Übung 2: Lineare Regression - Rainier Robles & Valentin Wolf

In [18]:
```
import numpy as np
from numpy.linalg import inv
```

Load the different training sets and prepare training sets for the various matchups

In [19]:
```
trainlist = []
trainsets = ['train3','train5','train7','train8']
for trainset in trainsets:
    trainlist.append(np.loadtxt(trainset,delimiter=','))

### set with 3 and 5
left = np.concatenate(((np.full((trainlist[0].shape[0],1),1)),trainlist[0]),axi
s=1)
right = np.concatenate(((np.full((trainlist[1].shape[0],1),-1)),trainlist[1]),a
xis=1)
threefive = np.concatenate((left,right),axis=0)

### set with 3 and 7
left = np.concatenate(((np.full((trainlist[0].shape[0],1),1)),trainlist[0]),axi
s=1)
right = np.concatenate(((np.full((trainlist[2].shape[0],1),-1)),trainlist[2]),a
xis=1)
threeseven = np.concatenate((left,right),axis=0)

### set with 3 and 8
left = np.concatenate(((np.full((trainlist[0].shape[0],1),1)),trainlist[0]),axi
s=1)
right = np.concatenate(((np.full((trainlist[3].shape[0],1),-1)),trainlist[3]),a
xis=1)
threeeight = np.concatenate((left,right),axis=0)

### set with 5 and 7
left = np.concatenate(((np.full((trainlist[1].shape[0],1),1)),trainlist[1]),axi
s=1)
right = np.concatenate(((np.full((trainlist[2].shape[0],1),-1)),trainlist[2]),a
xis=1)
fiveseven = np.concatenate((left,right),axis=0)

### set with 5 and 8
left = np.concatenate(((np.full((trainlist[1].shape[0],1),1)),trainlist[1]),axi
s=1)
right = np.concatenate(((np.full((trainlist[3].shape[0],1),-1)),trainlist[3]),a
xis=1)
fiveeight = np.concatenate((left,right),axis=0)

### set with 7 and 8
left = np.concatenate(((np.full((trainlist[2].shape[0],1),1)),trainlist[2]),axi
s=1)
right = np.concatenate(((np.full((trainlist[3].shape[0],1),-1)),trainlist[3]),a
xis=1)
seveneight = np.concatenate((left,right),axis=0)
```

Load the test set and prepare test sets for the various matchups

In [20]:
```
testset = np.loadtxt('zip.test')

testlist = []
totest = [3,5,7,8]

for i in totest:
    testlist.append(testset[testset[:,0] == i])

### set with 3 and 5
left = np.concatenate(((np.full((testlist[0].shape[0],1),1)),testlist[0][:,1:])
,axis=1)
right = np.concatenate(((np.full((testlist[1].shape[0],1),-1)),testlist[1][:,1:
]),axis=1)
threefive_test = np.concatenate((left,right),axis=0)

### set with 3 and 7
left = np.concatenate(((np.full((testlist[0].shape[0],1),1)),testlist[0][:,1:])
,axis=1)
right = np.concatenate(((np.full((testlist[2].shape[0],1),-1)),testlist[2][:,1:
]),axis=1)
threeseven_test = np.concatenate((left,right),axis=0)

### set with 3 and 8
left = np.concatenate(((np.full((testlist[0].shape[0],1),1)),testlist[0][:,1:])
,axis=1)
right = np.concatenate(((np.full((testlist[3].shape[0],1),-1)),testlist[3][:,1:
]),axis=1)
threeeight_test = np.concatenate((left,right),axis=0)

### set with 5 and 7
left = np.concatenate(((np.full((testlist[1].shape[0],1),1)),testlist[1][:,1:])
,axis=1)
right = np.concatenate(((np.full((testlist[2].shape[0],1),-1)),testlist[2][:,1:
]),axis=1)
fiveseven_test = np.concatenate((left,right),axis=0)

### set with 5 and 8
left = np.concatenate(((np.full((testlist[1].shape[0],1),1)),testlist[1][:,1:])
,axis=1)
right = np.concatenate(((np.full((testlist[3].shape[0],1),-1)),testlist[3][:,1:
]),axis=1)
fiveeight_test = np.concatenate((left,right),axis=0)

### set with 7 and 8
left = np.concatenate(((np.full((testlist[2].shape[0],1),1)),testlist[2][:,1:])
,axis=1)
right = np.concatenate(((np.full((testlist[3].shape[0],1),-1)),testlist[3][:,1:
]),axis=1)
seveneight_test = np.concatenate((left,right),axis=0)
```

In [21]:
```
trainlist[2]
```

Out[21]:
```
array([[-1., -1., -1., ..., -1., -1., -1.],
       [-1., -1., -1., ..., -1., -1., -1.],
       [-1., -1., -1., ..., -1., -1., -1.],
       ...,
       [-1., -1., -1., ..., -1., -1., -1.],
       [-1., -1., -1., ..., -1., -1., -1.],
       [-1., -1., -1., ..., -1., -1., -1.]])
```

In [22]:
```
class LinearRegression:

    def __init__(self):
        pass

    def train(self,array):
        """takes training set and splits into X (data) and y (labels)"""
        self.X = array[:,1:]
        self.y = array[:,0]
        self.beta = self.compute_beta()

    def pinv(self, X, increment=0.05):
        """attempts to invert a matrix, if not possible, then it tries again wi
th a refactored matrix"""
        I = np.eye(X.shape[0])
        my_lambda = 0
        X1 = X
        while True:
            try:
                inv(X1)
                break
            except np.linalg.LinAlgError:
                my_lambda += increment
                X1 = np.add(X, np.multiply(I,my_lambda))
        return inv(X1)

    def bigX(self,X):
        """returns array with first column filled with ones, and the rest fille
d with data from X"""
        justones = np.ones((1,X.shape[0]))
        result = np.concatenate((justones.T,X),axis=1)
        return result

    def covariance_matrix(self,X):
        """returns the covariance matrix of X"""
        return np.dot(X.T,X)

    def compute_beta(self):
        """finds beta given the training set and the label k to be tested"""
        X1 = np.dot(self.pinv(self.covariance_matrix(self.bigX(self.X))),self.b
igX(self.X).T)
        return np.dot(X1,self.y)

    def changelabels(self,arrtochange):
        """in labels, changes all positive numbers to 0s and all negative numbe
rs to 1s"""
        result = np.zeros((arrtochange.size))
        for i in range(0,arrtochange.size):
            if arrtochange[i] >= 0:
                result[i] = 0
            else:
                result[i] = 1
        return result

    def predict(self,testset):
        """classifier for exactly two classes, gives values between -1 and +1""
"
        self.X_test = self.bigX(testset[:,1:])
        self.y_test = testset[:,0]
        self.prediction = np.sum(np.multiply(self.beta.T,self.X_test),axis=1)
        return self.prediction

    def error_rate(self):
        pred = self.changelabels(self.prediction)
        testlabels = self.y_test
        datacount = pred.shape[0]
        bools = np.equal(self.changelabels(testlabels),pred.flatten())
        correct = np.sum(bools)
        return (datacount-correct)/datacount
```

3 vs 5

In [23]:
```
linreg35 = LinearRegression()
linreg35.train(threefive)
linreg35.predict(threefive_test)
print("3 vs 5 error rate = "+str(linreg35.error_rate()))
linreg35.confusion_matrix().astype('uint16')
```

```
3 vs 5 error rate = 0.0705521472393
```

Out[23]:
```
array([[154,  12],
       [ 11, 149]], dtype=uint16)
```

3 vs 7

In [24]:
```
linreg37 = LinearRegression()
linreg37.train(threeseven)
linreg37.predict(threeseven_test)
print("3 vs 7 error rate = "+str(linreg37.error_rate()))
linreg37.confusion_matrix().astype('uint16')
```

```
3 vs 7 error rate = 0.0255591054313
```

Out[24]:
```
array([[161,   5],
       [  3, 144]], dtype=uint16)
```

3 vs 8

In [25]:
```
linreg38 = LinearRegression()
linreg38.train(threeeight)
linreg38.predict(threeeight_test)
print("3 vs 8 error rate = "+str(linreg38.error_rate()))
linreg38.confusion_matrix().astype('uint16')
```

```
3 vs 8 error rate = 0.0481927710843
```

Out[25]:
```
array([[156,  10],
       [  6, 160]], dtype=uint16)
```

5 vs 7

In [26]:
```
linreg57 = LinearRegression()
linreg57.train(fiveseven)
linreg57.predict(fiveseven_test)
print("5 vs 7 error rate = "+str(linreg57.error_rate()))
linreg57.confusion_matrix().astype('uint16')
```

```
5 vs 7 error rate = 0.0162866449511
```

Out[26]:
```
array([[157,   3],
       [  2, 145]], dtype=uint16)
```

5 vs 8

In [27]:
```
linreg58 = LinearRegression()
linreg58.train(fiveeight)
linreg58.predict(fiveeight_test)
print("5 vs 8 error rate = "+str(linreg58.error_rate()))
linreg58.confusion_matrix().astype('uint16')
```

```
5 vs 8 error rate = 0.0368098159509
```

Out[27]:
```
array([[154,   6],
       [  6, 160]], dtype=uint16)
```

7 vs 8

In [28]:
```
linreg78 = LinearRegression()
linreg78.train(seveneight)
linreg78.predict(seveneight_test)[:10]
```

Out[28]:
```
array([  3.88427020e+12,   3.88427020e+12,   3.88427020e+12,
         3.88427020e+12,   3.88427020e+12,   3.88427020e+12,
         3.88427020e+12,   3.88427020e+12,   3.88427020e+12,
         3.88427020e+12])
```

In [29]:
```
print("7 vs 8 error rate = "+str(linreg78.error_rate()))
linreg78.confusion_matrix().astype('uint16')
```

```
7 vs 8 error rate = 0.5303514377
```

Out[29]:
```
array([[147,   0],
       [166,   0]], dtype=uint16)
```

In [30]:
```
linreg78.X_test
```

Out[30]:
```
array([[ 1.  , -1.  , -0.99, ..., -1.  , -1.  , -1.  ],
       [ 1.  , -1.  , -1.  , ..., -1.  , -1.  , -1.  ],
       [ 1.  , -1.  , -1.  , ..., -1.  , -1.  , -1.  ],
       ...,
       [ 1.  , -1.  , -1.  , ..., -1.  , -1.  , -1.  ],
       [ 1.  , -1.  , -1.  , ..., -1.  , -1.  , -1.  ],
       [ 1.  , -1.  , -1.  , ..., -1.  , -1.  , -1.  ]])
```

In [31]:
```
linreg78.beta[:10]
```

Out[31]:
```
array([  3.08378652e+12,  -2.42522583e+01,   4.36608887e+00,
        -6.03210449e+00,   3.46725464e+00,   1.34539795e+00,
        -3.82171631e-01,   1.30364990e+00,  -4.58984375e-01,
         1.40495300e+00])
```

Wir wissen nicht warum hier alles als 8 klassifiziert wird, möglicherweise sind 7 und 8 zu ähnlich. Sehr verwirrend.