xROC,yROC,color='k')
plt.plot([0,1],[0,1], lw=0.5, color='grey')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```



Precision recall curve

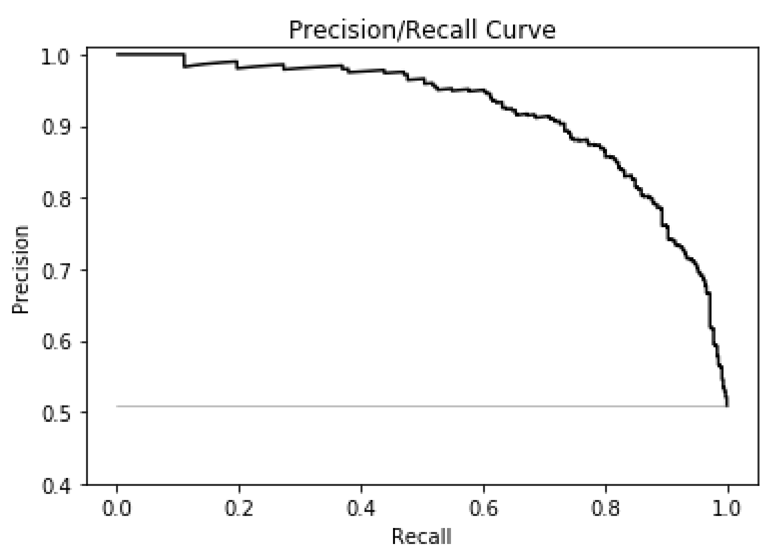
In [81]:

```
xPR = []
yPR = []

for i in range(1,len(probY)+1):
    preds = [x[1] for x in probY[:i]]
    labs = [x[2] for x in probY[:i]]
    prec = sum(labs) / len(labs)
    rec = sum(labs) / sum(y[9000:])
    xPR.append(rec)
    yPR.append(prec)
```

In [82]:

```
plt.plot(xPR,yPR,color='k')
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.ylim(0.4,1.01)
plt.plot([0,1],[sum(y[9000:]) / 1000, sum(y[9000:]) / 1000], lw=0.5, color = 'gr
plt.title("Precision/Recall Curve")
plt.show()
```



In [ ]:

Exercises

3.1

In [83]:

```
path = dataDir + "beer_50000.json"
f = open(path)
data = []
for l in f:
    data.append(eval(l))
f.close()
```

Count occurrences of each style

In [84]:

```
categoryCounts = defaultdict(int)
for d in data:
    categoryCounts[d['beer/style']] += 1

categories = [c for c in categoryCounts if categoryCounts[c] > 1000]

catID = dict(zip(list(categories),range(len(categories))))
```

Build one-hot encoding using common styles

In [85]:

```
def feat(d):
    feat = [0] * len(catID)
    if d['beer/style'] in catID:
        feat[catID[d['beer/style']]] = 1
    return feat + [1]
```

In [86]:

```
X = [feat(d) for d in data]
y = [d['beer/ABV'] > 5 for d in data]
```

In [87]:

```
mod = sklearn.linear_model.LogisticRegression()
mod.fit(X,y)
```

Out[87]:

```
LogisticRegression()
```

Compute and report metrics

In [88]:

```
def metrics(y, ypred):
    TP = sum([(a and b) for (a,b) in zip(y, ypred)])
    TN = sum([(not a and not b) for (a,b) in zip(y, ypred)])
    FP = sum([(not a and b) for (a,b) in zip(y, ypred)])
    FN = sum([(a and not b) for (a,b) in zip(y, ypred)])

    TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)
```

```
    BER = 1 - 0.5*(TPR + TNR)

    print("TPR = " + str(TPR))
    print("TNR = " + str(TNR))
    print("BER = " + str(BER))
```

In [89]:

```
ypred = mod.predict(X)
metrics(y, ypred)
```

```
TPR = 0.9943454210202828
TNR = 0.27828418230563
BER = 0.3636851983370436
```

3.2

Balance the classifier using the 'balanced' option

In [90]:

```
mod = sklearn.linear_model.LogisticRegression(class_weight='balanced')
mod.fit(X,y)
```

Out[90]:

```
LogisticRegression(class_weight='balanced')
```

In [91]:

```
ypred = mod.predict(X)
metrics(y, ypred)
```

```
TPR = 0.6779348494161033
TNR = 0.9751206434316354
BER = 0.1734722535761306
```

3.3

Precision/recall curves

In [92]:

```
probs = mod.predict_proba(X)
```

In [93]:

```
probY = list(zip([p[1] for p in probs], [p[1] > 0.5 for p in probs], y))
```

In [94]:

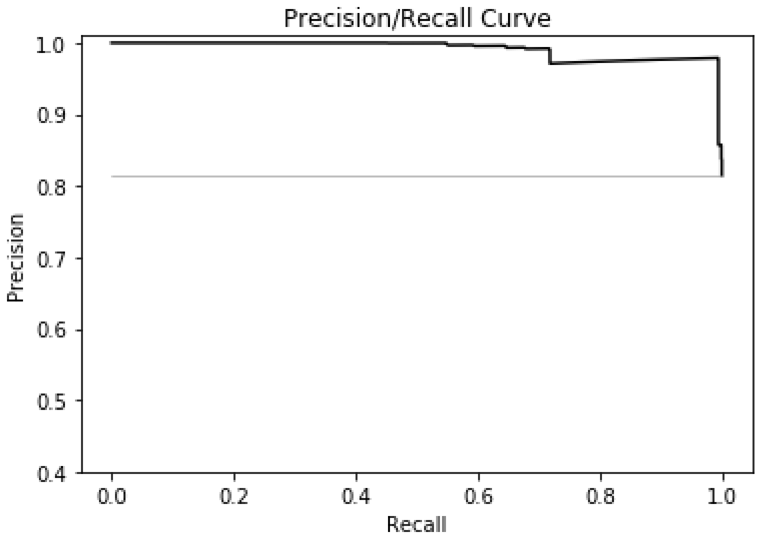
```
probY.sort(reverse=True) # Sort data by confidence
```

In [95]:

```
xPR = []
yPR = []

for i in range(1,len(probY)+1,100):
    preds = [x[1] for x in probY[:i]]
    labs = [x[2] for x in probY[:i]]
    prec = sum(labs) / len(labs)
    rec = sum(labs) / sum(y)
    xPR.append(rec)
    yPR.append(prec)
```

In [96]:
```
plt.plot(xPR,yPR,color='k')
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.ylim(0.4,1.01)
plt.plot([0,1],[sum(y) / len(y), sum(y) / len(y)], lw=0.5, color = 'grey')
plt.title("Precision/Recall Curve")
plt.show()
```



3.4

Model pipeline

In [97]:
```
dataTrain = data[:25000]
dataValid = data[25000:37500]
dataTest = data[37500:]
```

In [98]:
```
def pipeline(reg):
    mod = linear_model.LogisticRegression(C=reg, class_weight='balanced')

    X = [feat(d) for d in dataTrain]
    y = [d['beer/ABV'] > 5 for d in dataTrain]

    Xvalid = [feat(d) for d in dataValid]
    yvalid = [d['beer/ABV'] > 5 for d in dataValid]
    Xtest = [feat(d) for d in dataTest]
    ytest = [d['beer/ABV'] > 5 for d in dataTest]

    mod.fit(X,y)
    ypredValid = mod.predict(Xvalid)
    ypredTest = mod.predict(Xtest)

    # validation

    TP = sum([(a and b) for (a,b) in zip(yvalid, ypredValid)])
    TN = sum([(not a and not b) for (a,b) in zip(yvalid, ypredValid)])
    FP = sum([(not a and b) for (a,b) in zip(yvalid, ypredValid)])
    FN = sum([(a and not b) for (a,b) in zip(yvalid, ypredValid)])

    TPR = TP / (TP + FN)
```

```
    TNR = TN / (TN + FP)

    BER = 1 - 0.5*(TPR + TNR)

    print("C = " + str(reg) + "; validation BER = " + str(BER))

    # test

    TP = sum([(a and b) for (a,b) in zip(ytest, ypredTest)])
    TN = sum([(not a and not b) for (a,b) in zip(ytest, ypredTest)])
    FP = sum([(not a and b) for (a,b) in zip(ytest, ypredTest)])
    FN = sum([(a and not b) for (a,b) in zip(ytest, ypredTest)])

    TPR = TP / (TP + FN)
    TNR = TN / (TN + FP)

    BER = 1 - 0.5*(TPR + TNR)

    print("C = " + str(reg) + "; test BER = " + str(BER))

    return mod
```

In [99]:

```
for c in [0.000001, 0.00001, 0.0001, 0.001]:
    pipeline(c)
```

```
C = 1e-06; validation BER = 0.16646546520651895
C = 1e-06; test BER = 0.264015029296767 9
C = 1e-05; validation BER = 0.28744502888678103
C = 1e-05; test BER = 0.39631747406856366
C = 0.0001; validation BER = 0.28744502888678103
C = 0.0001; test BER = 0.39631747406856366
C = 0.001; validation BER = 0.28744502888678103
C = 0.001; test BER = 0.39631747406856366
```

3.5

Fit the classification problem using a regular linear regressor

In [100…

```
y_reg = [2.0*a - 1 for a in y] # Map data to {-1.0,1.0}
```

In [101…

```
y_reg[:10]
```

Out[101…

```
[-1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0, -1.0, -1.0]
```

In [102…

```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, y_reg)
```

Out[102…

```
LinearRegression(fit_intercept=False)
```

In [103…

```
yreg_pred = model.predict(X)
yreg_pred = [a > 0 for a in yreg_pred] # Map the outputs back to binary predicti
```

In [104…

```
metrics(y, yreg_pred)
```

```
TPR = 0.9943454210202828
TNR = 0.27828418230563
BER = 0.3636851983370436
```

In [ ]:

```

```