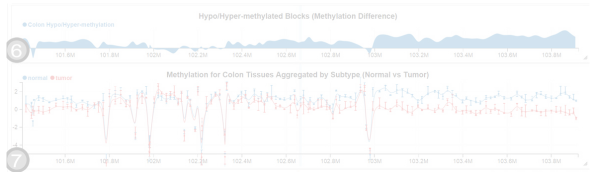


# Kernel Methods

Héctor Corrada Bravo

University of Maryland, College Park, USA

CMSC 643: 2018-10-02

## Support Vector Machines

Let's recall the SVM optimization problem

$$\mathrm{min}_{b,w} \frac{1}{2}|w|^2$$
$$\mathrm{s.t.} y_i(b + w' x_i) \geq 1 \ \forall i$$

This is a *constrained* optimization problem

Minimize the norm of $w$ under the constraint that it classifies every observation correctly (on the proper side of the *margin*).

## Support Vector Machines

We can switch between equivalent constrained minimization and constrained maximization problems.

In the maximum-margin hyper-plane case, the equivalent constrained maximization problem (the *dual* problem) is:

$$\max_\alpha \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i' x_k$$
$$\text{s. t.} \alpha_i \geq 0 \; \forall i$$

## Support Vector Machines

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i' x_k$$
$$\text{s. t.} \alpha_i \geq 0 \; \forall i$$

This quadratic optimization problem is usually easier to optimize than the original problem (notice there is only positivity constraints on $\alpha$).

We can use Projected Gradient Descent, where after each step we ensure that all $\alpha_i \geq 0$ by setting any $\alpha_i < 0$ to 0.

## Support Vector Machines

An important result is then that

*Key insight*: **SVMs only depend on pairwise "similarity" functions of observations**

$$\max_\alpha \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i' x_k$$
$$\text{s.t.} \alpha_i = 0 \; \forall i$$

Only inner products between observations are required as opposed to the observations themselves.

## Support Vector Machines

Also, we can write the *discriminant* function in equivalent form

$$f(x) = b + \sum_{i=1}^{n} y_i \alpha_i x' x_i$$

Geometrically, we can think of the inner product between observations as a "similarity" measure.

Therefore, we can fit these models with other measures that works as "similarities".

## Support Vector Machines

This leads to another important point:

*Key insight*: **SVMs only depend on a subset of observations (support vectors)**

Optimal solutions $b$, $w$ and $\alpha$ must satisfy the following condition:

$$\alpha_i[y_i(b + w'x_i) - 1] = 0 \; \forall i.$$

## Support Vector Machines

$$\alpha_i[y_i(b + w'x_i) - 1] = 0 \ \forall i.$$

Case 1: $\alpha_i > 0$, then the signed distance between observation $x_i$ and the decision boundary is 1.

This means that observation $x_i$ is *on the margin*

## Support Vector Machines

$$\alpha_i[y_i(b + w'x_i) - 1] = 0 \ \forall i.$$

Case 2: $y_i(b + w'x_i) > 1$, then observation $x_i$ is not on the margin and $\alpha_i = 0$.

## Support Vector Machines

To define the discriminant function in terms of $\alpha$s we only need observations that are *on the margin*,

i.e., those for which $\alpha_i > 0$.

These are called *support vectors*.

Also implies we only need Support Vectors to make predictions.

## Non-separable data

Let's review the SVM problem for non-separable data:

$$\min_{b,w,\xi} \sum_{i=1}^{N} \xi_i + \frac{\lambda}{2} \|w\|^2$$
$$\text{s. t } y_i(b + w' x_i) \geq 1 - \xi_i \; \forall i$$
$$\xi_i \geq 0 \; \forall i$$

## Non-separable data

An elegant result is that this formulation doesn't change the dual problem we saw before very much:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i' x_k$$
$$\text{s. t. } 0 \leq \alpha_i \leq \frac{1}{\lambda}\ \forall i$$

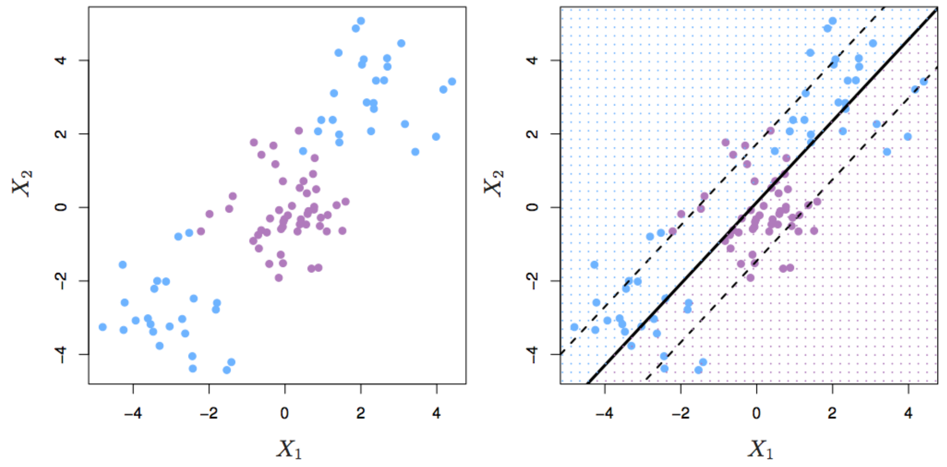
## Non-separable data

Only need support vectors, where $\alpha_i > 0$ to define the discriminant function and make predictions.

The larger the penalty parameter $\lambda$, the learned SVM will have fewer support vectors.

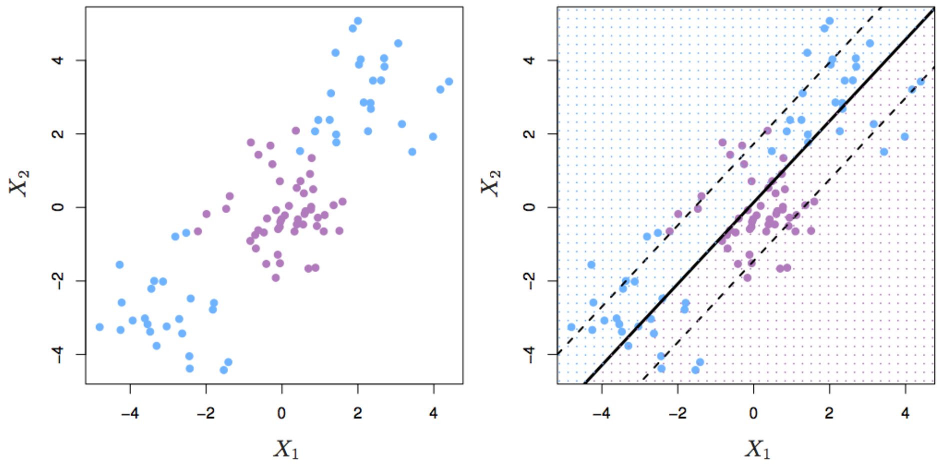
Think of the number of support vectors as a rough measure of the *complexity* of the SVM obtained.

## Non-linear Support Vector Machine



What to do when we need non-linear partitions of predictor space to get a classifier?

## Non-linear Support Vector Machine



Two options:

Construct non-linear features from original features

Kernel methods (in a second)

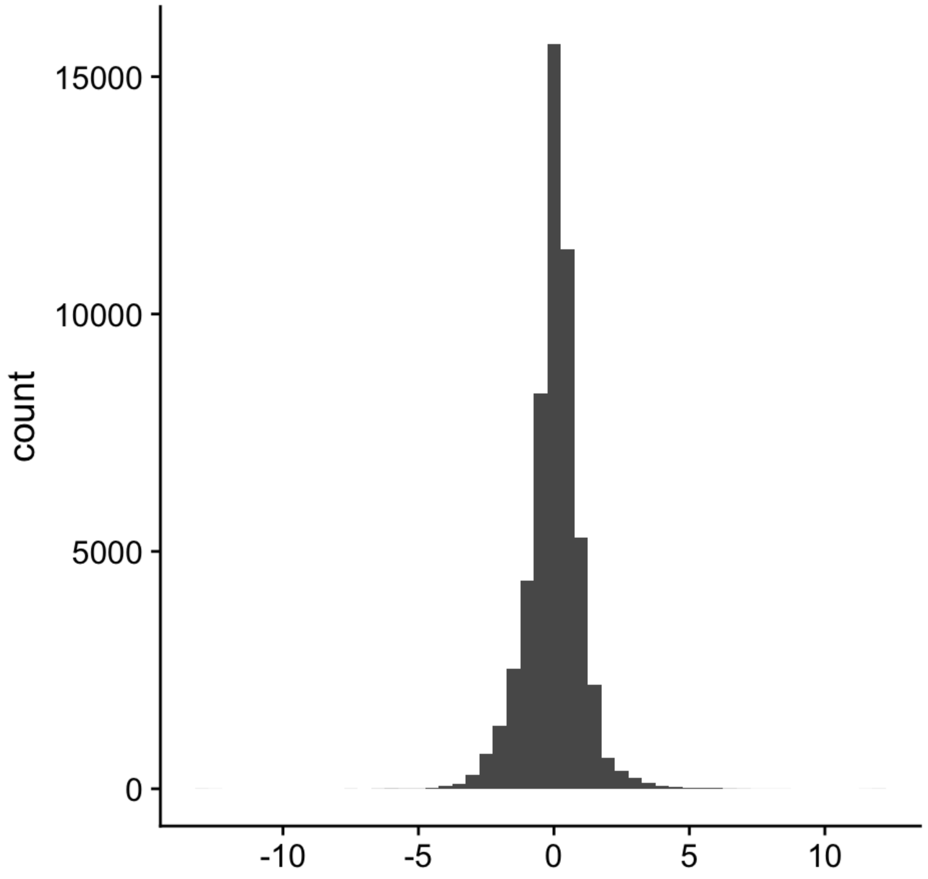
# Tangent: Feature Construction

## Centering and scaling

Given data $x = x_1, x_2, \ldots, x_n$, the transformation applied to obtain centered and scaled variable $z$ is:

$$z_i = \frac{(x_i - \overline{x})}{sd(x)}$$

where $\overline{x}$ is the mean of data $x$, and $sd(x)$ is its standard deviation.



## Tangent: Feature Construction

Another name for this transformation is to *standardize* a variable.

Quiz: What is the mean of $z$? What is it's standard deviation?

$$z_i = \frac{(x_i - \overline{x})}{sd(x)}$$

## Tangent: Feature Construction

One useful result of applying this transformation to variables in a dataset is that all variables are in the same, and thus comparable units.

On occasion, you will have use to apply transformations that only *center* (but not scale) data:

$$z_i = (x_i - \overline{x})$$

Quiz: what is the mean of $z$ in this case? What is it's standard deviation?

## Tangent: Feature Construction

Or, apply transformations that only *scale* (but not center) data:

$$z_i = \frac{x_i}{sd(x_i)}$$

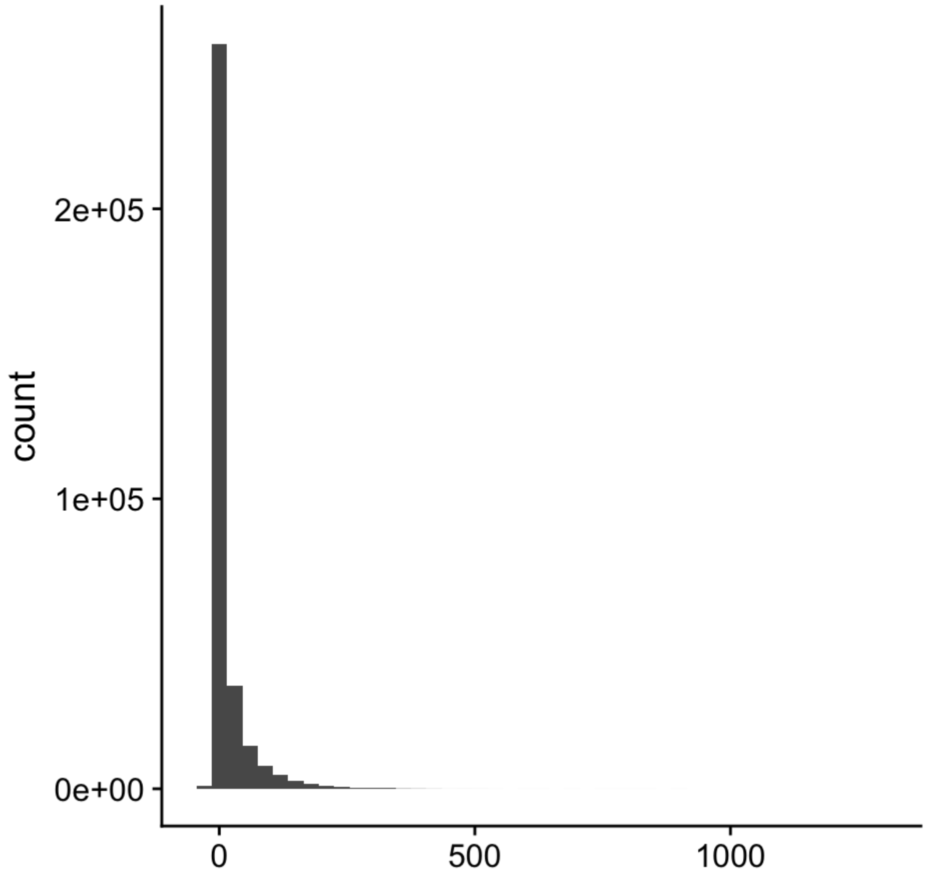
Question: what is the mean of $z$ in this case? What is it's standard deviation?

## Tangent: Feature Construction

### Skewed Data

In many data analysis, variables will have a *skewed* distribution over their range.

Applying a transformation to reduce skew can improve prediction performance.



## Tangent: Feature Construction

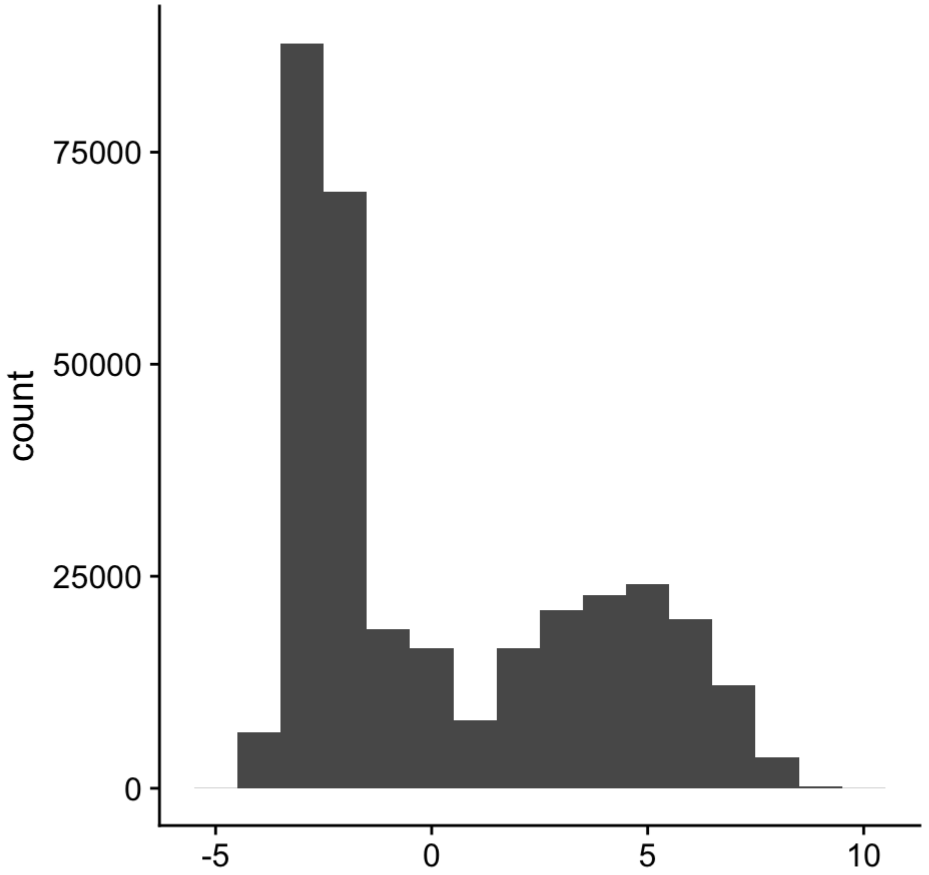
### Skewed data

Logarithmic transform

If some values are negative, two options

Started Log: shift all values so they are positive, apply `log2` Signed Log:

sign(x) × log2(abs(x) + 1).



## Tangent: Feature Construction

### Non-linear Transforms

Besides these "normalizing" transformations, we can construct features to induce non-linearity in our models.

Polynomial transformations: given feature $x_j$, use features $x_j, x_j^2, x_j^3$ in linear model

Interaction features: combine features using products in linear model, e.g., include feature $x_j x_{j'}$, etc.

## Non-linear Support Vector Machine

Kernel Methods provide a different way of doing this.

We can define the SVM discriminant function in terms of inner products of observations.

We can generalize inner product using "kernel" functions that provide something like an inner product:

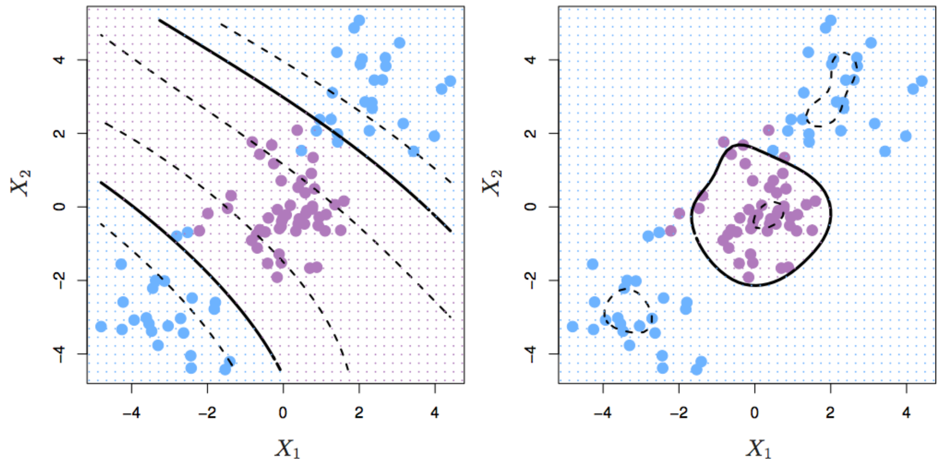
$$f(x) = b + \sum_{i=1}^{n} y_i \alpha_i k(x, x_i)$$

## Non-linear Support Vector Machine

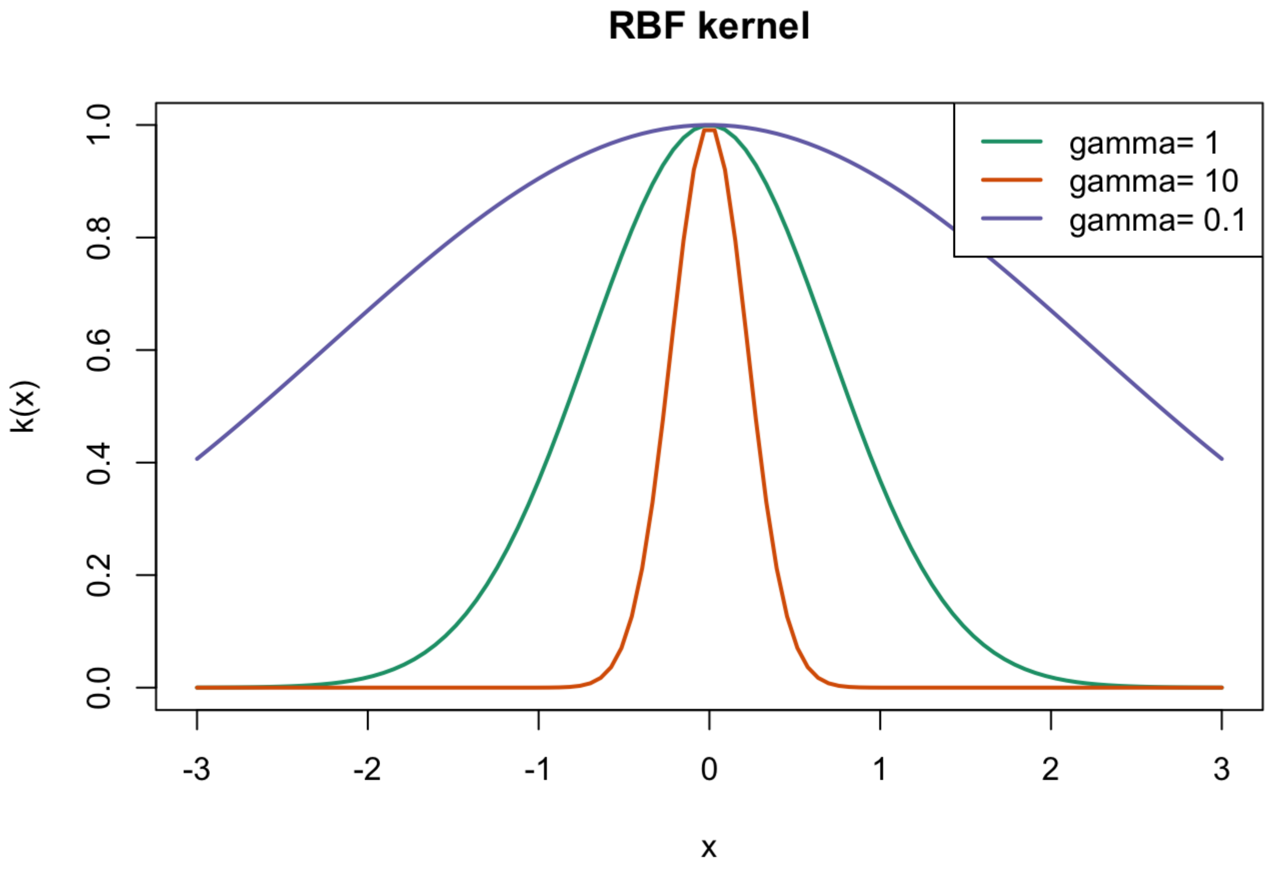
But, what is $k$? Let's consider two examples.

- *Polynomial kernel*: $k(x, x_i) = 1 + \langle x, x_i \rangle^d$
- *RBF (radial) kernel*:

$k(x, x_i) = \exp\{-\gamma \sum_{j=1}^{p} (x_j - x_{ij})^2\}$



## Non-linear Support Vector Machine



## Non-Linear Support Vector Machine

*Quiz* Show that using $d = 2$ with a polynomial kernel is equivalent to using a quadratic transformation on the input features

## Non-linear Support Vector Machine

The optimization problem is very similar

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k k(x_i, x_k)$$

$$\text{s.t. } 0 \leq \alpha_i \leq \frac{1}{\lambda} \; \forall i$$

## SVM classification example

Let's try fitting SVMs to the credit card default dataset we saw in previous examples.

Let's start with a linear SVM (where $\mathrm{k}$ is the inner product).

## SVM classification example

Here we are fitting three different SVMs resulting from using three different values of cost parameter $C = \frac{1}{\lambda}$.

| cost | number_svs | train_error | test_error |
|---:|---:|---:|---:|
| 1e-02 | 337 | 3.3 | 3.36 |
| 1e+00 | 338 | 3.3 | 3.36 |
| 1e+02 | 341 | 3.3 | 3.36 |

## SVM classification example

Let's try now a *non-linear* SVM by using a radial kernel.

Notice now that we have two parameters to provide to the fitting function: cost parameter $C$ and parameter $\gamma$ of the radial kernel function.

## SVM classification example

| cost | gamma | number_svs | train_error | test_error |
|---:|---:|---:|---:|---:|
| 0.01 | 0.01 | 332 | 3.30 | 3.36 |
| 1.00 | 0.01 | 349 | 3.30 | 3.36 |
| 10.00 | 0.01 | 341 | 3.30 | 3.36 |
| 0.01 | 1.00 | 394 | 3.30 | 3.36 |
| 1.00 | 1.00 | 436 | 2.74 | 2.74 |
| 10.00 | 1.00 | 392 | 2.64 | 2.78 |
| 0.01 | 10.00 | 481 | 3.30 | 3.36 |
| 1.00 | 10.00 | 1133 | 2.44 | 2.92 |

## The SVM as a regularized estimation method

Recall the regularized estimation formulation for the SVM:

$$\min_{b,w} \sum_{i=1}^{N} (1 - y_i f_i)_+ + \frac{\lambda}{2} \|w\|^2$$

If observation $x_i$ is on the proper side of the margin,

then $y_i f_i > 1$ and thus $(1 - y_i fi)_+ = 0$.

Otherwise, $(1 - y_i f_i)_+$ equals the signed distance to the margin for observation $x_i$.

## The SVM as a regularized estimation method

In the non-linear case we can write it as equivalent problem as

$$\min_{b,\alpha} \sum_{i=1}^{N} (1 - y_i f_i)_+ + \frac{\lambda}{2} \alpha' K \alpha$$

## Kernelized Logistic Regression

We can use the same "loss + penalty" formulation to obtain a kernelized version of logistic regression:

$$\min_{\beta_0,\beta} \sum_{i=1}^{N} \log(1 + e^{-y_i f_i}) + \frac{\lambda}{2}\alpha' K\alpha$$

## Kernelized Logistic Regression

As before, function $f$ has a linear expansion in terms of the kernel function:

$$f(x) = b + \sum_{i=1}^{N} \alpha_i k(x, x_i)$$

## Kernelized Logistic Regression

As before, function $f$ has a linear expansion in terms of the kernel function:

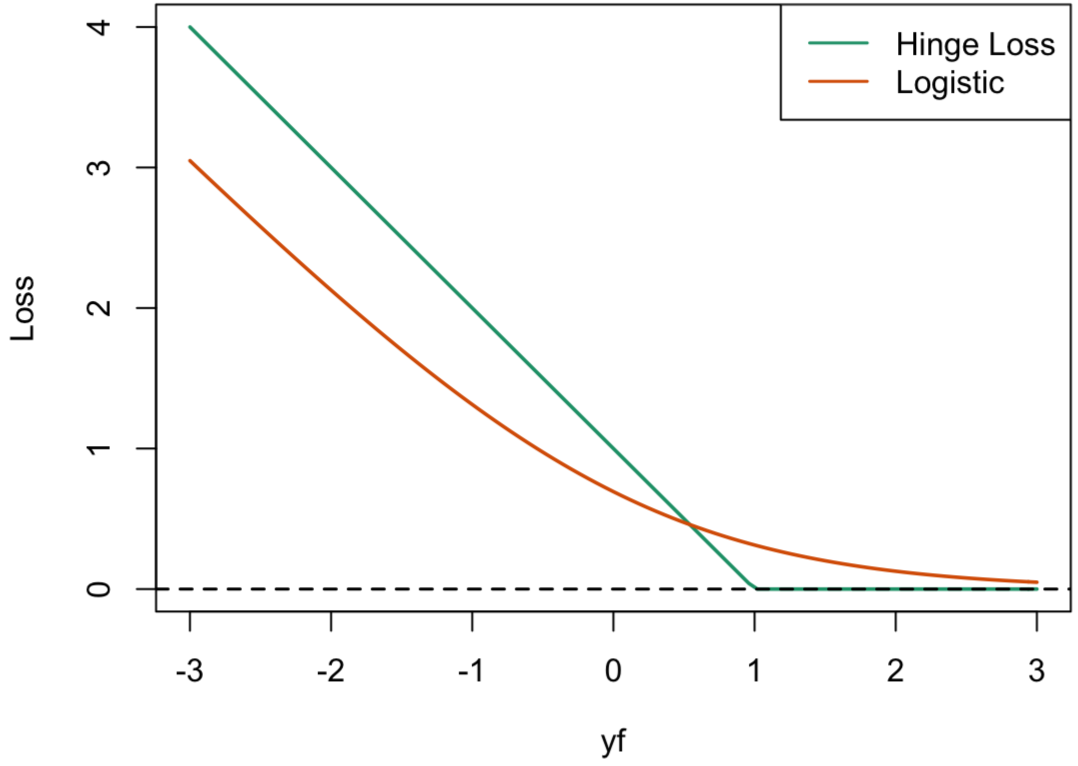
$$f(x) = b + \sum_{i=1}^{N} \alpha_i k(x, x_i)$$

Unlike the SVM, the logistic regression loss function does not tend to set $\alpha_i = 0$ for correctly classified observations.

## Kernelized logistic regression

The loss function in the first term of the formulation is called "Hinge-loss".

We can compare it with the likelihood function for logistic regression.



## Kernelized Logistic Regression

As before, function $f$ has a linear expansion in terms of the kernel function:

$$f(x) = b + \sum_{i=1}^{N} \alpha_i k(x, x_i)$$

Nonetheless, function $f$ retains the interpretation in logistic regression

$$f(x) = \frac{Pr(Y = +1|X = x)}{Pr(Y = -1|X = x)}$$

## Kernelized Regression

In similar fashion, we could build non-linear regression models using the "kernel trick" by using least squares as the loss function when predicting continuous outcomes.

$$\min b, \alpha \sum_{i=1}^{N} (y_i - f_i)^2 + \frac{\lambda}{2} \alpha' K \alpha$$

## Kernelized Regression

Again, function $f$ has a linear expansion in terms of the kernel function

$$f(x) = b + \sum_{i=1}^{N} \alpha_i k(x_i, x)$$

## Kernelized Regression

This does not lead to sparse representations over a subset of observations like SVMs.

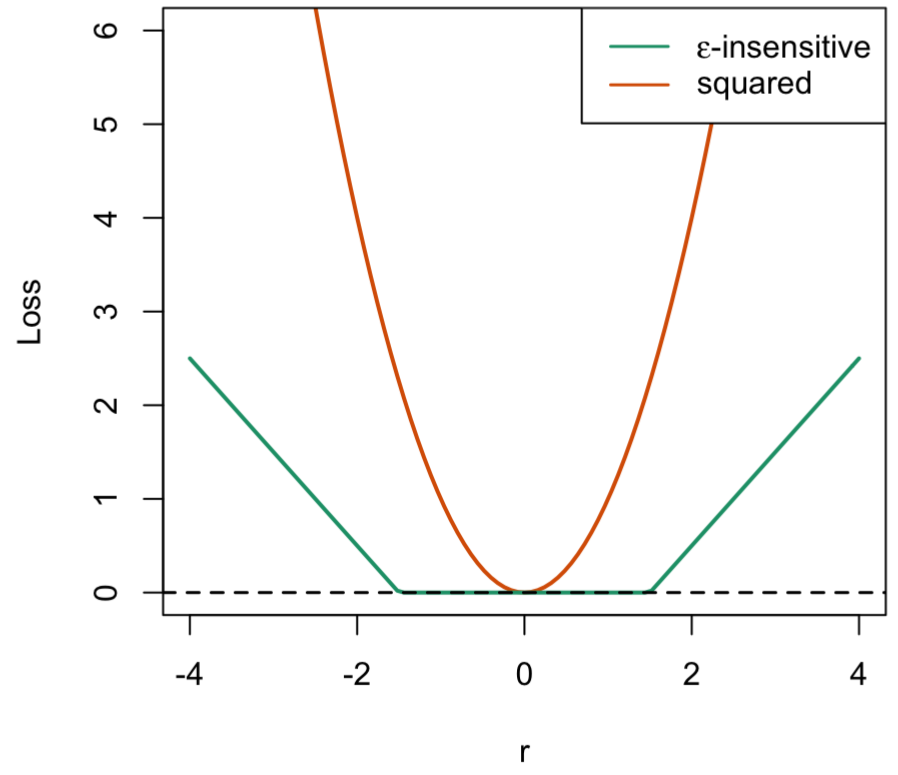
However, a different choice of loss function, similar to hinge loss, can lead to sparse representations.

## Support Vector Regression

Support Vector Regression refers to the "loss + penalty" formulation when $\epsilon$-insensitive loss is used:

$$V_\epsilon(r) = \begin{cases} 0 & \text{if} |r| < \epsilon \\ |r| - \epsilon & \text{otherwise} \end{cases}$$

## Support Vector Regression

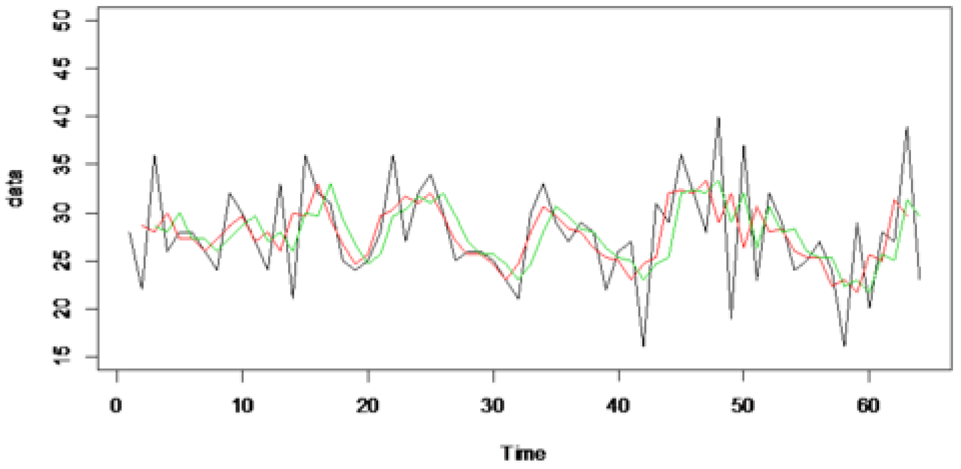


We can compare `(\epsilon)`-insensitive loss to squared loss

## Applications: Modeling labeled data sequences

Consider the case where predictors for observations are structured as sequences.

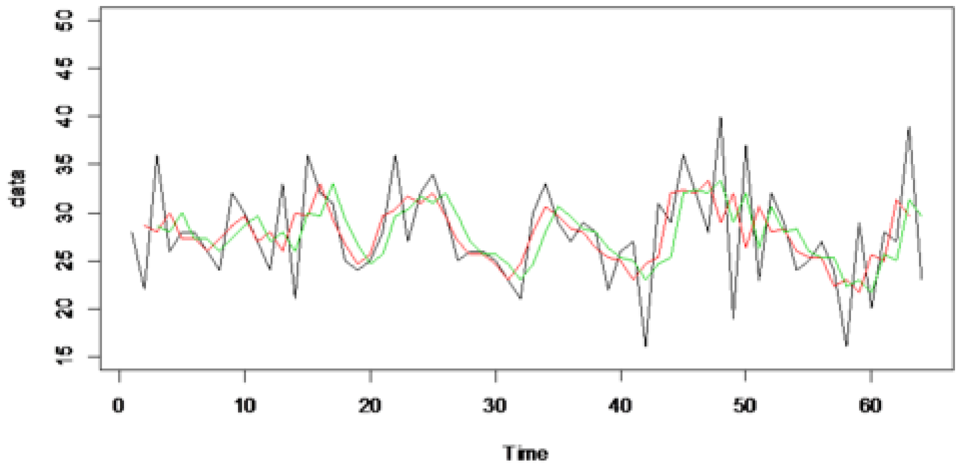
For instance, predictors correspond to some variable measured over time.



## Modeling labeled data sequences

In this case, each observation is represented by a time series

we want to discriminate between time series that belong to two different classes.



## Modeling labeled data sequences

Using the results above, we could model this using a Support Vector Machine, providing a kernel that captures similarity between time series.

Some proposals for this include:

- Autoregressive kernels: Cuturi, Doucet (2011) https://arxiv.org/abs/1101.0673.

The likelihod of a vector autoregressive model is used to create a similarity metric.

## Modeling labeled data sequences

Using the results above, we could model this using a Support Vector Machine, providing a kernel that captures similarity between time series.

Some proposals for this include:

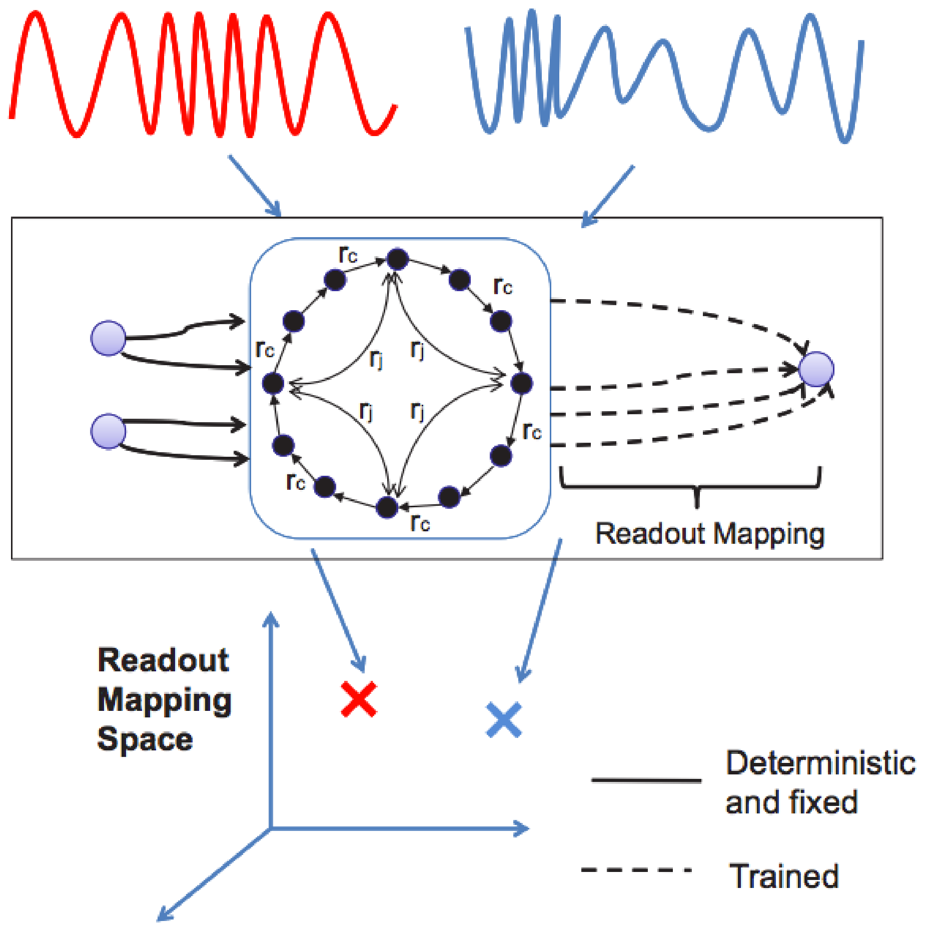
- Dynamic Time Warping Kernel: Shimodaira (2002) https://papers.nips.cc/paper/2131-dynamic-time-alignment-kernel-in-support-vector-machine.pdf.

A warping method is used to define distances between data series

## Modeling labeled data sequences

- Reservoir Computing: Chen et al. http://dl.acm.org/citation.cf id=2487700.

Reservoir state models are used to represent time series and derive kernels



## Structured Output

The "loss + penalty" representation also allows additional flexibility in the types of outcomes that are predicted.

For instance, consider the case where outcomes are numerical vectors

$y_i = (y_{i1}, y_{i2}, \ldots, y_{iT})$ for each observation

along with predictors $x_i$ as before.

## Structured Output

In this case, we would use function $f$ to represent a vector as well:

$$
\begin{aligned}
f(x) = \langle \alpha_{01} + \sum_{i=1}^{N} \alpha_{i1} k(x_i, x) \\
\alpha_{02} + \sum_{i=1}^{N} \alpha_{i2} k(x_i, x) \\
\vdots \\
\alpha_{0T} + \sum_{i=1}^{N} \alpha_{iT} k(x_i, x) \rangle
\end{aligned}
$$

## Summary

The general "loss + penalty" formulation along with kernel methods are capable of capturing a wide array of learning applications.

## Summary

The general "loss + penalty" formulation along with kernel methods are capable of capturing a wide array of learning applications.

A number of effective methods to represent similarities between data series, e.g., time series, as kernel functions allows the usage of this framework to those types of problems.

## Summary

The general "loss + penalty" formulation along with kernel methods are capable of capturing a wide array of learning applications.

A number of effective methods to represent similarities between data series, e.g., time series, as kernel functions allows the usage of this framework to those types of problems.

Structured output formulations are applicable to learn multivariate outcomes with dependency structure between the components of the outcomes.