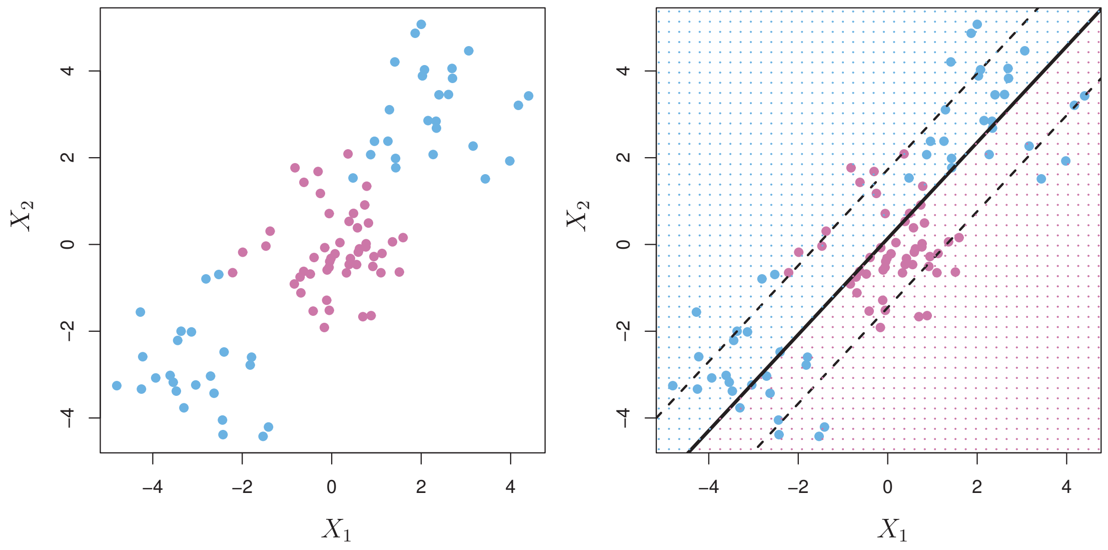
Support Vector Machines

Classification with Non-Linear Decision Boundaries

- The support vector classifier is a natural approach for classification in the two-class setting, if the boundary between the two classes is linear.
- However, in practice we are sometimes faced with non-linear class boundaries.



Left: The observations fall into two classes, with a non-linear boundary between them. Right: The support vector classifier seeks a linear boundary, and consequently performs very poorly.

- We could address the problem of possibly non-linear boundaries between classes by enlarging the feature space using quadratic, cubic, and even higher-order polynomial functions of the predictors.
- For instance, rather than fitting a support vector classifier using p features

$$X_1, X_2, \ldots, X_p,$$

we could instead fit a support vector classifier using $2p$ features

$$X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2.$$

Then the problem would become

$$\begin{aligned}
&\underset{\beta_0,\beta_{11},\beta_{12},\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} \quad M \\
&\text{subject to} \quad \sum_{j=1}^{p}\sum_{k=1}^{2}\beta_{jk}^2 = 1, \\
&y_i\left(\beta_0 + \sum_{j=1}^{p}\beta_{j1}x_{ij} + \sum_{j=1}^{p}\beta_{j2}x_{ij}^2\right) \geqslant M(1-\epsilon_i), \\
&\epsilon_i \geqslant 0, \quad \sum_{i=1}^{n}\epsilon_i \leqslant C
\end{aligned}$$

- One might additionally want to enlarge the feature space with higher-order polynomial terms, or with interaction terms of the form $X_jX_{j'}$ for $j \neq j'$. Alternatively, other functions of the predictors could be considered rather than polynomials.
- It is not hard to see that there are many possible ways to enlarge the feature space so we could end up with a huge number of features. Then computations would become unmanageable.

The Support Vector Machine

- The support vector machine (SVM) is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using kernels.
- The kernel approach is simply an efficient computational approach for enlarging the feature space in order to accommodate a non-linear boundary between the classes.

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

where there are n parameters α_i, $i = 1, \ldots, n$, one per training observation.

The inner product of two r-vectors a and b is defined as $\langle a, b \rangle = \sum_{i=1}^{r} a_i b_i$.

$a = (a_1, a_2, \ldots, a_r)^T$, $b = (b_1, b_2, \ldots, b_r)^T$

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0 , all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.

- It turns out that α_i is nonzero only for the support vectors in the solution—that is, if a training observation is not a support vector, then its α_i equals zero. So if S is the collection of indices of these support points, we can rewrite any solution function as

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

- We can replace each inner product in $f(x)$ with a generalization of the inner product of the form

$$K(x_i, x_{i'})$$

where K is some function that we will refer to as a *kernel.*

For instance, we could simply take

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j} = \langle x_i, x_{i'} \rangle \quad (1)$$

which would just give us back the support vector classifier. Equation (1) is known as a linear kernel because the support vector classifier is linear in the features.

- One could replace every inner product in $f(x)$ with the quantity

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij}x_{i'j})^d \qquad (2)$$

  This is known as a *polynomial kernel* of degree d, where d is a positive integer.

- Using such a kernel with $d > 1$, instead of the standard linear kernel, in the support vector classifier algorithm leads to a much more flexible decision boundary.

when the support vector classifier is combined with a non-linear kernel such as (2), the resulting classifier is known as a support vector machine.

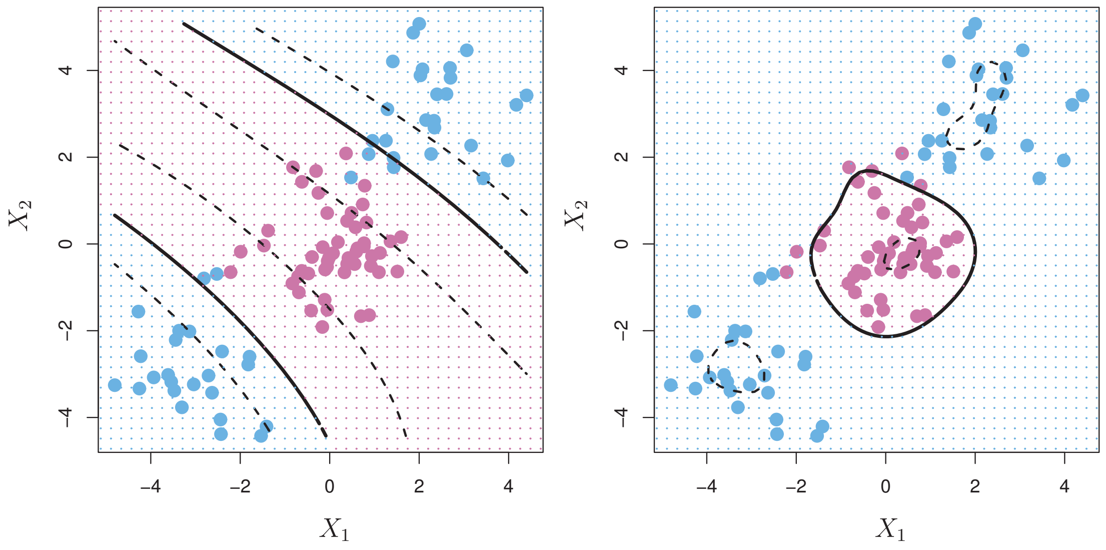
- Note that in this case the (non-linear) function has the form

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i)$$

- Another popular choice of a kernel is the *radial* kernel, which takes the form

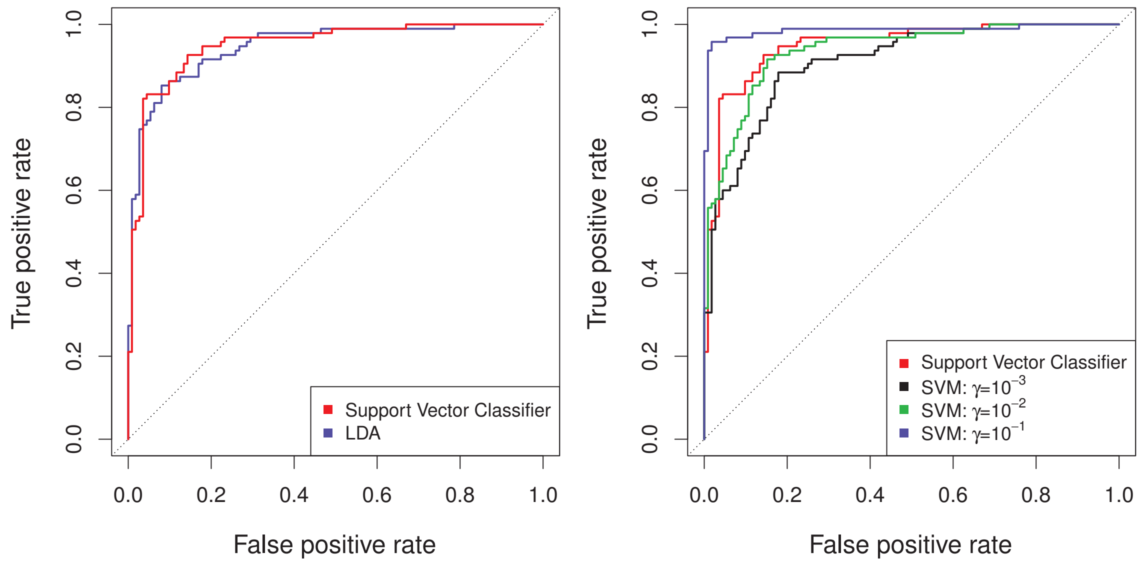
$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right),$$

where γ is a positive constant.



Left: An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from the previous figure, resulting in a far more appropriate decision rule. Right: An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.

Example: Here we apply SVM and LDA to the Heart data and compare their performances. The aim is to use 13 predictors such as Age, Sex, and Chol in order to predict whether an individual has heart disease. After removing 6 missing observations, the data consist of 297 subjects, which we randomly split into 207 training and 90 test observations.

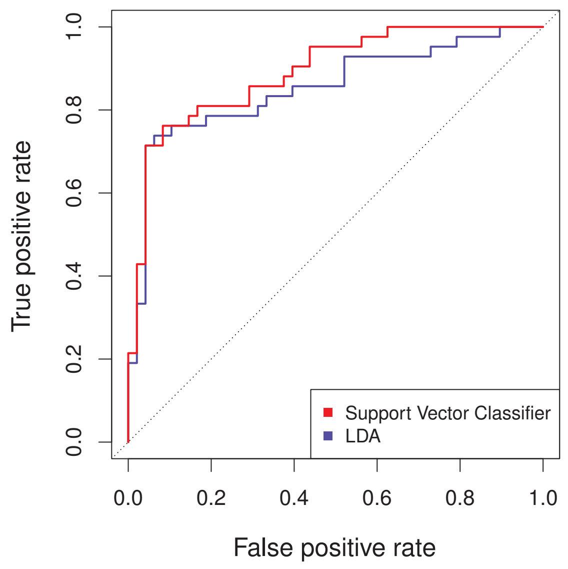


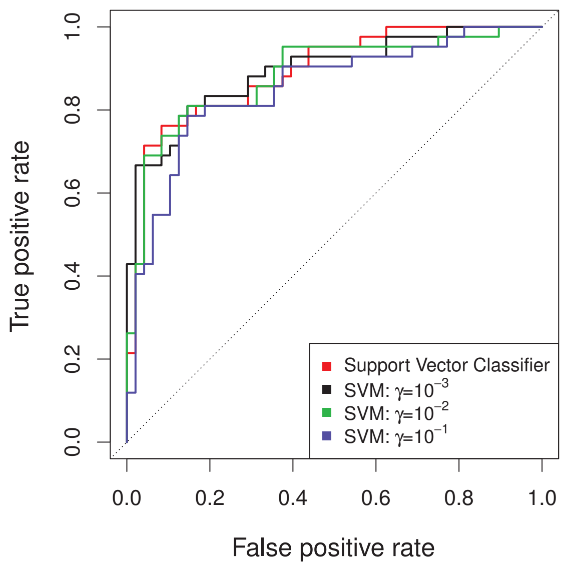
ROC curves for the Heart data training set. Left: The support vector classifier and LDA are compared. Right: The support vector classifier is compared to an SVM using a radial basis kernel with $\gamma = 10^{-3}$, $\gamma = 10^{-2}$, and $\gamma = 10^{-1}$. ROC curve is obtained by changing the threshold 0 to threshold t in $\hat{f}(X) > t$, and recording false positive and true positive rates as t varies.

Note: An optimal classifier will hug the top left corner of the ROC plot.

Note that the support vector classifier is equivalent to a SVM using a polynomial kernel of degree $d=1$.

As γ increases, the SVM fit becomes more non-linear.





ROC curves for the test set of the Heart data. Left: The support vector classifier and LDA are compared. Right: The support vector classifier is compared to an SVM using a radial basis kernel with $\gamma = 10^{-3}$, $\gamma = 10^{-2}$, and $\gamma = 10^{-1}$.

SVMs with More than Two Classes

A number of proposals for extending SVMs to the K-class case, $K > 2$, have been made, the two most popular are the *one-versus-one* and *one-versus-all* approaches.

One-Versus-One Classification:

Fit $\binom{K}{2}$ 2-class SVM classifiers $\hat{f}_{kl}(x)$. Classify x^* to the class that wins the most pairwise competitions.

One-Versus-All Classification:

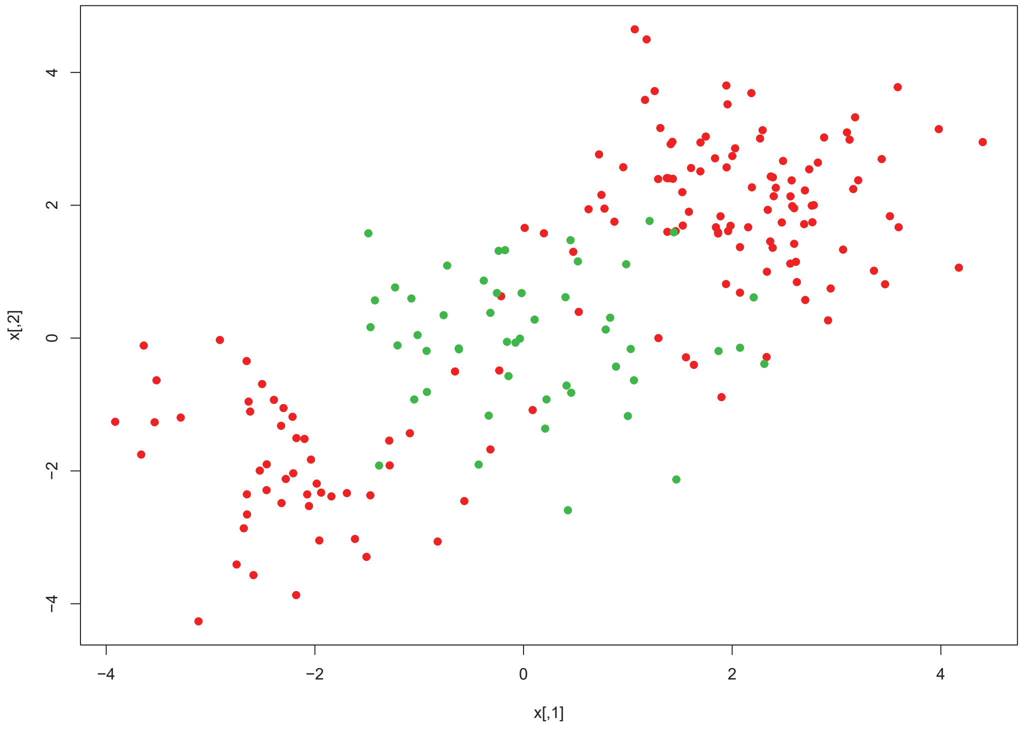
Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

Performing Support Vector Machine in R

In order to fit an SVM using a non-linear kernel, we once again use the svm() function. However, now we use a different value of the parameter kernel.To fit an SVM with a polynomial kernel we use kernel = ”polynomial”, and to fit an SVM with a radial kernel we use kernel = ”radial”. In the former case we also use the degree argument to specify a degree for the polynomial kernel, and in the latter case we use gamma to specify a value of γ for the radial basis kernel.

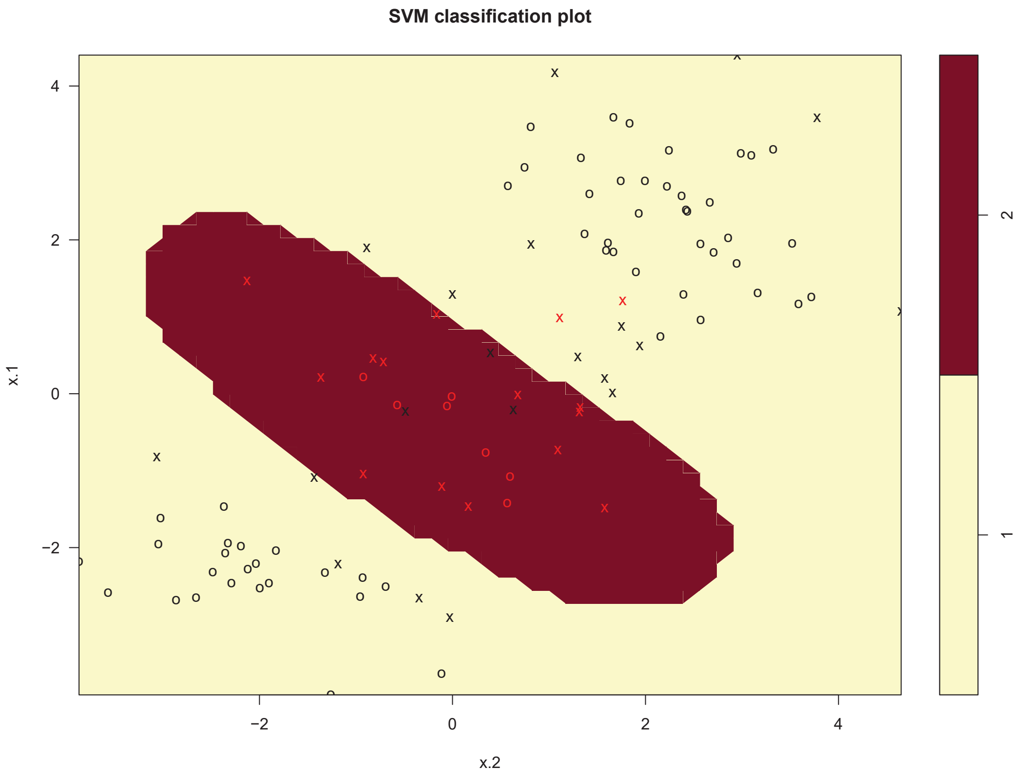
We first generate some data with a non-linear class boundary, as follows:

```
> set.seed(1)
> x <- matrix(rnorm(200 * 2), ncol = 2)
> x[1:100, ] <- x[1:100, ] + 2
> x[101:150, ] <- x[101:150, ] - 2
> y <- c(rep(1, 150), rep(2, 50))
> dat <- data.frame(x = x, y = as.factor(y))
> plot(x, col = y+1, pch = 19)
```



The data is randomly split into training and testing groups. We then fit the training data using the svm() function with a radial kernel and $\gamma = 1$:

```
> set.seed(1)
> train <- sample(200, 100)
> svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1,
    cost = 1)
> plot(svmfit, dat[train, ])
```



The plot shows that the resulting SVM has a decidedly non-linear boundary. The summary() function can be used to obtain some information about the SVM fit:

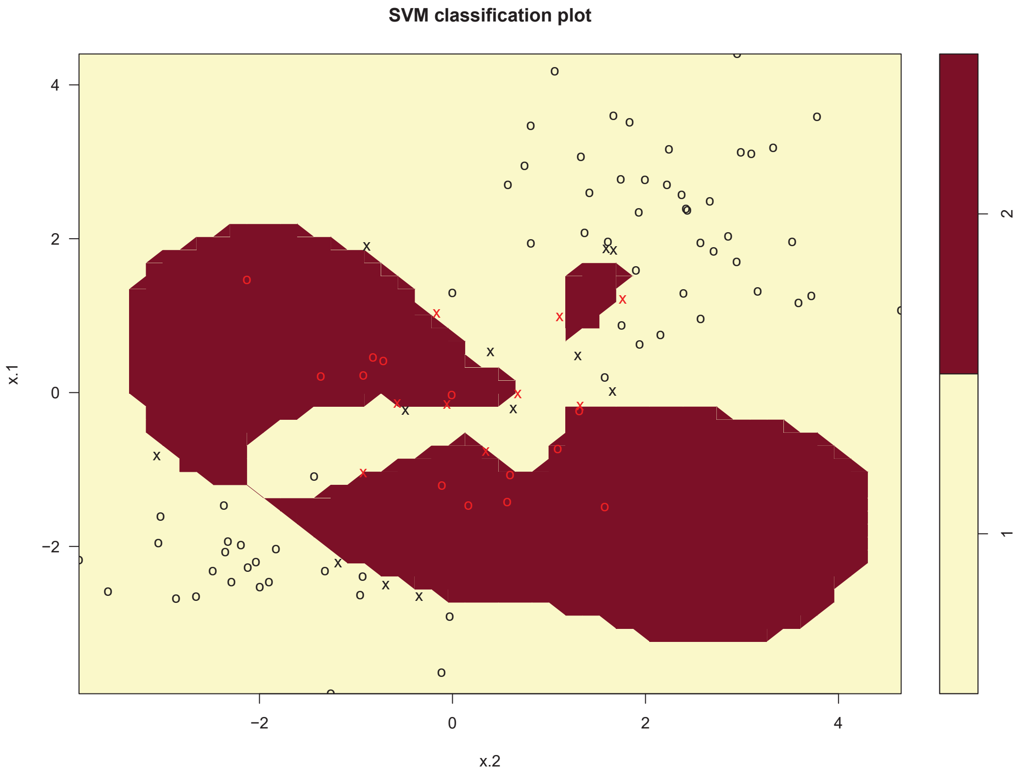
```
> summary(svmfit)

Call:
svm(formula = y ~ ., data = dat[train, ], kernel = "radial",
    gamma = 1, cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  35

 ( 20 15 )


Number of Classes:  2

Levels:
 1 2



> table(predict = svmfit$fitted, truth = dat$y[train])
       truth
predict  1  2
      1 74  2
      2  4 20
```

training err. rate $= \frac{6}{100} = 0.06$

If we increase the value of cost, we can reduce the number of training errors. However, this comes at the price of a more irregular decision boundary that seems to be at risk of overfitting the data.

```
> svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial", gamma = 1,
   cost = 1e5)
> plot(svmfit, dat[train, ])
> table(predict = svmfit$fitted, truth = dat$y[train])
       truth
predict  1  2
      1 78  0
      2  0 22
```

training err. rate $= 0$



We can perform cross-validation using tune() to select the best choice of γ and cost for an SVM with a radial kernel:

```
> set.seed(1)
> tune_out <- tune(svm, y ~ ., data = dat[train, ],
+                      kernel = "radial", ranges = list(
+                      cost = c(0.1, 1, 10, 100, 1000),
+                      gamma = c(0.5, 1, 2, 3, 4)
+                    )
+                 )
> summary(tune_out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation
```