In [1]:
```
import csv
import dateutil.parser # for parsing plain-text dates
import demjson # json (loose formatting)
import json
import math
import matplotlib.pyplot as plt
import numpy
import sklearn
from collections import defaultdict
from sklearn import linear_model
```

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/"
```

Regression (lines of best fit)

Read in a small dataset of reviews of fantasy novels (from Goodreads)

In [3]:
```
path = dataDir + "fantasy_100.json"
f = open(path)

data = []

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

f.close()
```

What does the data look like? (json format)

In [4]:
```
data[0]
```

Out[4]:
```
{'book_id': '18245960',
 'date_added': 'Sun Jul 30 07:44:10 -0700 2017',
 'date_updated': 'Wed Aug 30 00:00:26 -0700 2017',
 'n_comments': 1,
 'n_votes': 28,
 'rating': 5,
 'read_at': 'Sat Aug 26 12:05:52 -0700 2017',
 'review_id': 'dfdbb7b0eb5a7e4c26d59a937e2e5feb',
 'review_text': 'This is a special book. It started slow for about the first thi
rd, then in the middle third it started to get interesting, then the last third
blew my mind. This is what I love about good science fiction - it pushes your th
inking about where things can go. \n It is a 2015 Hugo winner, and translated fr
om its original Chinese, which made it interesting in just a different way from
most things I\'ve read. For instance the intermixing of Chinese revolutionary hi
story - how they kept accusing people of being "reactionaries", etc. \n It is a
book about science, and aliens. The science described in the book is impressive
- its a book grounded in physics and pretty accurate as far as I could tell. Tho
ugh when it got to folding protons into 8 dimensions I think he was just making
```

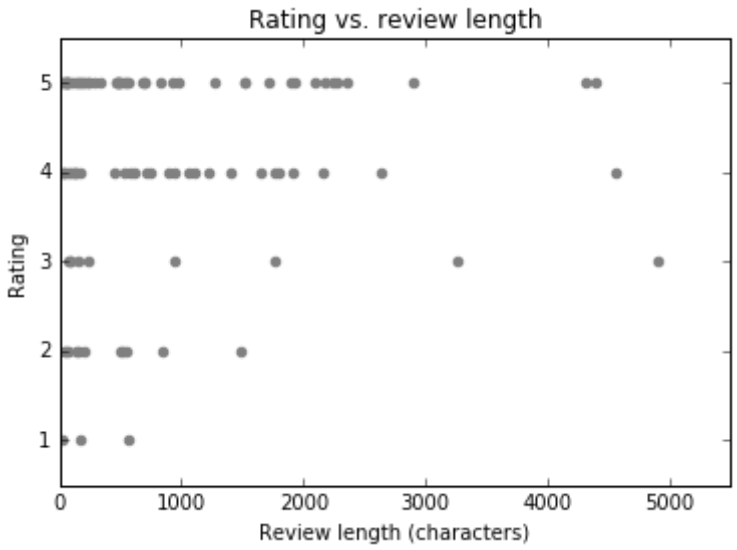
```
stuff up - interesting to think about though. \n But what would happen if our SE
TI stations received a message - if we found someone was out there - and the per
son monitoring and answering the signal on our side was disillusioned? That part
of the book was a bit dark - I would like to think human reaction to discovering
alien civilization that is hostile would be more like Enders Game where we would
band together. \n I did like how the book unveiled the Trisolaran culture throug
h the game. It was a smart way to build empathy with them and also understand wh
at they\'ve gone through across so many centuries. And who know a 3 body problem
was an unsolvable math problem? But I still don\'t get who made the game - maybe
that will come in the next book. \n I loved this quote: \n "In the long history
of scientific progress, how many protons have been smashed apart in accelerators
by physicists? How many neutrons and electrons? Probably no fewer than a hundred
million. Every collision was probably the end of the civilizations and intellige
nces in a microcosmos. In fact, even in nature, the destruction of universes mus
t be happening at every second--for example, through the decay of neutrons. Als
o, a high-energy cosmic ray entering the atmosphere may destroy thousands of suc
h miniature universes...."',
 'started_at': 'Tue Aug 15 13:23:18 -0700 2017',
 'user_id': '8842281e1d1347389f2ab93d60773d4d'}
```

To predict ratings from review length, let's start by assembling vectors of both

In [5]:
```
ratings = [d['rating'] for d in data]
lengths = [len(d['review_text']) for d in data]
```

And generating a quick scatter plot of the data to see overall trends...

In [6]:
```
plt.scatter(lengths, ratings, color='grey')
plt.xlim(0, 5500)
plt.ylim(0.5, 5.5)
plt.xlabel("Review length (characters)")
plt.ylabel("Rating")
plt.title("Rating vs. review length")
plt.show()
```



To perform regression, convert to features (X) and labels (y)

In [7]:
```
X = numpy.matrix([[1,l] for l in lengths]) # Note the inclusion of the constant
y = numpy.matrix(ratings).T
```

Fit a linear regression model to the data (sklearn)

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

Out[8]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=False)
```

Extract the model coefficients (theta)

In [9]:
```
theta = model.coef_
theta
```

Out[9]:
```
array([[3.98394783e+00, 1.19363599e-04]])
```

Same thing using numpy (lstsq)...

In [10]:
```
theta,residuals,rank,s = numpy.linalg.lstsq(X, y, rcond=None)
theta
```

Out[10]:
```
matrix([[3.98394783e+00],
        [1.19363599e-04]])
```

Same thing using by computing the pseudoinverse directly

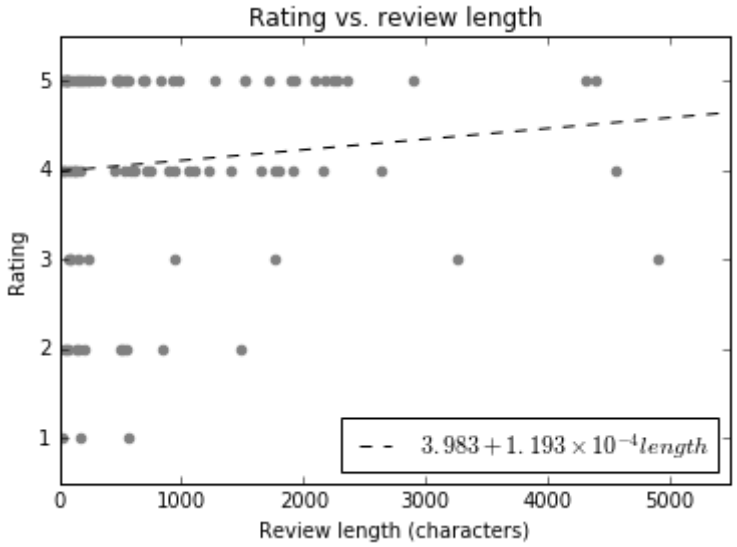
In [11]:
```
numpy.linalg.inv(X.T*X)*X.T*y
```

Out[11]:
```
matrix([[3.98394783e+00],
        [1.19363599e-04]])
```

Plot the line of best fit...

In [12]:
```
xplot = numpy.arange(0,5501,10)
yplot = [(theta[0] + theta[1]*x).item() for x in xplot]
```

In [13]:
```
plt.scatter(lengths, ratings, color='grey')
plt.plot(numpy.array(xplot), yplot, color = 'k', linestyle = '--',\
         label = r"$3.983 + 1.193 \times 10^{-4} \mathit{length}$")
plt.xlim(0, 5500)
plt.ylim(0.5, 5.5)
plt.xlabel("Review length (characters)")
plt.ylabel("Rating")
plt.title("Rating vs. review length")
plt.legend(loc='lower right')
plt.show()
```



Fit a slightly more complex model with two features (review length and number of comments)

In [14]:
```
def feature(d):
    feat = [1] # Constant feature
    feat.append(len(d['review_text'])) # Length of review
    feat.append(d['n_comments']) # Number of comments
    return feat
```

In [15]:
```
X = numpy.matrix([feature(d) for d in data])
```

Fit the model and extract the coefficients

In [16]:
```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, y)
theta = model.coef_
theta
```

Out[16]:
```
array([[3.95432090e+00, 7.24295694e-05, 1.08071028e-01]])
```

Mean Squared Error and R^2

Extract model predictions

In [17]:
```
y_pred = model.predict(X)
```

Sum of squared errors (SSE)

In [18]:
```
sse = sum([x**2 for x in (y - y_pred)])
```

Mean squared error (MSE)

In [19]:
```
mse = sse / len(y)
mse
```

Out[19]:
```
matrix([[1.22988869]])
```

R^2 and fraction of variance unexplained (FVU)

In [20]:
```
fvu = mse / numpy.var(y)
r2 = 1 - fvu
r2
```

Out[20]:
```
matrix([[0.04057361]])
```

Can also get the R^2 coefficient directly from the library...

In [21]:
```
model.score(X, y)
```

Out[21]:
```
0.040573608198668176
```

Simple feature transformations

Polynomial (quadratic) function

In [22]:
```
def feature(d):
    feat = [1]
    feat.append(len(d['review_text']))
    feat.append((len(d['review_text']))**2) # Quadratic term
    return feat
```

In [23]:
```
X = numpy.matrix([feature(d) for d in data])
```

In [24]:
```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, y)
theta = model.coef_
theta
```

Out[24]:
```
array([[ 3.86513512e+00,  4.34379750e-04, -8.37700479e-08]])
```

Compute the R^2 coefficient (using the library)

In [25]:
```
model.score(X, y)
```

Out[25]:
```
0.026559772903043233
```

Cubic function

In [26]:
```
def feature(d):
    feat = [1]
    feat.append(len(d['review_text'])/1000)
    feat.append((len(d['review_text'])/1000)**2) # Quadratic term
    feat.append((len(d['review_text'])/1000)**3) # Cubic term
    return feat
```

In [27]:
```
X = numpy.matrix([feature(d) for d in data])
```

In [28]:
```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, y)
theta = model.coef_
theta
```

Out[28]:
```
array([[ 3.92656525,  0.11531522,  0.14023331, -0.03494859]])
```

In [29]:
```
model.score(X, y)
```

Out[29]:
```
0.02926072167925875
```

Bar plot of daily ratings trends

Use a (slightly larger) dataset of fantasy reviews

In [30]:
```
path = dataDir + "fantasy_10000.json"
f = open(path)

data = []

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

f.close()
```

Extract averages for each day

In [31]:
```
weekAverages = defaultdict(list)

for d in data:
    timeString = d['date_added']
    t = dateutil.parser.parse(timeString)
    weekAverages[t.weekday()].append(d['rating'])

for k in weekAverages:
    weekAverages[k] = sum(weekAverages[k]) / len(weekAverages[k])
```

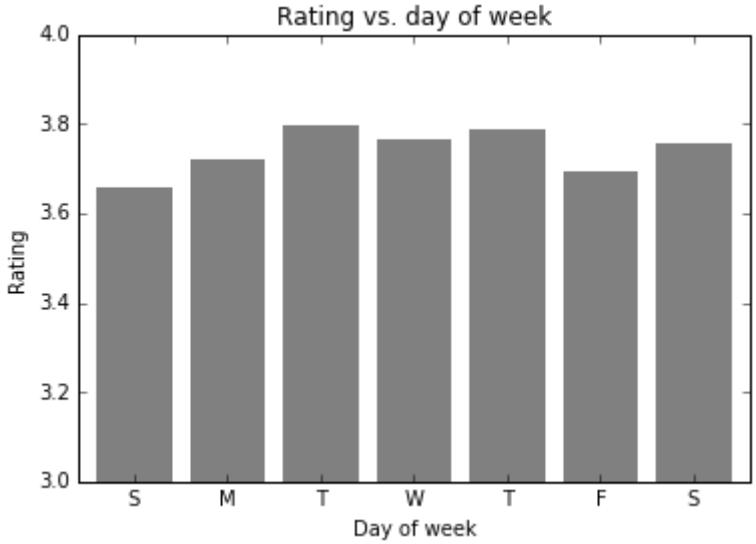
In [32]:
```
weekAverages
```

Out[32]:
```
defaultdict(list,
            {0: 3.6558265582655824,
             1: 3.7186505410566517,
             2: 3.795968234575443,
             3: 3.7673956262425445,
             4: 3.7895545314900154,
             5: 3.693290734824281,
             6: 3.755786113328013})
```

Plot the averages for each day

In [33]:
```
xplot = list(weekAverages.keys())
xplot.sort()
yplot = [weekAverages[k] for k in xplot]
```

In [34]:
```
plt.bar(xplot,yplot,color='grey',lw=0)
plt.xticks(numpy.arange(0.4,7.4,1), ["S", "M", "T", "W", "T", "F", "S"])
plt.xlim(-0.2,7.0)
plt.ylim(3,4)
plt.ylabel("Rating")
plt.xlabel("Day of week")
plt.title(r"Rating vs. day of week")
plt.show()
```



Binary (one-hot) features

Read a small dataset of beer reviews with gender attributes

In [35]:
```
path = dataDir + "beer_500.json"
f = open(path)

data = []

for l in f:
    d = demjson.decode(l) # Handles the looser json format in this file (fairly
    data.append(d)

f.close()
```

What does the data look like?

In [36]:
```
data[0]
```

Out[36]:
```
{'beer/ABV': 5.0,
 'beer/beerId': '47986',
```

```
 'beer/brewerId': '10325',
 'beer/name': 'Sausa Weizen',
 'beer/style': 'Hefeweizen',
 'review/appearance': 2.5,
 'review/aroma': 2.0,
 'review/overall': 1.5,
 'review/palate': 1.5,
 'review/taste': 1.5,
 'review/text': 'A lot of foam. But a lot.\tIn the smell some banana, and then l
actic and tart. Not a good start.\tQuite dark orange in color, with a lively car
bonation (now visible, under the foam).\tAgain tending to lactic sourness.\tSame
for the taste. With some yeast and banana.',
 'review/timeStruct': {'hour': 20,
  'isdst': 0,
  'mday': 16,
  'min': 57,
  'mon': 2,
  'sec': 3,
  'wday': 0,
  'yday': 47,
  'year': 2009},
 'review/timeUnix': 1234817823,
 'user/profileName': 'stcules'}
```

Filter the dataset to include only those users who specified gender

In [37]:
```
data = [d for d in data if 'user/gender' in d]
```

How many users have specified gender?

In [38]:
```
len(data)
```

Out[38]:
```
215
```

Binary representation of gender attribute

In [39]:
```
def feat(d):
    return [1, 1.0 * (d['user/gender'] == 'Female')]
```

In [40]:
```
X = [feat(d) for d in data]
y = [len(d['review/text'].split()) for d in data]
```

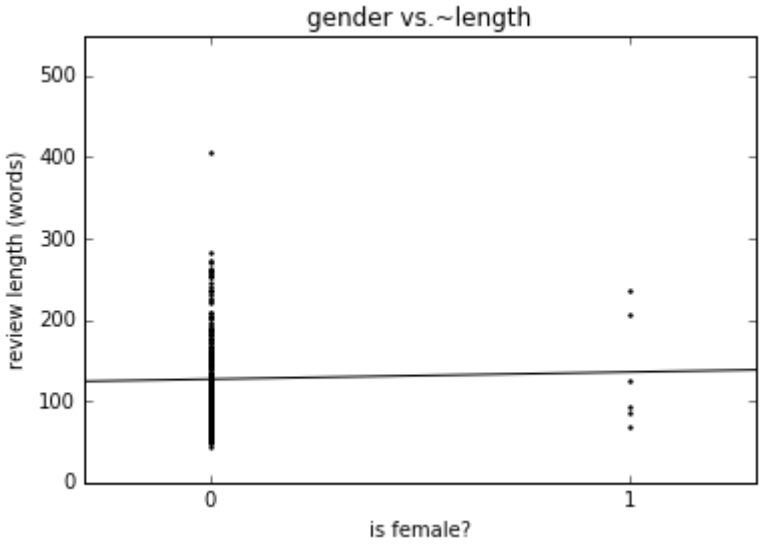
Fit a model to the binary feature

In [41]:
```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, y)
theta = model.coef_
theta
```

Out[41]:
```
array([127.07177033,   8.761563  ])
```

The model indicates that females tend to leave slightly longer reviews than males. Plot data from the two groups.

In [42]:
```
xplot = numpy.arange(-1,3,1)
yplot = [theta[0] + theta[1]*x for x in xplot]
plt.plot(xplot, yplot, color='k')
plt.scatter([[x[1] for x in X]], y, color='k', s=2)
plt.xlim(-0.3, 1.3)
plt.ylim(0, 550)
plt.xticks([0,1])
plt.xlabel("is female?")
plt.ylabel(r"review length (words)")
plt.title(r"gender vs.~length")
plt.show()
```



Transformation of output variables

Dataset of reddit submissions (and resubmissions)

In [43]:
```
f = open(dataDir + "redditSubmissions.csv")
cs = csv.reader(f)
header = next(cs)
header
```

Out[43]:
```
['#image_id',
 'unixtime',
 'rawtime',
 'title',
 'total_votes',
 'reddit_id',
 'number_of_upvotes',
 'subreddit',
 'number_of_downvotes',
 'localtime',
 'score',
 'number_of_comments',
 'username']
```

Compute popularity as a function of submission number (i.e., how many times has this identical image been submitted)

In [44]:

```
pops = defaultdict(list)
imId = None
count = 0
for l in cs:
    d = dict(zip(header, l))
    if d['#image_id'] != imId:
        count = 0
    imId = d['#image_id']
    count += 1
    try:
        pops[count].append(int(d['number_of_upvotes']))
    except Exception as e:
        continue

for x in pops:
    pops[x] = sum(pops[x]) / len(pops[x])
```

Extract a single feature which is just the submission number (0 for first submission, etc., 1 for first resubmission, etc.)

In [45]:

```
x = list(pops.keys())
x.sort()
X = [[1,a] for a in x]
```

The label to predict is the number of upvotes as a function of submission number

In [46]:

```
y = [pops[a] for a in x]
ylog = [math.log2(a) for a in y]
```

Fit two models: one with the original output variable, one with a log-transformed output variable

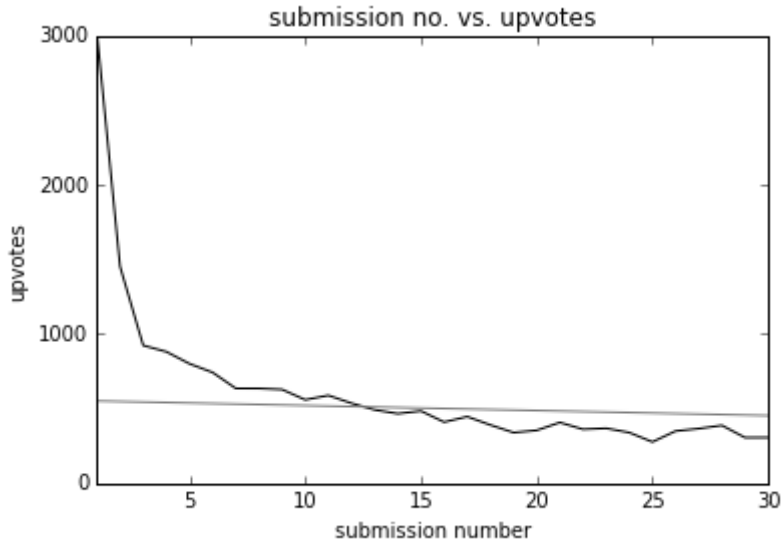
In [47]:

```
mod = linear_model.LinearRegression(fit_intercept = False)
modlog = linear_model.LinearRegression(fit_intercept = False)

mod.fit(X,y)
modlog.fit(X,ylog)

theta = mod.coef_
thetalog = modlog.coef_
```

Plot data and fitted values for the two models

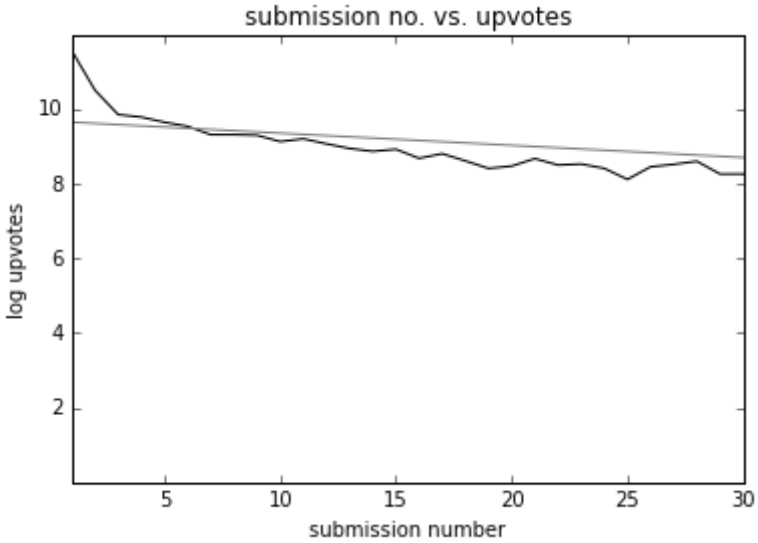
In [48]:

```
plt.plot(x,y,color='k')
plt.xlim(1,30)
plt.ylim(0,3000)
x1 = [0,30]
y1 = [theta[0] + theta[1]*a for a in x1]
plt.plot(x1,y1,color='grey')
plt.xlabel("submission number")
plt.ylabel("upvotes")
plt.yticks([0,1000,2000,3000])
plt.title("submission no. vs. upvotes")
plt.show()
```



In [49]:

```
plt.plot(x,ylog, color='k')
plt.xlim(1,30)
x1 = [0,30]
y1 = [thetalog[0] + thetalog[1]*a for a in x1]
plt.plot(x1,y1, color='grey')
plt.xlabel("submission number")
plt.ylabel("log upvotes")
plt.yticks([2,4,6,8,10])
plt.title("submission no. vs. upvotes")
plt.show()
```



In [ ]:

Exercises

2.1

Simple regression question, same form as the examples above

In [50]:
```
path = dataDir + "fantasy_100.json"
f = open(path)
data = []
for l in f:
    d = json.loads(l)
    data.append(d)
f.close()
```

In [51]:
```
ratings = [d['rating'] for d in data]
lengths = [len(d['review_text']) for d in data]
```

In [52]:
```
X = numpy.matrix([[1,l] for l in lengths]) # Note the inclusion of the constant
y = numpy.matrix(ratings).T
```

In [53]:
```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, y)
```

Out[53]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=False)
```

Extract theta

In [54]:
```
theta = model.coef_
theta
```

Out[54]:
```
array([[3.98394783e+00, 1.19363599e-04]])
```

In [55]:
```
y_pred = model.predict(X)
```

In [56]:
```
sse = sum([r**2 for r in y - y_pred])
```

MSE

In [57]:
```
mse = sse / len(y)
mse
```

Out[57]:
```
matrix([[1.26575168]])
```

2.2

Fit a model with an additional variable

In [58]:
```
def feat(d):
    f = [1]
    f.append(len(d['review_text']))
    f.append(d['n_comments'])
    return f
```

In [59]:
```
X = [feat(d) for d in data]
```

In [60]:
```
model = sklearn.linear_model.LinearRegression(fit_intercept=False)
model.fit(X, y)
```

Out[60]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=False)
```

Extract theta

In [61]:
```
theta = model.coef_
theta
```

Out[61]:
```
array([[3.95432090e+00, 7.24295694e-05, 1.08071028e-01]])
```

Compute the MSE

In [62]:
```
y_pred = model.predict(X)
sse = sum([r**2 for r in y - y_pred])
mse = sse / len(y)
mse
```

Out[62]:
```
matrix([[1.22988869]])
```

(explanation: the coefficient of length becomes smaller, as the variability in ratings is already largely explained by the number of comments)

2.3

Sketch proof:

$\frac{\partial}{\partial \theta_0} \sum_i (\theta_0 - y_i)^2 = 2 \sum_i (\theta - y_i)$

equals zero when $\sum_i \theta = \sum_i y_i$, i.e., $\theta = \frac{1}{N} \sum_i y_i$

2.4

Sketch:

$\frac{\partial}{\partial \theta_0} \sum_i |\theta_0 - y_i| = \sum_i \delta(y_i > \theta_0) - \sum_i \delta(y_i < \theta_0)$

zero when θ_0 is the median value of y (meaning that the two terms balance out)

2.5

Sketch:

$\max_\theta \prod_i \frac{1}{2b} \exp(-\frac{|x_i \cdot \theta - y_i|}{b})$

$= \max_\theta \sum_i -|x_i \cdot \theta - y_i|$

$= \min_\theta \sum_i |x_i \cdot \theta - y_i|$

2.6

Sketch:

rewrite $\sum_i (x_i \cdot \theta - y_i)^2$ as $(X\theta - y)^T (X\theta - y)$

$= (\theta^T X^T - y^T)(X\theta - y)$

$= \theta^T X^T X\theta - 2y^T X\theta - y^T y$ (all terms are scalar)

$\frac{\partial}{\partial \theta}(\theta^T X^T X\theta - 2y^T X\theta + y^T y) = 2(\theta^T X^T X - y^T X)$

$= 0$ when $\theta^T X^T X = y^T X$; or when $X^T X\theta = X^T y$

i.e., when $\theta = (X^T X)^{-1} X^T y$

2.7

Similar to 2.3: when solving θ by computing $\frac{\partial}{\partial \theta} \sum_i (x_i \cdot \theta - y_i)^2 = 2 \sum_i (x_i \cdot \theta - y_i)$, the expression will be minimized when $\sum_i (x_i \cdot \theta - y_i) = 0$

In [ ]: