Linear programming III

September 16, 2020

Review

What have covered in previous two classes

- LP problem setup: linear objective function, linear constraints. Optimal solution at exist extreme point(s).
- Simplex method: go through extreme points to find the optimal solution.
- Primal-dual property of the LP problem.
- Interior point algorithm: based on the primal-dual property, travel through the interior of the feasible solution space.
- Quadratic programming: based on KKT condition.
- LP application: quantile regression – minimize the sum of the asymmetric absolute deviations.

Consider usual regression settings with data $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})$ is a p-vector of predictors and y_i is the response for the i^{th} object.

The ordinary linear regression setting is:

- Find coefficient to minimize the residual sum of squares:

$$\hat{\mathbf{b}} = \underset{\mathbf{b}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (y_i - \mathbf{x}_i \mathbf{b})^2 \right\}$$

  Here $\mathbf{b} = (b_1, b_2, \ldots, b_p)^T$ is a vector of coefficients.

- The solution happens to be the MLE assuming a normal model:

$$y_i = \mathbf{x}_i \mathbf{b} + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2)$$

- This is undesirable when the number of predictors (p) is large, because

  1. When $p > n$, there is no degree of freedoms for residual so the model can't be fit.
  2. One wants a small subset of predictors even when $p < n$, but OLS provides an estimated coefficient for each predictor.

The LASSO

LASSO stands for "**Least Absolute Shrinkage and Selection Operator**", which aims for model selection when p is large (works even $p > n$). The LASSO procedure will "shrink" the coefficients toward 0, and eventually force some to be exactly 0 (predictors with 0 coefficient will be selected out).

The LASSO estimates are defined as:

$$\tilde{\mathbf{b}} = \underset{\mathbf{b}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} (y_i - \mathbf{x}_i \mathbf{b})^2 \right\}, s.t. \|\mathbf{b}\|_1 \leq t$$

Here $\|\mathbf{b}\|_1 = \sum_{j=1}^{p} |b_j|$ is the L_1 norm, and $t \geq 0$ is a tuning parameter controlling the strength of shrinkage.

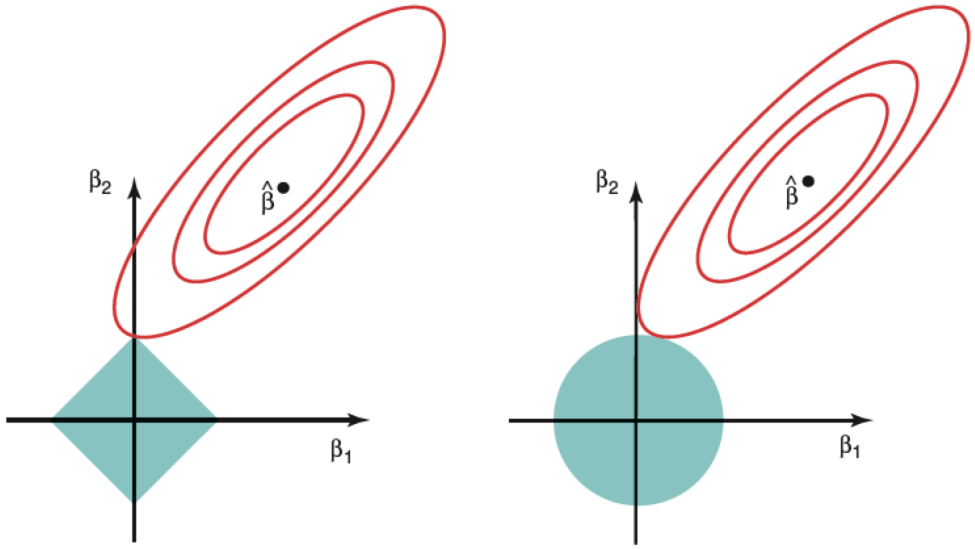
So LASSO tries to minimize the residual sum of square, with a constraint on the sum of the absolute values of the coefficients.

NOTE: There are other types of "regularized" regressions. For example, regression with an $\boldsymbol{L_2}$ penalty, e.g., $\sum_j b_j^2 \leq t$, is called "**ridge regression**".

Model selection by LASSO

The feasible solution space for LASSO is linear (defined by the constraints), so the optimal solution is at a corner point. **The implication**: at optimal, many coefficient (non-basic variables) will be 0 $\Rightarrow$ variable selection.

On the contrary, ridge regression usually doesn't have any coefficient being 0, so it doesn't do model selection.



LASSO model fitting

The LASSO problem can be solved by standard **quadratic programming algorithm**.

In LASSO, we need to solve the following optimization problem:

$$max \quad -\sum_{i=1}^{n}(y_i - \sum_j b_j x_j)^2$$
$$s.t. \quad \sum_j |b_j| \leq t$$

This is not a standard LP form, since the constraints have absolute value operator!

LASSO model fitting (cont.)

The trick is to convert the problem into the standard QP problem setting, i.e., to remove the absolute value operator.

The trick to get rid of absolute value operator is to let $b_j = b_j^+ - b_j^-$, where $b_j^+, b_j^- \geq 0$. Then $|b_j| = b_j^+ + b_j^-$, and the problem can be written as:

$$
\begin{aligned}
max \quad & -\sum_{i=1}^{n}(y_i - \sum_j b_j^+ x_j + \sum_j b_j^- x_j)^2 \\
s.t. \quad & \sum_j (b_j^+ + b_j^-) \leq t, \\
& b_j^+, b_j^- \geq 0
\end{aligned}
$$

This is a standard QP problem can be solved by standard QP solvers.

A little more on LASSO

The Lagrangian for the LASSO optimization problem is:

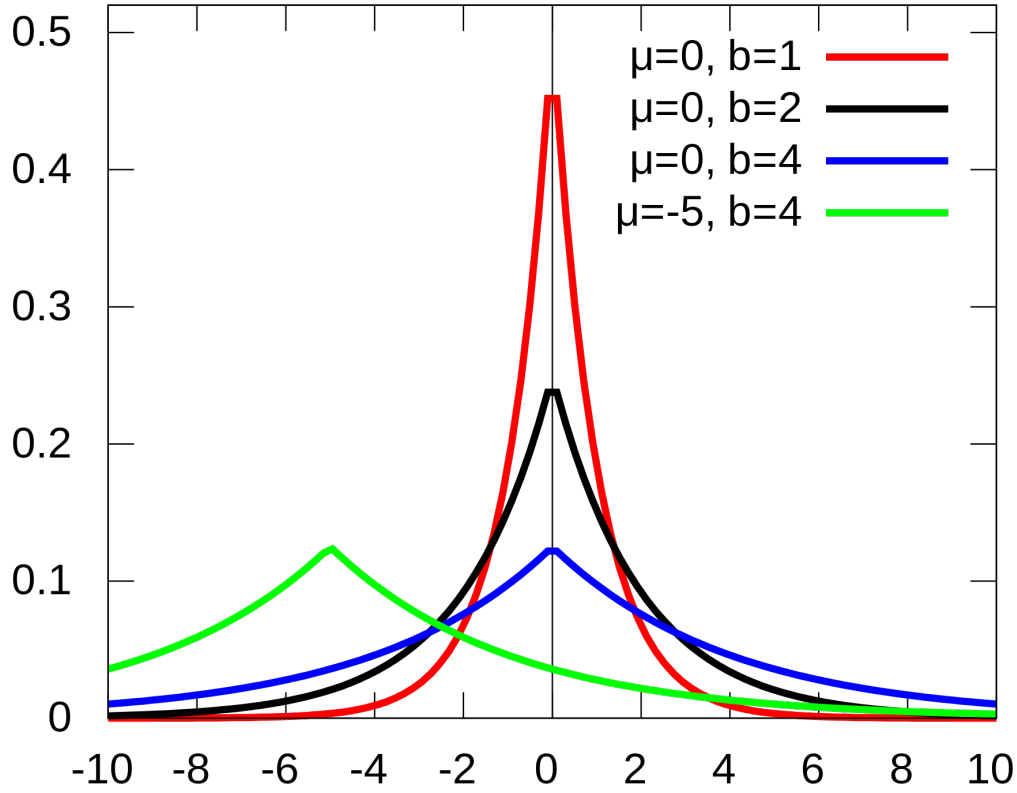
$$L(\mathbf{b}, \lambda) = -\sum_{i=1}^{n}(y_i - \sum_j b_j x_j)^2 - \lambda \sum_{j=1}^{p} |b_j|$$

This is equivalent to the likelihood function of a hierarchical model with a double exponential (DE) prior on b's (remember ADE used in quantile regression?):

$$Y|X, \mathbf{b} \sim N(X\mathbf{b}, \sigma^2)$$
$$b_j \sim DE(1/\lambda)$$

The DE density function is

$$f(x, \tau) = \frac{1}{2\tau} exp\left(\frac{-|x|}{\tau}\right).$$



As a side note, the ridge regression is equivalent with the hierarchical model with a **Normal** prior on $\mathbf{b}$ (verify it).

LASSO in R

The `glmnet` package has function `glmnet`

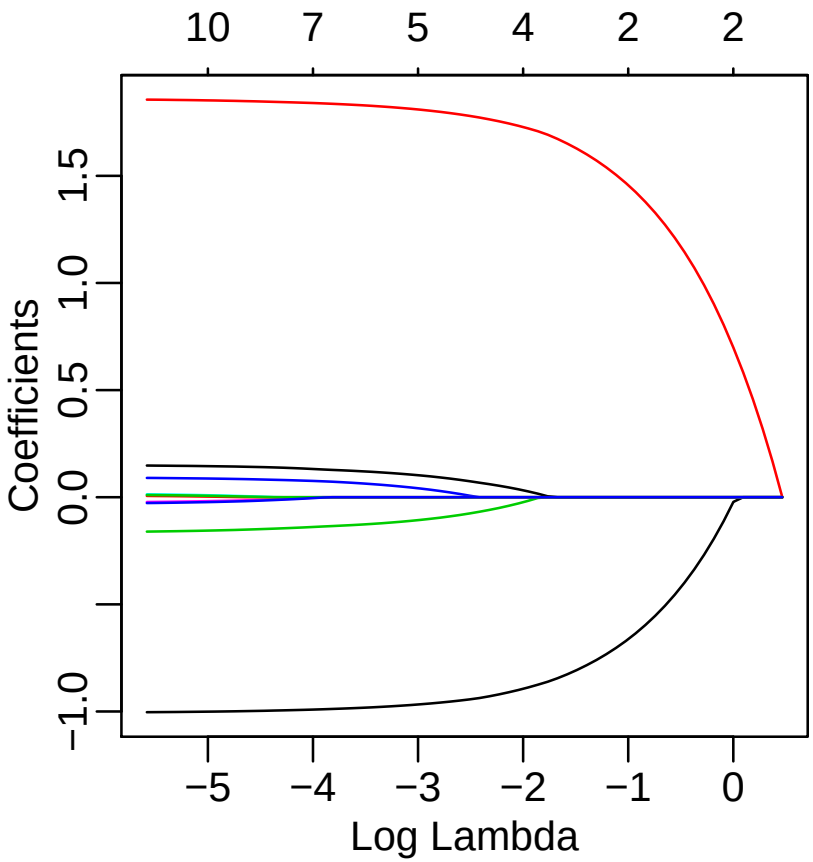
```
glmnet                  package:glmnet                  R Documentation

fit a GLM with lasso or elasticnet regularization

Description:

     Fit a generalized linear model via penalized maximum likelihood.
     The regularization path is computed for the lasso or elasticnet
     penalty at a grid of values for the regularization parameter
     lambda. Can deal with all shapes of data, including very large
     sparse data matrices. Fits linear, logistic and multinomial,
     poisson, and Cox regression models.

Usage:

     glmnet(x, y, family=c("gaussian","binomial","poisson","multinomial","cox","mgaussian"),
         weights, offset=NULL, alpha = 1, nlambda = 100,
         lambda.min.ratio = ifelse(nobs<nvars,0.01,0.0001), lambda=NULL,
         standardize = TRUE, intercept=TRUE, thresh = 1e-07,  dfmax = nvars + 1,
         pmax = min(dfmax * 2+20, nvars), exclude, penalty.factor = rep(1, nvars),
         lower.limits=-Inf, upper.limits=Inf, maxit=100000,
         type.gaussian=ifelse(nvars<500,"covariance","naive"),
         type.logistic=c("Newton","modified.Newton"),
         standardize.response=FALSE, type.multinomial=c("ungrouped","grouped"))
```

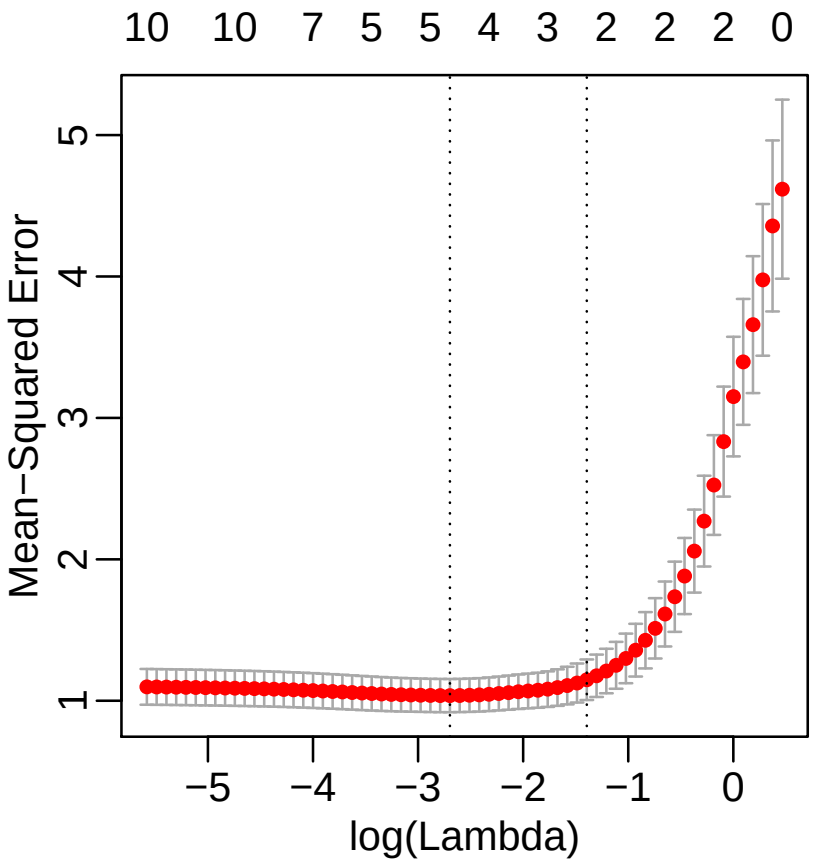
LASSO in R example


```
> x=matrix(rnorm(100*20),100,10)
> b = c(-1, 2)
> y=rnorm(100) + x[,1:2]%*%b
> fit1=glmnet(x,y)
>
> coef(fit1, s=0.05)
11 x 1 sparse Matrix of class "dgCMatrix"
                       1
(Intercept)  0.003020916
V1          -0.967153276
V2           1.809566641
V3          -0.106775004
V4           0.041574896
V5           .
V6           .
V7           0.102566050
V8           .
V9           .
V10          .
```

```
> coef(fit1, s=0.1)
11 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept)  0.01304181
V1          -0.92725224
V2           1.76178647
V3          -0.05743472
V4           .
V5           .
V6           .
V7           0.05953563
V8           .
V9           .
V10          .
> coef(fit1, s=0.5)
11 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept)  0.08689072
V1          -0.52883089
V2           1.29823139
V3           .
V4           .
V5           .
V6           .
.........
```

```
> plot(fit1, "lambda")
#### run cross validation
> cv=cv.glmnet(x,y)
> plot(cv)
```





Support Vector Machine (SVM)

Figures for the slides are obtained from Hastie et al. ***The Elements of Statistical Learning***.

Problem setting:

- Given training data pairs $(x_1, y_1), \ldots, (x_N, y_N)$. x_i's are p-vector predictors. $y_i \in \{-1, 1\}$ are outcomes.
- Our goal: to predict y based on x (find a classifier).
- Such classifier is defined as a function of x, $G(x)$. G is estimated based on the training data (x, y) pairs.
- Once G is obtained, it can be used for future predictions.

There are many ways to construct $G(x)$, and Support Vector Machine (SVM) is one of them. We'll first consider the simple case: $G(x)$ is based on linear function of x. It's often called **linear SVM** or **support vector classifier**.

Simple case: perfectly separable case

- First define a linear hyperplane by $\{x : f(x) = x^T b + b_0 = 0\}$. It is required that b is a unit vector with $\|b\| = 1$ for identifiability.
- A classification rule can be defined as $G(x) = sign[x^T b + b_0]$.
- The problem is to estimate b's.

Consider a simple case where two groups are perfectly separated. We want to find a "border" to separate two groups.

- There are infinite number of borders can perfectly separate two groups. Which one is optimal?
- Conceptually, the optimal border should separates the two classes with the largest margins.
- We define the optimal border to be the one satisfying: (1) the distances between the closest points to the border are the same in both groups, denote the distance by M; and (2) M is maximized.

M is called the "margin".

Problem setup

Then problem to find the best border can be framed into following optimization problem:

$$\begin{aligned} \max_{\beta,\beta_0} \quad & M \\ s.t. \quad & y_i(x_i^T b + b_0) \geq M, i = 1, \ldots, N \end{aligned}$$

This is not a typical LP/QP problem so we do some transformations to make it look more familiar.

Divided both sides of the constraint by M, and define $\beta = b/M$, $\beta_0 = b_0/M$, the constraints become:

$$y_i(x_i^T \beta + \beta_0) \geq 1.$$

This means that we scale the coefficients of the border hyperplane, so that the margin lines are in the forms of $x_i^T \beta + \beta_0 + 1 = 0$ (lower margin) and $x_i^T \beta + \beta_0 - 1 = 0$ (upper margin).

Now we have

$$\|\beta\| = \|b\|/M = 1/M.$$

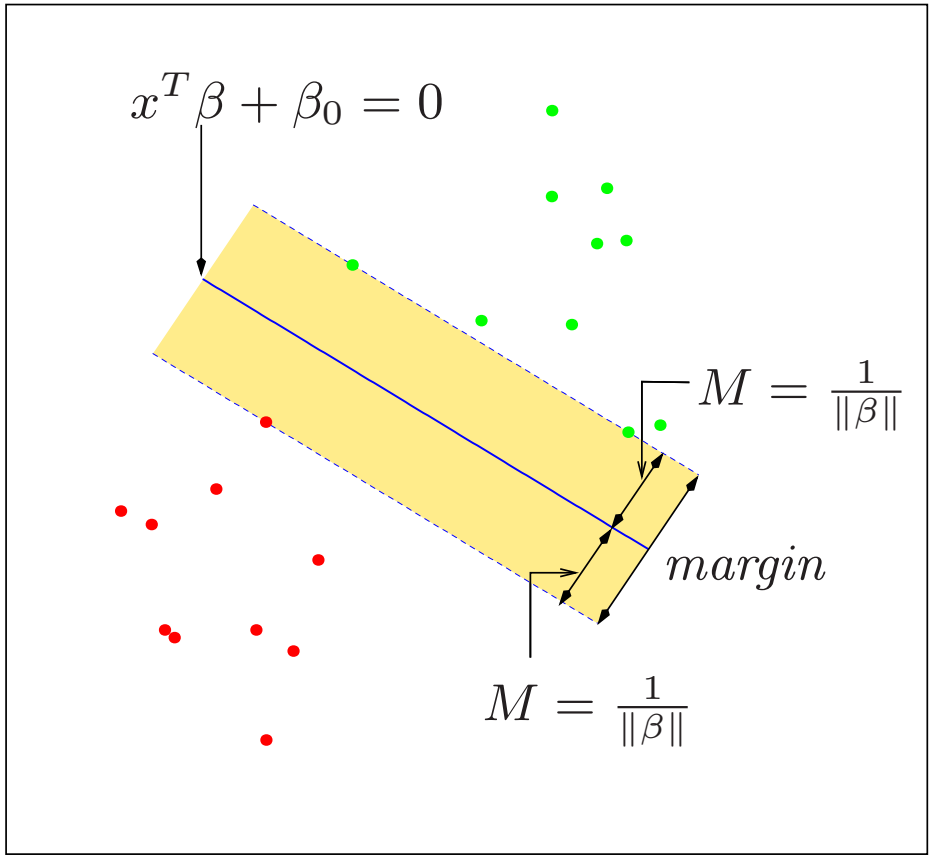
So the objective function (maximizing M) is equivalent to minimizing $\|\beta\|$.

After this transformation, the optimization problem can be expressed as a simpler, more familiar form:

$$\begin{aligned} \min_{\beta,\beta_0} \quad & \|\beta\| \\ s.t. \quad & y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, \ldots, N \end{aligned}$$

This is a typical quadratic programming problem.

Illustration of the optimal border (solid line) with margins (dash lines).



Non-separable case

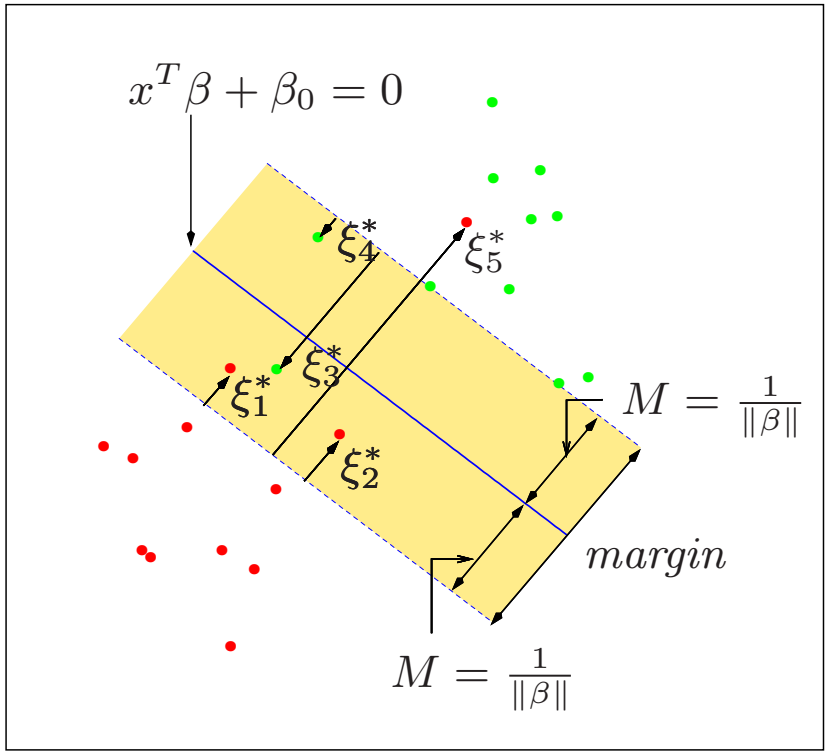
When two classes are not perfectly separable, we still want to find a border with two margins. But now there will be points on the wrong sides. We introduce slack variables to account for those points:

Define slack variables $\{\xi_1, \ldots, \xi_N\}$ (also known as "**hinge loss**") as

$$\xi_i = \max(0, 1 - y_i(x_i^T \beta + \beta_0))$$

We can see that:

- $\xi_i \geq 0 \ \forall i$.
- $\xi = 0$ when the point is on the correct side of the margin.
- ξ is proportional to the distance from the margin: $\xi > 1$ when the point passes the border to the wrong side. $0 < \xi < 1$ when the point is in the margin but still on the correct side.


$x^T\beta + \beta_0 = 0$
ξ_4^*
ξ_5^*
ξ_3^*
ξ_1^*
ξ_2^*
$M = \frac{1}{\|\beta\|}$
$margin$
$M = \frac{1}{\|\beta\|}$

Now the constraints in the original optimization problem is modified to:

$$y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, i = 1, \ldots, N$$

- ξ_i can be interpreted as the proportional amount by which the predication is on the wrong side of the margin.
- We try to minimize the amount of wrong classification, in addition to maximizing the margin. Thus, $\sum_i \xi_i$ is added to the objective function.
- Together, the optimization problem can be formulated as the following quadratic programming problem :

$$\begin{aligned} \min_{\beta,\beta_0} \quad & \frac{1}{2}\|\beta\| + \gamma \sum_i \xi_i \\ s.t. \quad & y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, \quad \xi_i \geq 0 \end{aligned}$$

A bit more about the objective function: When $\|\beta\|$ is small, M will be large (wide margin) and values of ξ_i will be large. The objective function balances the two parts.

Computation

The primal Lagrangian is:

$$L_P = \frac{1}{2}\|\beta\|^2 + \gamma \sum_i \xi_i - \sum_i \alpha_i [y_i(x_i^T \beta + \beta_0) - (1 - \xi_i)] - \sum_i \mu_i \xi_i$$

Take derivatives of β, β_0, ξ_i then set to zero, get (the stationary conditions) :

$$\beta = \sum_i \alpha_i y_i x_i$$

$$0 = \sum_i \alpha_i y_i$$

$$\alpha_i = \gamma - \mu_i, \forall i$$

Plug these back to the primal Lagrangian, get the following dual objective function (verify):

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_{i'} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'}$$

The L_D needs to be maximized subject to constraints:

$$\sum_i \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq \gamma$$

The KKT conditions for the problem (in additional to the stationary conditions) include following complementary slackness and primal/dual feasibilities:

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] = 0$$

$$\mu_i \xi_i = 0$$

$$y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) \geq 0$$

$$\alpha_i, \mu_i, \xi_i \geq 0$$

The QP problem can be solved using interior point method based on these.

Solve for $\hat{\beta}_0$

With $\hat{\alpha}_i$ and $\hat{\beta}$ given, we still need to get $\hat{\beta}_0$ to construct the decision boundary.

One of the complementary slackness condition is:

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] = 0$$

Any point with $\hat{\alpha}_i > 0$ and $\hat{\xi}_i = 0$ (the points on the margins) can be used to solve for $\hat{\beta}_0$.

In practice we often use the average of those to get a stable result for $\hat{\beta}_0$.

The support vectors

At optimal solution, β is in the form of: $\hat{\beta} = \sum_i \hat{\alpha}_i y_i x_i$.

This means $\hat{\beta}$ is a linear combination of $y_i x_i$, and only depends on those data points with $\hat{\alpha} \neq 0$. These data points are called "**support vectors**".

Remember the primal constraint is

$$y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i$$

And according to the complementary slackness in the KKT conditions, at optimal point we have:

$$\alpha_i[y_i(x_i^T \beta + \beta_0) - (1 - \xi_i)] = 0, \quad \forall i$$

which means α_i could be non-zero only when $y_i(x_i^T \beta + \beta_0) - (1 - \xi_i) = 0$.

What does this result tell us?

For points with non-zero α_i:

- The points with $\xi_i = 0$ will have $y_i(x_i^T\beta + \beta_0) = 1$, or these points are on the margin lines.
- Other points with $y_i(x_i^T\beta + \beta_0) = 1 - \xi_i$ are on the wrong side of the margins.

So only the points on the margin or at the wrong side of the margin are informative for the separating hyperplane. These points are called the "**support vectors**", because they provide "support" for the decision boundary.

This makes sense, because the points that can be correctly separated and "far away" from the margin (those "easy" points) don't tell us anything about the classification rule (the hyperplane).

Support Vector Machine

We have discussed **support vector classifier**, which uses hyperplane to separate two groups. **Support Vector Machine** enlarges the feature space to make the procedure more flexible.

To be specific, we transform the input data x_i using some basis functions $h_m(x), m = 1, \ldots, M$. Now the input data become $h(x_i) = (h_1(x_i), \ldots, h_M(x_i))$. This basically transform the data to another space, which could be nonlinear in the original space.

We then find SV classifier in the transformed space using the same procedure, e.g., find optimal

$$\hat{f}(x) = h(x)^T \hat{\beta} + \hat{\beta}_0.$$

And the decision is made by: $\hat{G}(x) = sign(\hat{f}(x))$.

Note: the classifier is linear in the transformed space, but nonlinear in the original one.

Choice of basis function

Now the problem becomes the choice of basis function, or do we even need to choose basis function.

Recall in the linear space, β is in the form of:

$$\beta = \sum_i \alpha_i y_i x_i.$$

In the transformed space, it becomes:

$$\beta = \sum_i \alpha_i y_i h(x_i).$$

So the decision boundary is:

$$f(x) = h(x)^T \sum_i \alpha_i y_i h(x_i) + \beta_0 = \sum_i \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0.$$

Moreover, the dual objective function in transformed space becomes:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_{i'} \alpha_i \alpha_{i'} y_i y_{i'} \langle h(x_i), h(x_{i'}) \rangle$$

What does this tell us?

Both the objective function and the decision boundary in the transformed space involves only the inner products of the transformed data, not the transformation itself!

So the basis functions are not important, as long as we know $\langle h(x), h(x_i) \rangle$.

Kernel tricks

Define the kernal function $K : \mathbb{R}^P \times \mathbb{R}^P \to \mathbb{R}$, to represent the inner product in the transformed space:

$$K(x, x') = \langle h(x), h(x') \rangle.$$

K needs to be a symmetric and positive semi-definite. With the kernel trick, the decision boundary becomes:

$$f(x) = \sum_i \alpha_i y_i K(x, x_i) + \beta_0.$$

Some popular choices of the kernel functions are:

- Polynomial with d degree: $K(x, x') = (a_0 + a_1 \langle x, x' \rangle)^d$.
- Radial basis function (RBF): $K(x, x') = exp\{-\|x - x'\|^2 / c\}$.
- Sigmoid: $K(x, x') = tanh(a_0 + a_1 \langle x, x' \rangle)$.

Computation of SVM

With kernels defined, the Lagrangian dual function is:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_{i'} \alpha_i \alpha_{i'} y_i y_{i'} K(x_i, x_{i'})$$

Maximize L_D, with α_i's being the unknowns, subject to the same constrains:

$$\sum_i \alpha_i y_i = 0$$

$$0 < \alpha_i < \gamma$$

This is a standard QP problem can be solved easily.

The role of γ

To control the smoothness of boundary.

- γ is the tuning parameter for $\sum_i \xi_i$ in the objective function.
- It is introduced to control the total misclassification.
- we can always project the original data to higher dimensional space so that they can be better separated by a linear classifier (in the transformed space), but
  - Large γ: fewer error in transformed space, wiggly boundary in original space.
  - Small γ: more errors in transform space, smoother boundary in original space.

γ is a tuning parameter often obtained from cross-validation.

A little more about the decision rule

Recall that the decision boundary only depends on support vectors, or the points with $\alpha_i \neq 0$. So $f(x)$ can be written as:

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

where S is the set of support vectors.

The kernel $K(x, x')$ can be seen as a similarity measure between x and x'. So to classify for point x, the decision is made essentially by a weighted sum of similarity of x to all the support vectors.

SVM in R

There are several `R` packages include SVM function: `e1071`, `kernlab`, `klaR`, `svmpath`, etc.

Table below summarize the R SVM functions. For more details please refer to the "*Support Vector Machines in R*" paper at the class website.

	`ksvm()` (**kernlab**)	`svm()` (**e1071**)	`svmlight()` (**klaR**)	`svmpath()` (**svmpath**)
Formulations	C-SVC, ν-SVC, C-BSVC, spoc-SVC, one-SVC, ϵ-SVR, ν-SVR, ϵ-BSVR	C-SVC, ν-SVC, one-SVC, ϵ-SVR, ν-SVR	C-SVC, ϵ-SVR	binary C-SVC
Kernels	Gaussian, polynomial, linear, sigmoid, Laplace, Bessel, Anova, Spline	Gaussian, polynomial, linear, sigmoid	Gaussian, polynomial, linear, sigmoid	Gaussian, polynomial

Summary of SVM

Strengths of SVM:

- flexibility.
- scales well for high-dimensional data.
- can control complexity and error trade-off explicitly.
- as long as a kernel can be defined, non-traditional (vector) data, like strings, trees can be input.

Weakness:

- how to choose a good kernel (a low degree polynomial or radial basis function can be a good start).