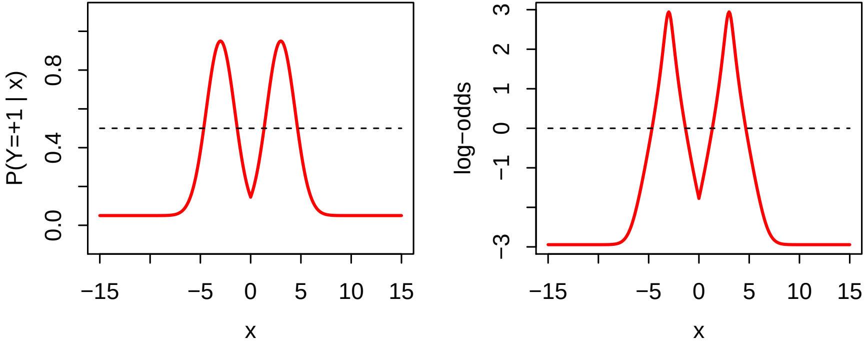
HT2015: SC4
Statistical Data Mining and Machine Learning

Dino Sejdinovic
Department of Statistics
Oxford

http://www.stats.ox.ac.uk/~sejdinov/sdmml.html

A Hard 1D classification task

```
## true conditional probabilities
truep <- function(x) {
  return(0.05+0.9*pmax(exp(-(x-3)^2/4),exp(-(x+3)^2/4)))
}
x  <- seq(-15,15,.1)
condp  <- truep(x) #P(Y=+1 | x)
par(mar=c(4,4,.1,.1),cex.lab=.95,cex.axis=.9,mgp=c(2,.7,0),tcl=-.3)
plot(x,condp,type='l',lwd=2,col=2,ylim=c(-.1,1.1),ylab='P(Y=+1 | x)')
lines(c(-15,15),c(0.5,0.5),lty=2)
plot(x,log(condp/(1-condp)),type='l',lwd=2,col=2,ylab='log-odds')
lines(c(-15,15),c(0.0,0.0),lty=2)
```



A linear decision boundary is not helpful. Log-odds are far from linear.

Nonlinear features

Use the transformed dataset

$$x \mapsto \varphi(x) = (x, x^2, x^3, \ldots, x^p).$$

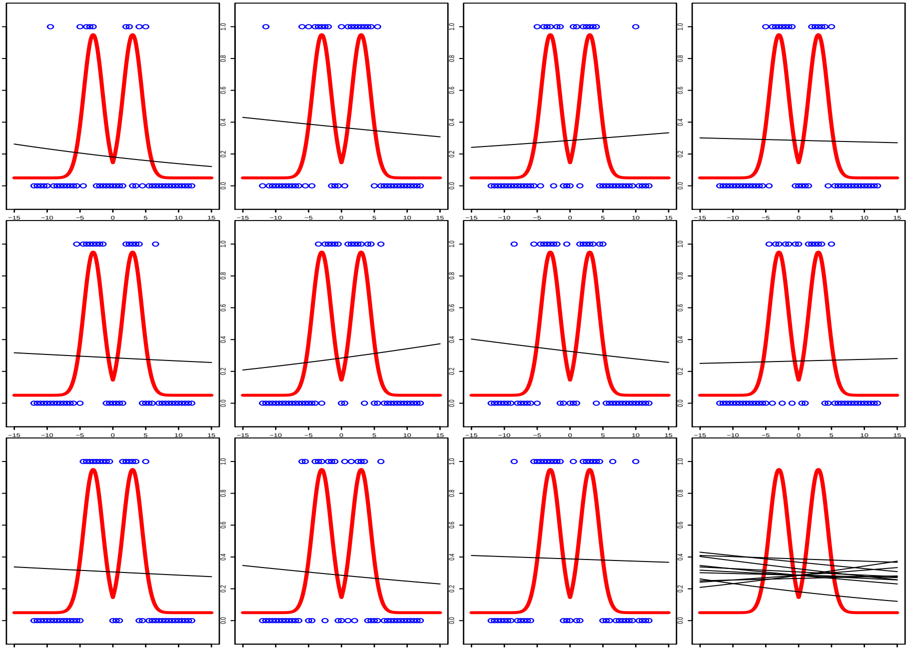
```r
## extract nonlinear features: {x^i}
phi <- function(x,deg) {
  d <- matrix(0,length(x),deg)
  for (i in 1:deg) {
    d[,i] <- x ^ i
  }
  return (data.frame(d))
}
```

Demo on Overfitting in Logistic Regression

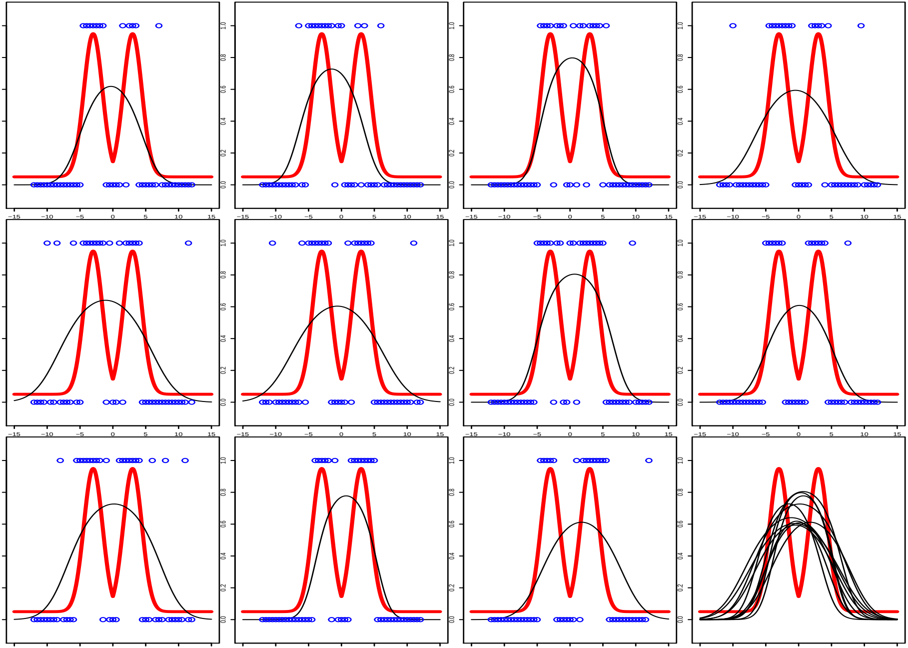
```r
set.seed(123)
## demo learning logistic regression, with different datasets generated,
## and using different degree polynomials as features
demolearn <- function(trainx,testx,truep,deg) {
  trainp <- truep(trainx)
  testp  <- truep(testx)
  par(mfrow=c(3,4),ann=FALSE,cex=.3,mar=c(1,1,1,1))
  predp <- matrix(0,length(testx),11)
  for (i in 1:11) {
    trainy <- as.numeric(runif(length(trainx)) < trainp)
    lr <- glm(trainy ~ .,data=phi(trainx,deg),family=binomial)
    predp[,i] <- predict(lr,newdata=phi(testx,deg),type="response")
    plot(testx,testp,type="l",col=2,lwd=3,ylim=c(-.1,1.1))
    lines(testx,predp[,i],type="l")
    points(trainx,trainy,pch=1,col=4,cex=2)
  }
  plot(testx,testp,type="l",lwd=3,col=2,ylim=c(-.1,1.1))
  for (i in 1:11) {
    lines(testx,predp[,i],type="l")
  }
  return(predp)
}

trainx <- seq(-12,12,.5)
testx  <- seq(-15,15,.1)
```

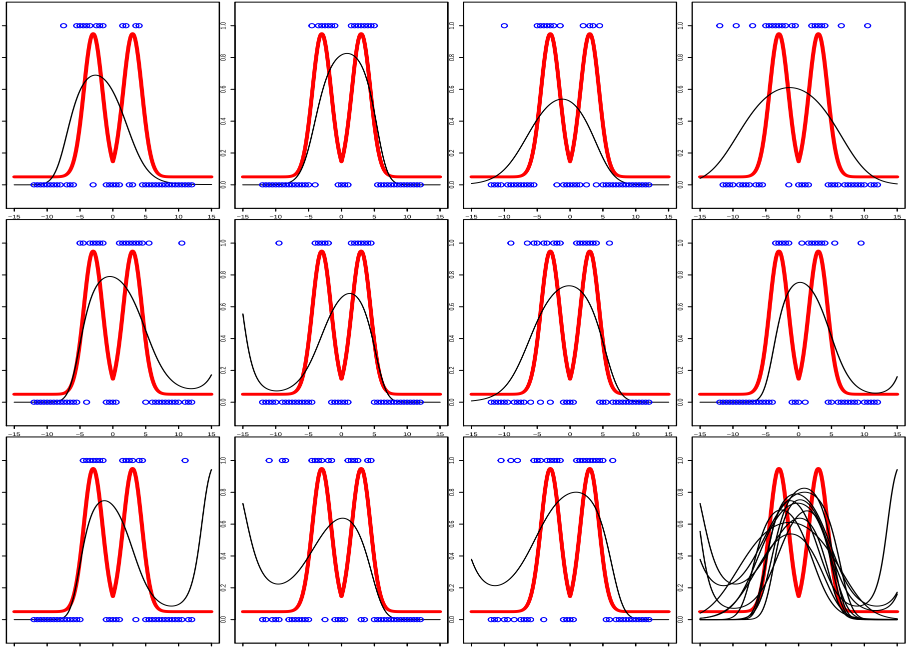
```r
pp <- demolearn(trainx,testx,truep,1)
```



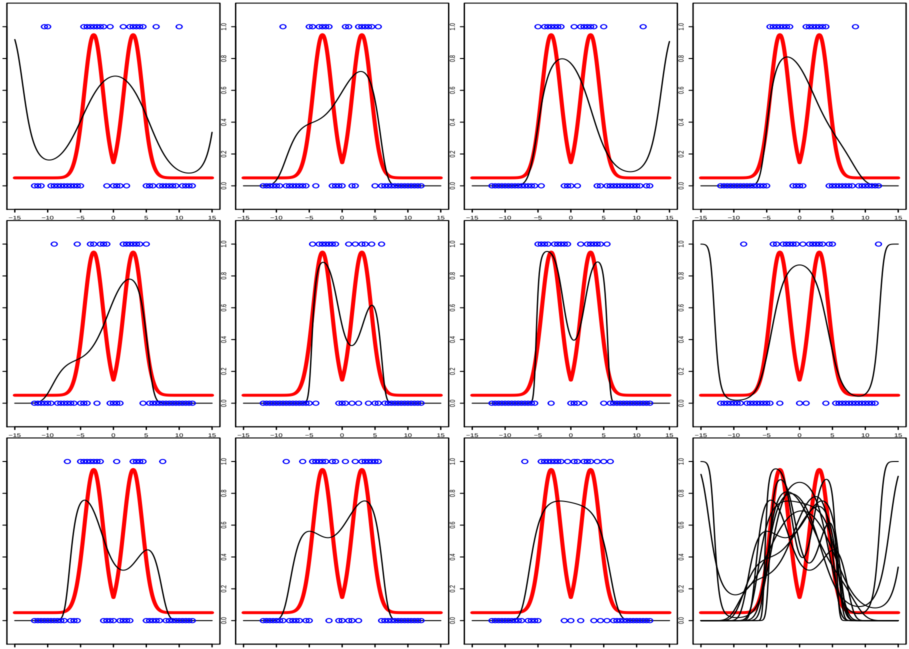
```
pp <- demolearn(trainx,testx,truep,2)
```



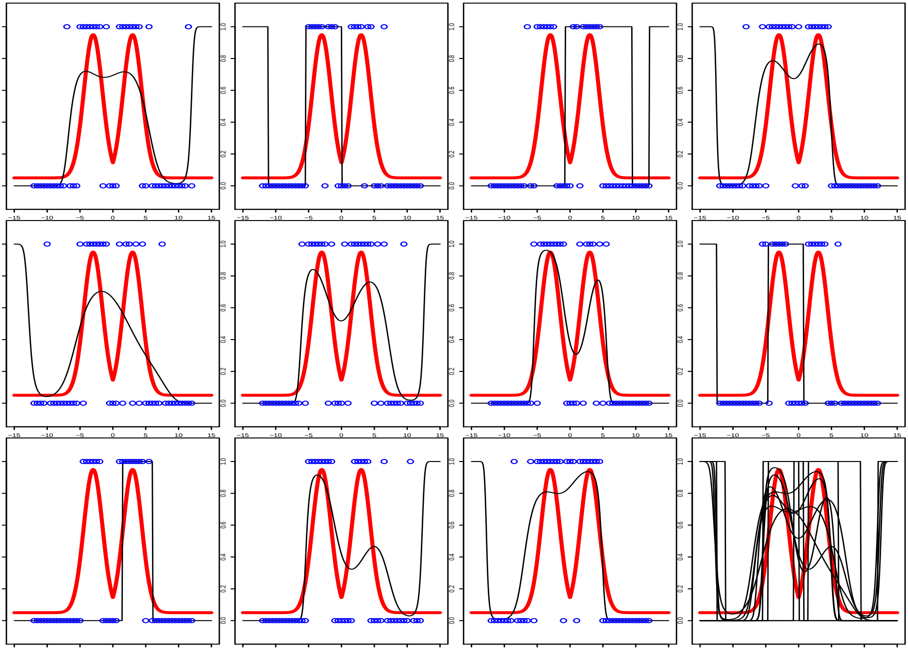
```
pp <- demolearn(trainx,testx,truep,3)
```



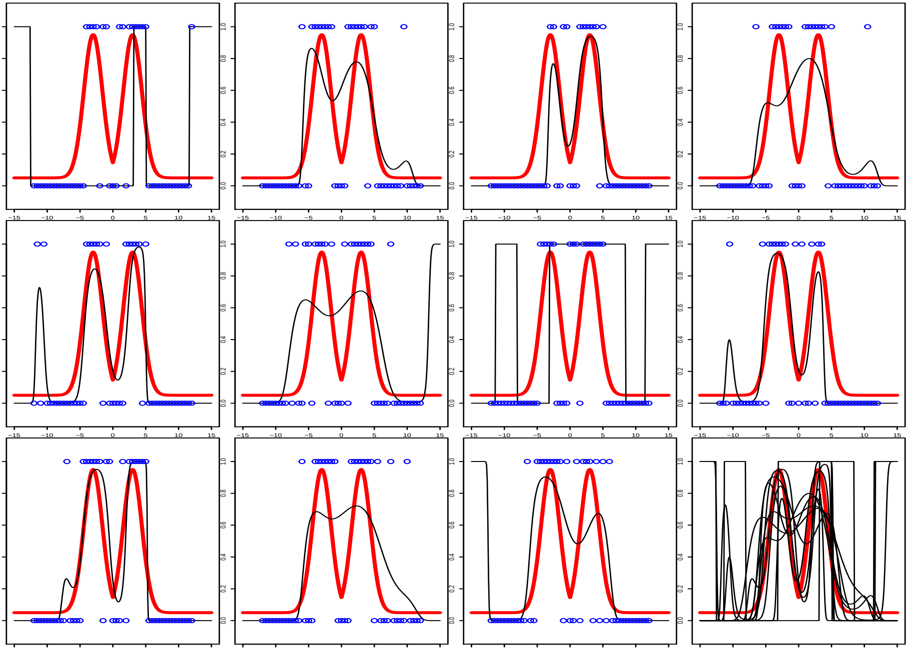
```
pp <- demolearn(trainx,testx,truep,4)
```



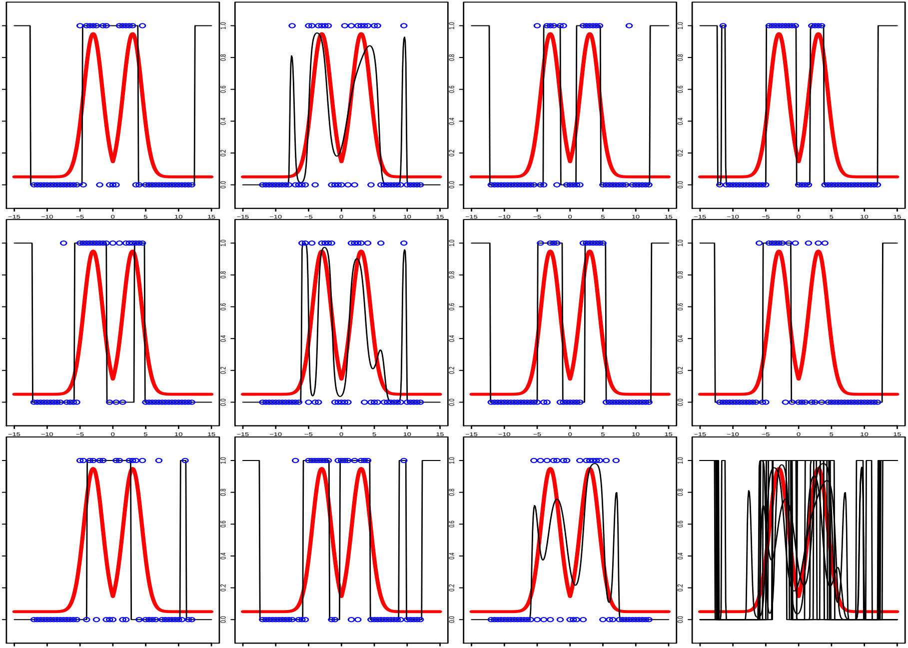
```
pp <- demolearn(trainx,testx,truep,5)
```



```r
pp <- demolearn(trainx,testx,truep,6)
```



```
pp <- demolearn(trainx,testx,truep,10)
```



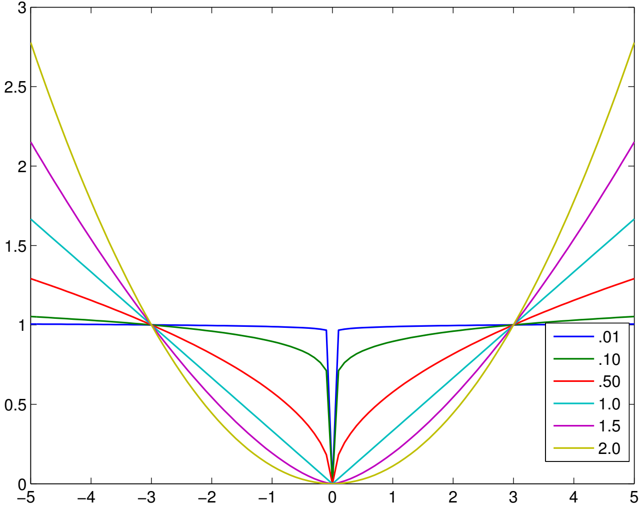
Regularization

- Flexible models for high-dimensional problems require many parameters.
- With many parameters, learners can easily overfit.
- **Regularization**: Limit flexibility of model to prevent overfitting.
- Add term **penalizing large values of parameters** θ.

$$\min_{\theta} R^{\text{emp}}(f_\theta) + \lambda \|\theta\|_\rho^\rho = \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_\theta(x_i)) + \lambda \|\theta\|_\rho^\rho$$

  where $\rho \geq 1$, and $\|\theta\|_\rho = (\sum_{j=1}^{p} |\theta_j|^\rho)^{1/\rho}$ is the L_ρ norm of θ (also of interest when $\rho \in [0, 1)$, but is no longer a norm).
- Also known as **shrinkage** methods—parameters are shrunk towards 0.
- λ is a **tuning parameter** (or **hyperparameter**) and controls the amount of regularization, and resulting complexity of the model.

Regularization



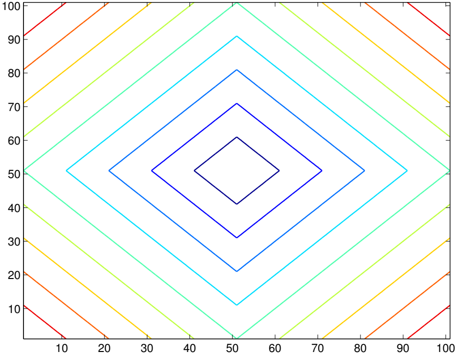
L_ρ regularization profile for different values of ρ.

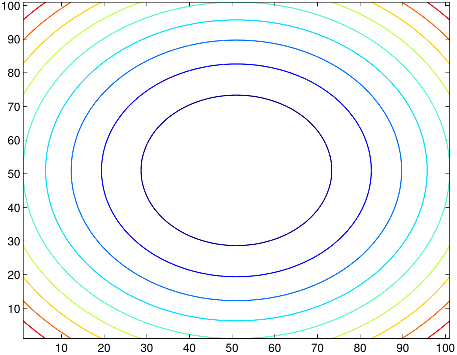
Types of Regularization

- **Ridge regression / Tikhonov regularization**: $\rho = 2$ (Euclidean norm)
- **LASSO**: $\rho = 1$ (Manhattan norm)
- **Sparsity-inducing** regularization: $\rho \leq 1$ (nonconvex for $\rho < 1$)
- **Elastic net** regularization: mixed L_1/L_2 penalty:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_\theta(x_i)) + \lambda \left[(1-\alpha)\|\theta\|_2^2 + \alpha \|\theta\|_1 \right]$$

Shape of regularization





L_1 and L_2 norm contours.

L_1 promotes sparsity

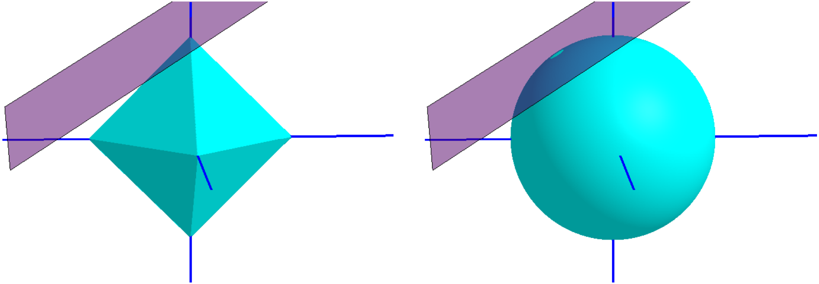


Figure : The intersection between the L_1 (left) and the L_2 (right) ball with a hyperplane.

L_1 regularization often leads to optimal solutions with many zeros, i.e., the regression function depends only on the (small) number of features with non-zero parameters.

figure from M. Elad, Sparse and Redundant Representations, 2010.

L_1-regularization in **R**: `glmnet`

`glmnet` computes the regularization for the Lasso or elastic net penalty at a grid of values for the regularization parameter λ.

```
library(glmnet)
trainy <- as.numeric(runif(length(trainx)) < truep(trainx))
slr <- glmnet(as.matrix(phi(trainx, 6)), as.factor(trainy), family = "binomial")
```

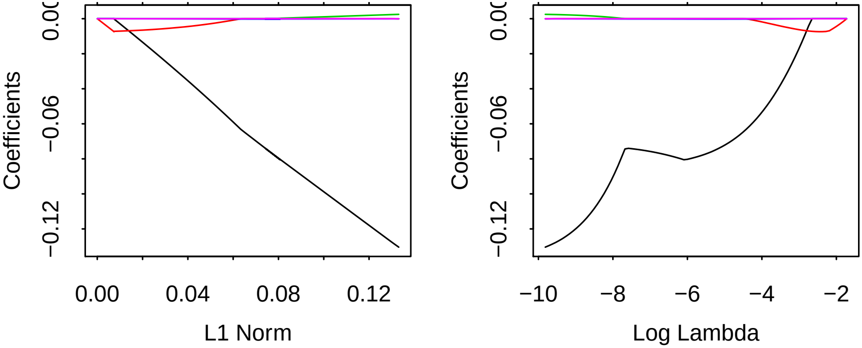
Can obtain actual coefficients at various values of λ.

```
coef(slr,s=c(0.001,0.01,0.1))
```

```
## 7 x 3 sparse Matrix of class "dgCMatrix"
##                          1            2             3
## (Intercept)  3.296780e-02  0.004495835 -2.550538e-01
## X1          -7.613699e-02 -0.066486153  .
## X2           .             .           -7.323870e-03
## X3           .             .            .
## X4          -1.838644e-04 -0.000209874 -7.621291e-06
## X5          -1.801337e-06  .            .
## X6          -6.408190e-07  .            .
```

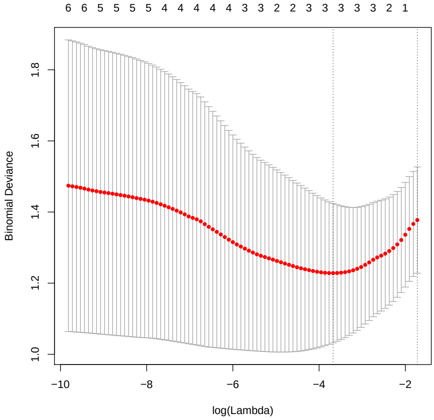
Regularization path

```
par(mar=c(4,4,.1,.1),cex.lab=.95,cex.axis=.9,mgp=c(2,.7,0),tcl=-.1)
plot(slr)
plot(slr,xvar="lambda")
```



Fitting λ by cross-validation.

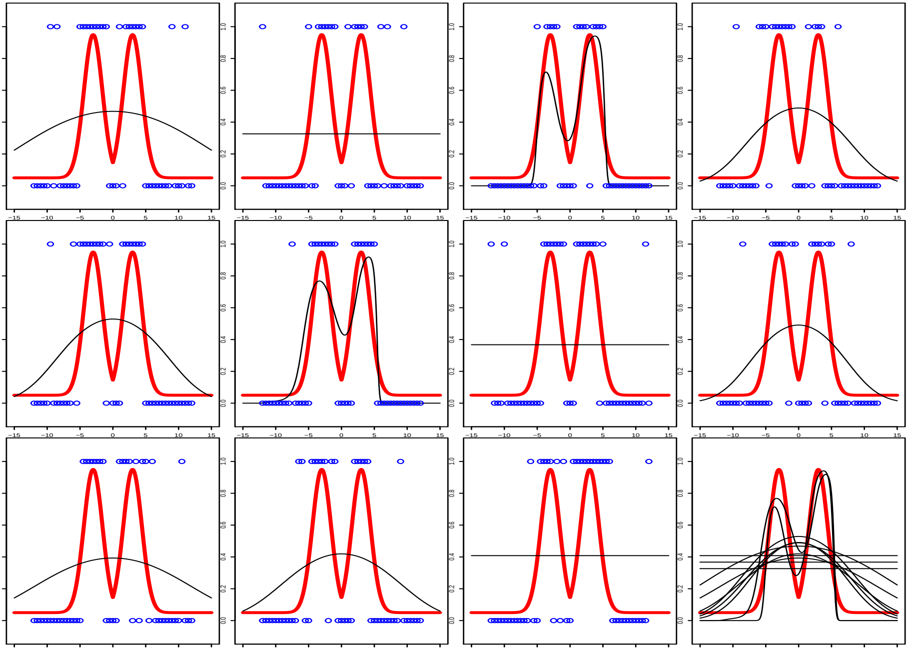
```
cv.slr <- cv.glmnet(as.matrix(phi(trainx,6)),as.factor(trainy),
          family="binomial")
cv.slr$lambda.min #minimum mean cross-validated error
## [1] 0.02534005
cv.slr$lambda.1se #most regularized model within one std error of minimum
## [1] 0.178769
plot(cv.slr)
```



Demo on L_1-Regularized Logistic Regression

```r
demolearnL1 <- function(trainx, testx, truep, deg = 6) {
    trainp <- truep(trainx)
    testp <- truep(testx)
    par(mfrow = c(3, 4), ann = FALSE, cex = 0.3, mar = c(1, 1, 1, 1))
    predp <- matrix(0, length(testx), 11)
    for (i in 1:11) {
        trainy <- as.numeric(runif(length(trainx)) < trainp)
        cv.slr <- cv.glmnet(as.matrix(phi(trainx, deg)), as.factor(trainy),
            family = "binomial")
        predp[, i] <- predict(cv.slr, newx = as.matrix(phi(testx, deg)), s = cv.slr$lambda.1se,
            type = "response")
        plot(testx, testp, type = "l", col = 2, lwd = 3, ylim = c(-0.1, 1.1))
        lines(testx, predp[, i], type = "l")
        points(trainx, trainy, pch = 1, col = 4, cex = 2)
    }
    plot(testx, testp, type = "l", lwd = 3, col = 2, ylim = c(-0.1, 1.1))
    for (i in 1:11) {
        lines(testx, predp[, i], type = "l")
    }
}
```

```
demolearnL1(trainx,testx,truep,deg=6)
```



```
demolearnL1(trainx,testx,truep,deg=10)
```