TODO

- Add derivations requested in class

Load and standardize the data

Download the prostate cancer dataset from the course website.

In this prostate cancer study 9 variables (including age, log weight, log cancer volume, etc.) were measured for **97 patients**.

We will now construct a model to **predict the 9th variable**, a linear combination of the other 8.

A description of this dataset appears in the textbook of Hastie et al, freely available on the course website:

> "The data for this example come from a study by Stamey et al. (1989) that examined the correlation between the level of prostate specific antigen (PSA) and a number of clinical measures, in 97 men who were about to receive a radical prostatectomy.
>
> **The goal is to predict the log of PSA (lpsa)** from a number of measurements including log cancer volume (lcavol), log prostate weight lweight, age, log of benign prostatichyperplasia amount lbph, seminal vesicle invasion svi, log of capsular penetration lcp, Gleason score gleason, and percent of Gleason scores 4 or 5 pgg45."

"***The Elements of Statistical Learning.*** *Data Mining, Inference, and Prediction*"

Trevor Hastie, Robert Tibshirani, Jerome Friedman

https://web.stanford.edu/~hastie/ElemStatLearn//printings/ESLII_print12.pdf (https://web.stanford.edu/~hastie/ElemStatLearn//printings/ESLII_print12.pdf)

In [250]:
```
import numpy as np
```

In [251]:
```
# 1. First load the data and split it into a response vector (y) and a matri
x of attributes (X),
X = np.loadtxt('prostate.data.txt', skiprows=1, delimiter='\t')
y = X[:, -1]
X = X[:,0: -1]
print('Patients: {}'.format(len(y)))
print('  lcavol\tlweight\t  age\t\tlbph\t\tsvi\tlcp\tgleason\tpgg45\tlpsa')
print(X[0, :])
print(y[0])
```

```
Patients: 97
  lcavol        lweight   age           lbph            svi     lcp     gleas
on      pgg45   lpsa
[-0.5798185   2.769459   50.         -1.38629436  0.         -1.38629436
  6.          0.        ]
-0.4307829
```

In [252]:
```
#Choose the first 50 patients as the training data. The remaining patients w
ill be the test data.
y_train, y_test = y[0:50], y[50:]
X_train, X_test = X[0:50], X[50:]

print('Training set: {} elements'.format(len(y_train)))
print('Test set: {} elements'.format(len(y_test)))
```

```
Training set: 50 elements
Test set: 47 elements
```

In [253]:
```
# Set both variables to have zero mean and standardize the input variables t
o have unit variance.
# Get mean and std
X_bar = np.mean(X_train, axis=0)
X_std = np.std(X_train, axis=0)
y_bar = np.mean(y_train)

# Before normalization
print('Before normalization')
print(X_train[0:3])
print(y_train[0:3], '\n')
# Normalize
y_train = y_train - y_bar
X_train = (X_train - X_bar) / X_std

#After normalization
print('After normalization')
print(X_train[0:3])
print(y_train[0:3])
```

```
Before normalization
[[-0.5798185   2.769459   50.         -1.38629436  0.         -1.38629436
   6.          0.        ]
 [-0.99425227  3.319626   58.         -1.38629436  0.         -1.38629436
   6.          0.        ]
 [-0.51082562  2.691243   74.         -1.38629436  0.         -1.38629436
   7.         20.        ]]
[-0.4307829 -0.1625189 -0.1625189]

After normalization
[[-1.26972558 -1.84388102 -1.72937235 -0.92242702 -0.20412415 -0.62841263
  -0.7428692  -0.56945562]
 [-1.67344433 -0.34399259 -0.64000394 -0.92242702 -0.20412415 -0.62841263
  -0.7428692  -0.56945562]
 [-1.2025165  -2.05711677  1.53873288 -0.92242702 -0.20412415 -0.62841263
   0.63281451  0.26065461]]
[-2.05581329 -1.78754929 -1.78754929]
```

Important detail:

In the training step, we will learn the bias θ_0 **separately**.

We do this because it makes **no sense** to apply regularization to the **bias** θ_0.

Recall that the purpose of regularization is to **get rid of unwanted input attributes**.

We will need the terms (X_bar, X_std, y_bar) when we do predictions.

Mathematically, what we are saying is that the bias term will be computed separately as follows:

$$\theta_0 = \bar{y} - \bar{x}^T \hat{\theta}_j$$

where $\bar{y}$ is the mean of the elements of the training data vector y and $\bar{x}^T$ is the vector of 8 means for the input attributes.

Note that in this case the 8-dimensional parameter vector $\hat{\theta}$ includes all the parameters other than the bias term that have been **learned with ridge regression**.

That is, we first learn $\hat{\theta}$ using standardized data and then proceed to learn θ_0 (bias).

When we encounter a **new** input x^* in the **test set**, we need **to standardize** it before making a prediction.

The actual prediction should be:

$$\hat{y} = \bar{y} + \sum_{j=1}^{8} \frac{x_j^* - \bar{x}_j}{\sigma_j} \hat{\theta}_j$$

Figure 1: Regularization path for ridge regression.

where $\hat{x}_j$ and σ_j are the **mean* and** standard deviation **of the j-th attribute obtained from the** training data**.

One reason for standardizing the inputs is that we want them to be **comparable**.

If we had an input much bigger than the other, we would have wanted to apply a different regularizer to it.

By standardizing the inputs first, we only need **a single scalar regularization** coefficient δ^2.

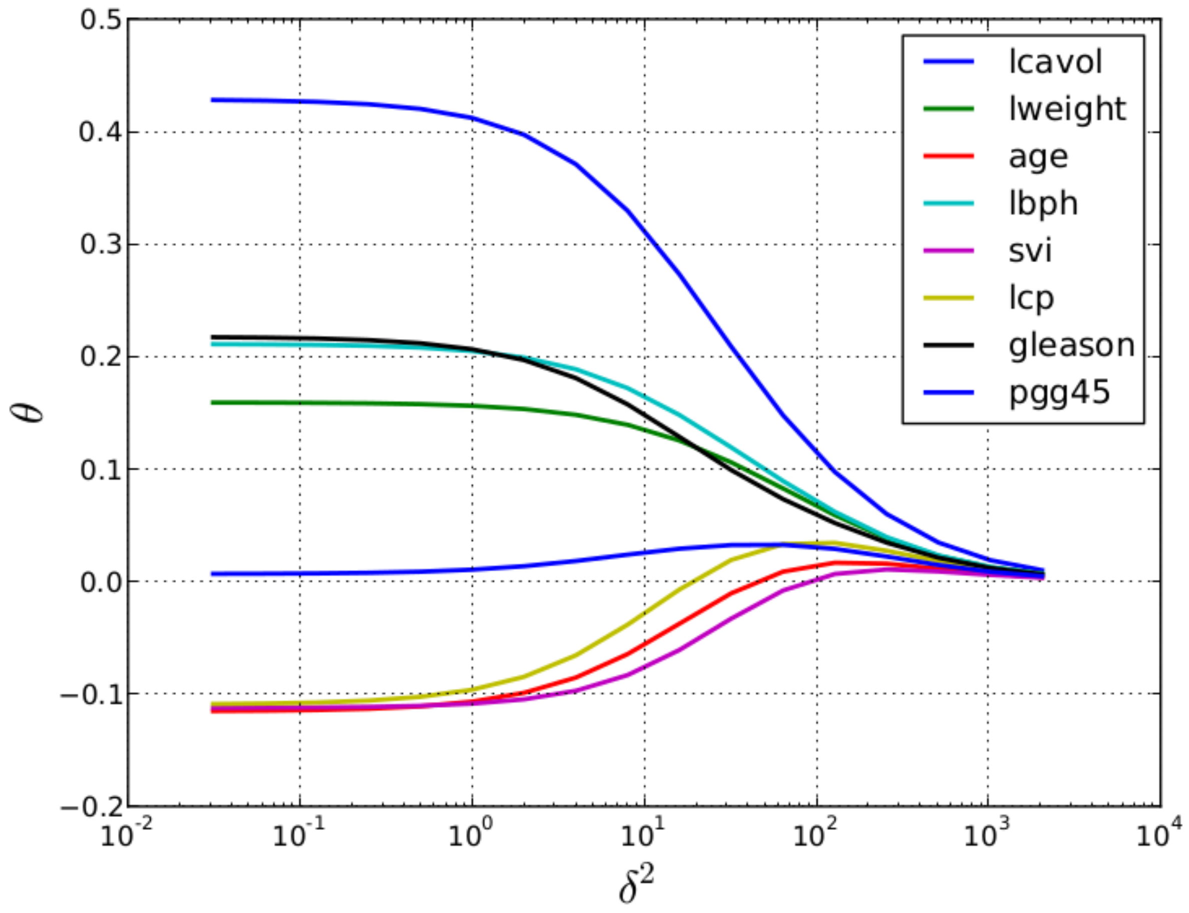


Figure 1: Regularization path for ridge regression.

1.3 Ridge regression

$$\hat{\theta} = (X^T X + \delta^2 I)^{-1}(y^T X)$$

We will now construct a model using ridge regression to predict the 9th variable as a linear combination of the other 8.

1. Write code for ridge regression starting from the following skeleton:

   def ridge(X, y, d2):

   ```
   ???
   return theta
   ```

```
In [254]: def ridge(X, y, d2):
              # Identity matrix of size 8
              num_feat = np.shape(X)[1]
              I = np.eye(num_feat)
              # shape(X) : (97, 8)
              Xt = np.transpose(X)
              yt = np.transpose(y)
              XtX = np.dot(Xt, X)
              ytX = np.dot(yt, X)
              theta = np.dot(np.linalg.inv(XtX + d2*I), ytX)
              return theta
```

Compute the ridge regression solutions for a range of regularizers (δ^2).

Plot the values of each θ in the y-axis against δ^2 in the x-axis.

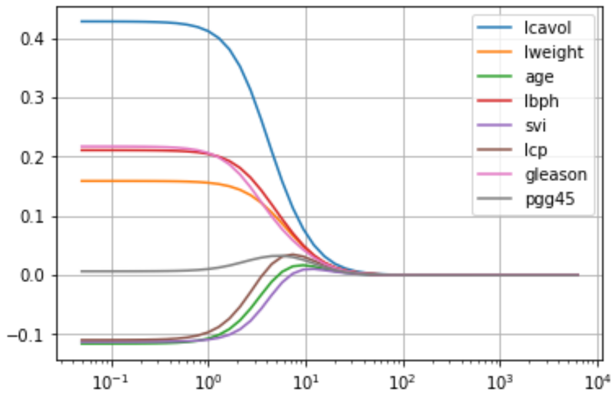
This set of plotted values is known as a regularization path.

Your plot should look like Figure 1. Hand in your version of this plot, along with the code you used to generate it.

In [255]:
```
def get_thetas(X_train, y_train):
    thetas = []
    d2s = []
    for i in np.arange(-3, 9, 0.25):
        d2 = 10**i
        d2s.append(d2)
        thetas.append(ridge(X_train, y_train, d2))
    return thetas

thetas = get_thetas(X_train, y_train)
print('thetas shape:', np.shape(thetas))
```

```
thetas shape: (48, 8)
```

In [256]:
```
import matplotlib.pyplot as plt

labels=['lcavol','lweight','age','lbph','svi','lcp','gleason','pgg45']

plt.plot(d2s, thetas)
plt.xscale('log')
plt.grid()
plt.legend(labels)
plt.show()
```



For each computed value of θ, compute the train and test error.

Remember, you will have to standardize your test data with the same means and standard deviations as above (X_bar, X_std, y_bar) before you can make a prediction and compute your test error.

In other words, to make a prediction do:

y_hat = y_bar + numpy.dot((X_test - Xbar) / X_std, theta)

Choose a value of δ^2 **using cross-validation**.

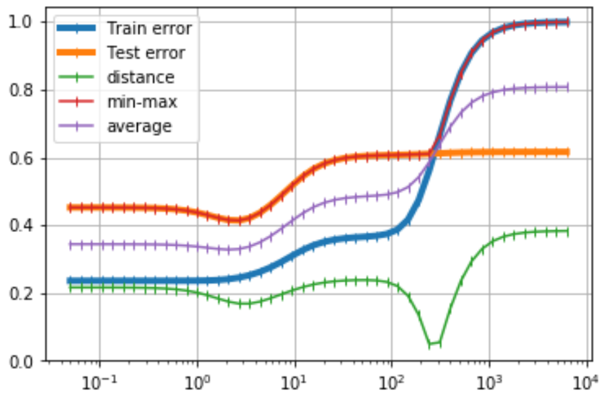
What is this value? Show all your intermediate cross-validation steps and the criterion you used to choose δ^2.

Plot the train and test errors as a function of δ^2 .

In [257]:
```
train_errors = np.zeros(len(d2s))
test_errors  = np.zeros(len(d2s))
cv_folds = 5
for f in range(cv_folds):
    # 5-fold cross validation
    #  + 19 for test
    #  + 79 for training
    data_size = len(y)
    test_size = 19
    # | 1 | 2 | 3 | 4 | 5 |
    start_test = 0+f*test_size
    end_test = start_test + test_size
    start_1_train = 0
    end_1_train   = max(0, start_test)
    start_2_train = min(end_test, data_size)
    end_2_train   = -1
    # Separate data
    y_test = y[start_test:end_test]
    X_test = X[start_test:end_test]
    y_train = np.concatenate((y[start_1_train:end_1_train], y[start_2_train:
end_2_train]))
    X_train = np.concatenate((X[start_1_train:end_1_train], X[start_2_train:
end_2_train]))
    # Get thetas for training data
    thetas = get_thetas(X_train, y_train)
    # Calculate mean and std
    X_bar = np.mean(X_train, axis=0)
    X_std = np.std(X_train, axis=0)
    y_bar = np.mean(y_train)
    standar_X_test = (X_test - X_bar) / X_std
    for i, theta in enumerate(thetas):
        # Training error (accumulates over k-folds)
        y_hat_train = np.dot(X_train, theta)
        error = np.linalg.norm((y_train - y_hat_train), ord=2)/np.linalg.nor
m(y_train, ord=2)
        train_errors[i] += error
        # Test error (accumulates over k-folds)
        y_hat_test  = y_bar + np.dot(standar_X_test, theta)
        error = np.linalg.norm((y_test - y_hat_test), ord=2)/np.linalg.norm
(y_test, ord=2)
        test_errors[i] += error

# Take mean of k-folds errors
train_errors = train_errors / cv_folds
test_errors = test_errors / cv_folds
```

In [ ]:
```
```

In [265]:
```
plt.plot(d2s, train_errors, label='Train error', marker='|', linewidth=4)
plt.plot(d2s, test_errors, label='Test error', marker='|', linewidth=4)
plt.plot(d2s, abs(test_errors-train_errors), label='distance', marker='|')
plt.plot(d2s, np.maximum(test_errors,train_errors), label='min-max', marke
r='|')
plt.plot(d2s, (test_errors+train_errors)/2., label='average', marker='|')
plt.xscale('log')
plt.grid()
plt.legend()
plt.show()
#print(d2s)
```



In [267]:
```
best_d2_idx = np.argmin((test_errors+train_errors)/2.)
best_d2 = d2s[best_d2_idx]
print('d^2 chosen by best average error:')
print('best d^2 = 10^{:.2f} ~ {}\n'.format(best_d2, 10**best_d2))

best_d2_idx = np.argmin(np.maximum(test_errors,train_errors))
best_d2 = d2s[best_d2_idx]
print('d^2 chosen by min-max error:')
print('best d^2 = 10^{:.2f} ~ {}\n'.format(best_d2, 10**best_d2))

best_d2_idx = np.argmin(abs(test_errors-train_errors))
best_d2 = d2s[best_d2_idx]
print('d^2 chosen by min difference:')
print('best d^2 = 10^{:.2f} ~ {}'.format(best_d2, 10**best_d2))
```

```
d^2 chosen by best average error:
best d^2 = 10^2.12 ~ 130.91819730783624

d^2 chosen by min-max error:
best d^2 = 10^2.72 ~ 522.7352996704365

d^2 chosen by min difference:
best d^2 = 10^244.69 ~ 4.919627995404652e+244
```

Answer: Best $\delta^2 = 523$

$$\hat{\theta}_{best} = (X^T X + 523\, I\,)^{-1}(y^T X)$$

Bayesian Linear Regression

Let $X \in R^{n \times d}$ and $y \in R^{n \times 1}$.

Assume the likelihood is Gaussian:

$$p(y|X, \theta, \Sigma) = |2\pi\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(y - X\theta)^T \Sigma^{-1} (y - X\theta)}$$

Assume that the prior for θ is also Gaussian:

$$p(\theta) = |2\pi\Delta|^{-\frac{1}{2}} e^{-\frac{1}{2}\theta^T \Delta^{-1} \theta}$$

1. Using Bayes rule and completing squares, derive an expression for the posterior distribution of θ.

   In this part, assume that the covariance Σ **is given**.

   State clearly what the mean and variance of the posterior are.

   Also, state the conditions under which the posterior mean would be equivalent to the ridge and maximum likelihood estimators.

1. Derive an analytical, closed-form expression for the maximum likelihood estimate of Σ.

   You will need the following properties of matrix derivatives (for a symmetric matrix A):

$$\frac{\partial log|A0|}{\partial A} = A^{-1}$$
$$\frac{\partial trace(X^T A X)}{\partial A} = XX^T$$

Hint: Take derivatives of the log-likelihood with respect to $\Sigma - 1$.

```
In [ ]:
```