Lecture 8: Linear Classifiers and More Model Validation

INFO 1998: Introduction to Machine Learning

CDS Education

We explore, learn, and educate big minds.

Hilton

Cornell Data Talks - May 6th 6-7PM EST

Join us to hear about journeys in Data Science with

Hilton Head of Global Advanced Analytics
~ Steven Longstreet

https://tinyurl.com/datatalkscornell

Agenda

1. Perceptron
2. SVM
3. More Cross-Validation techniques

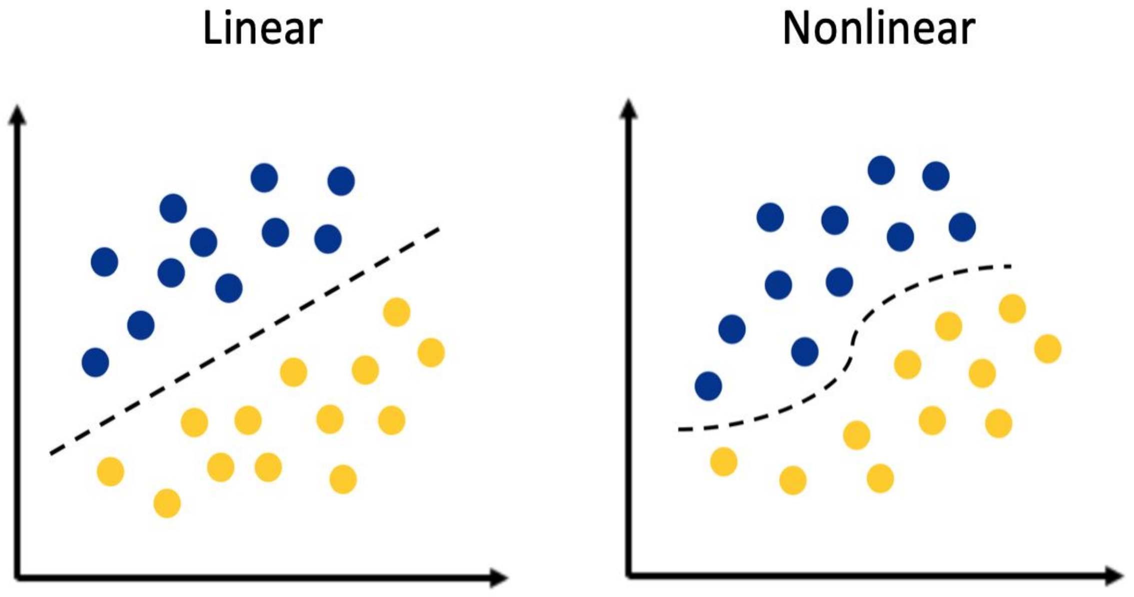
Linear Classifiers

Source

Linear Classifiers

A linear classifier is a hyper plane that is used to classify our data points

A hyperplane is our **decision boundary** and our goal is to find the hyper plane that best classifies our data



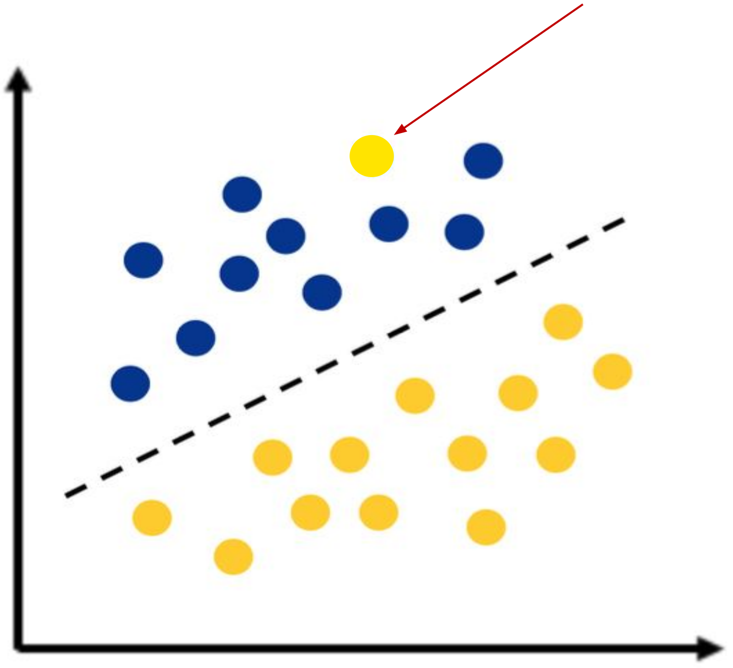
Source

Linearly Separable

In this example, we cannot partition our dataset into yellow and purple with a linear decision boundary. This means that our data is not **linearly separable.**

Outliers are frequently the reason a data set is not linearly separable.

This data set is not linearly separable because of an outlier



Perceptron Learning Algorithm

Goal: find a normal vector w that perfectly classifies all the points in our data set

Algorithm:

```
Initialize classifier as some random hyperplane
While there exists a misclassified point x:
    Tilt classifier slightly so that it classifies x correctly
        (or, is a little closer to classifying x correctly)
End While
```

“Use your mistakes as your stepping stones”

Demo

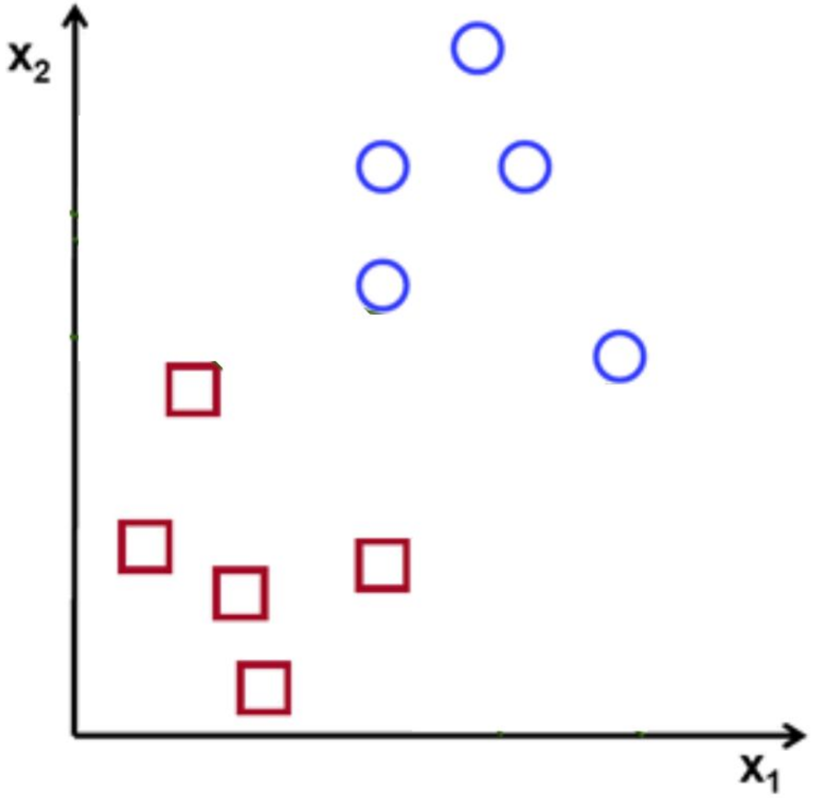
Limitations of Perceptron

The training algorithm will never terminate if your training dataset is not linearly separable ☹️

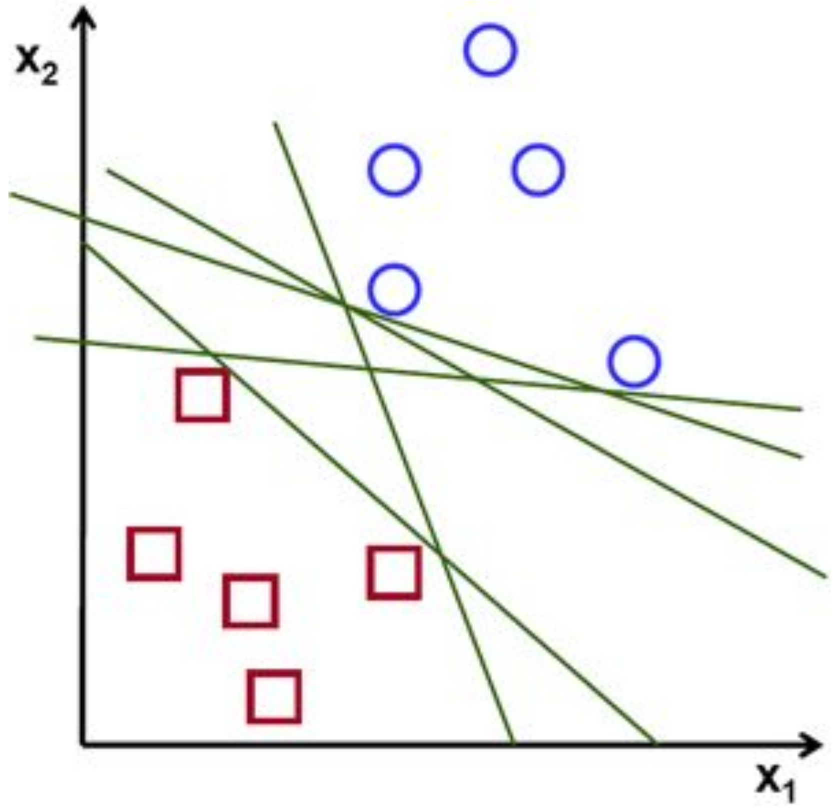
Is a great model to understand the intuition behind the training of a linear classifier: iteratively improve classifier by using misclassified points ☺

SVM

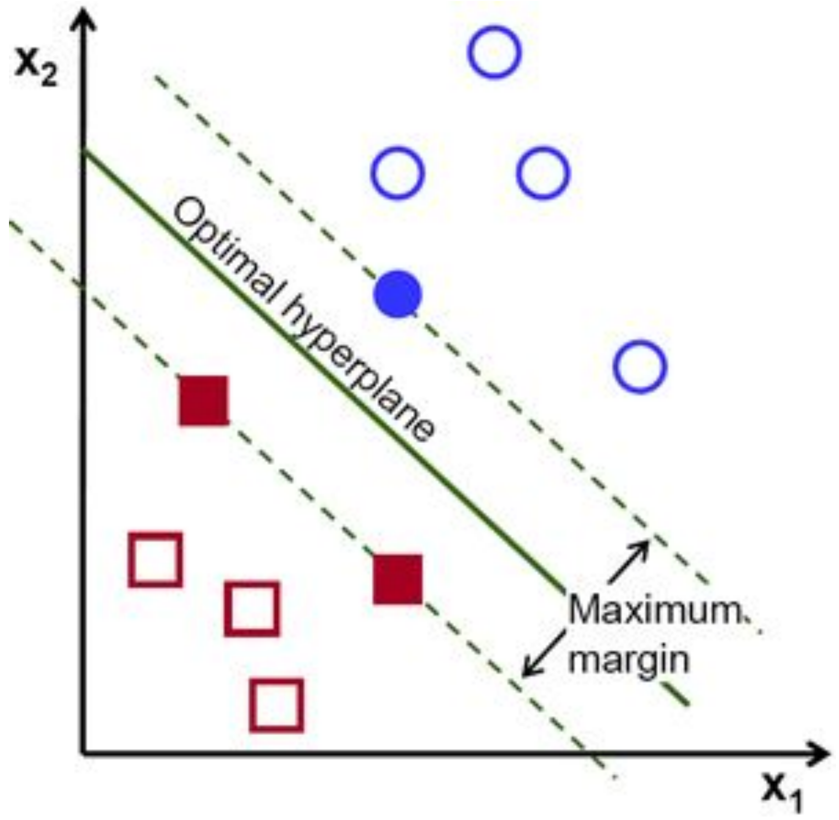
Classify (+) and (-)



Which Hyperplane?



Optimal Hyperplane



Support Vector Machine

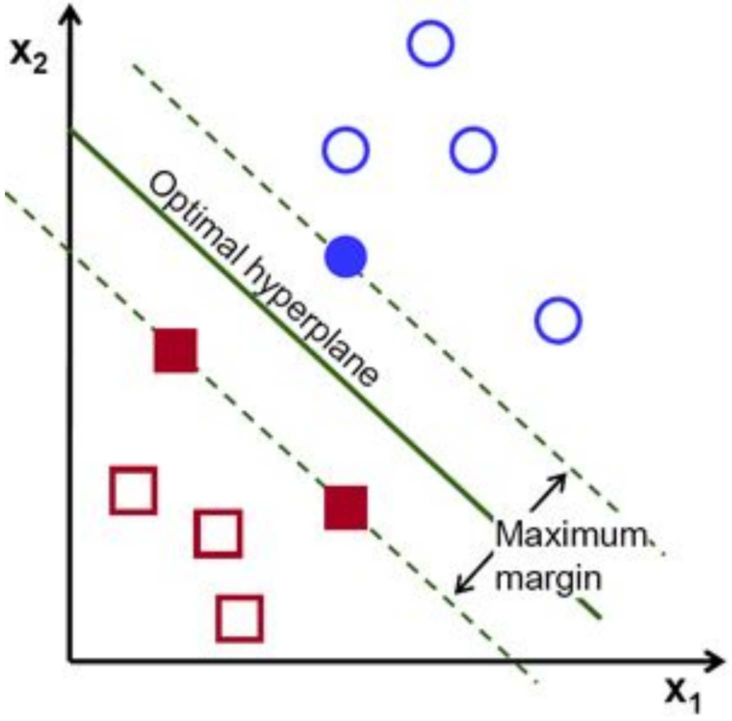
Memory efficient

Effective in a higher dimensions

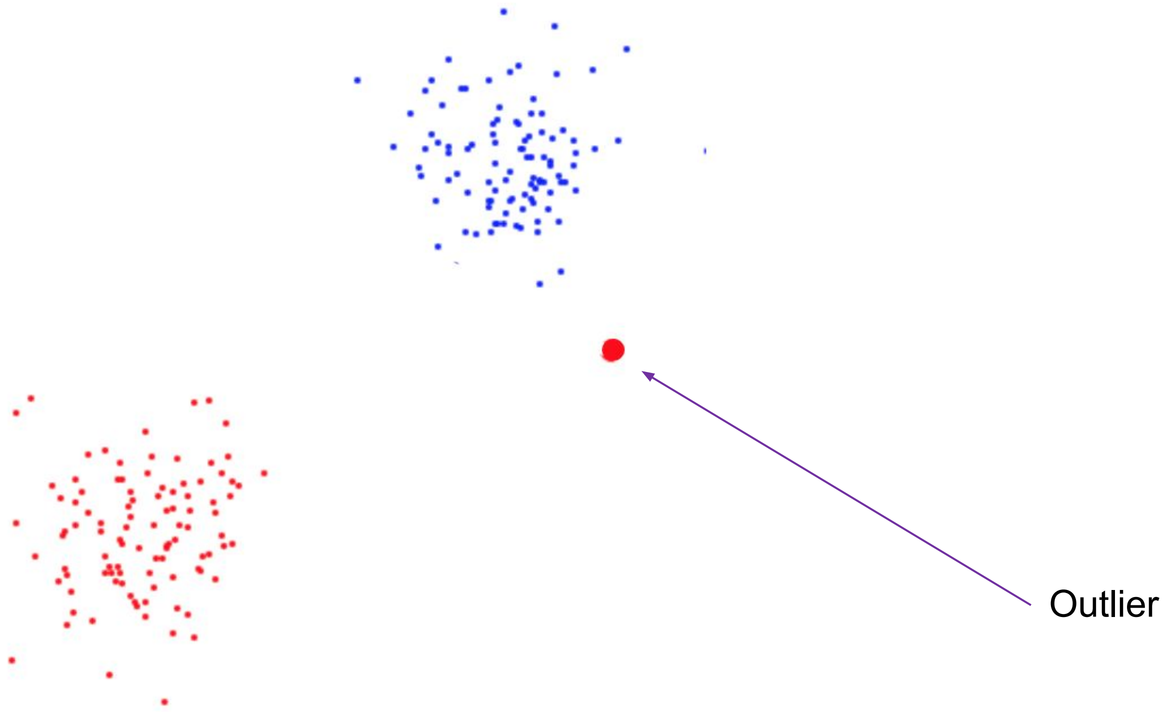
Slow calculation time

Maximal Margin Classifier

- We want to find a **separating hyperplane**
- Once we find candidates for the hyperplane, we try to maximize the **margin**, the normal distance from borderline points
  - Only **Support Vectors** matter

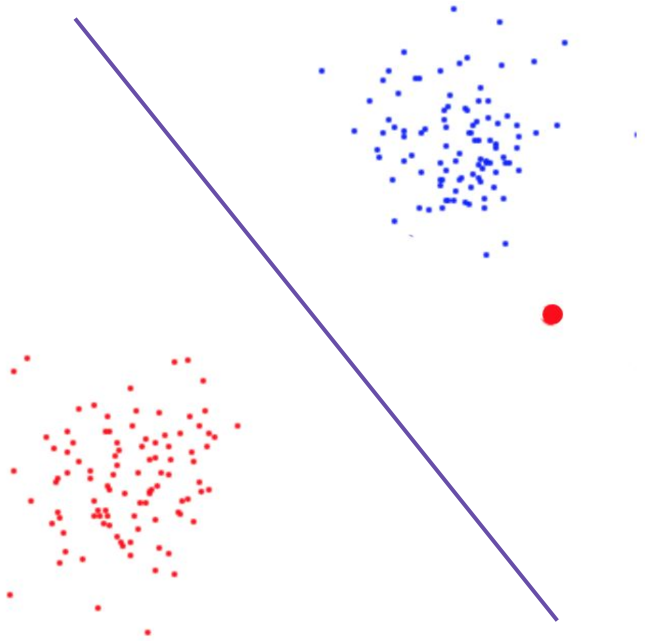


What if…

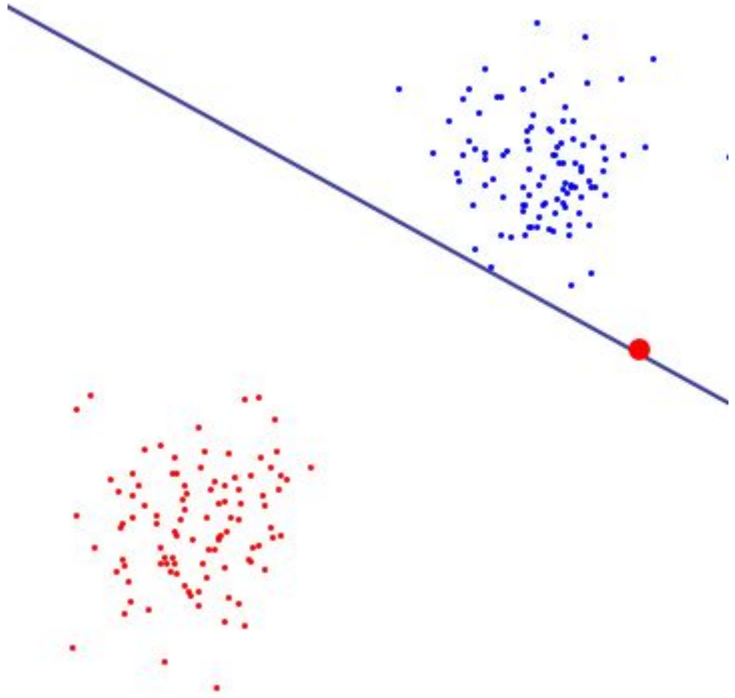


Which Decision Boundary is better?

Boundary 1



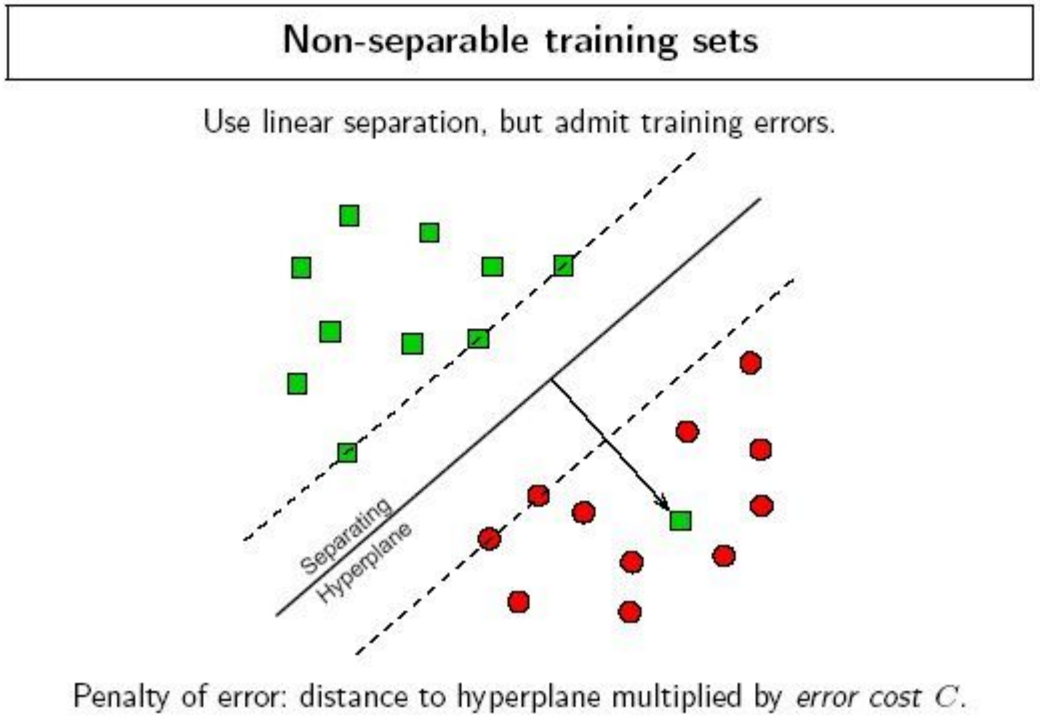
Boundary 2



Margins

Use cost function to penalize misclassified points

Choice of cost function makes margin "hard" vs. "soft"

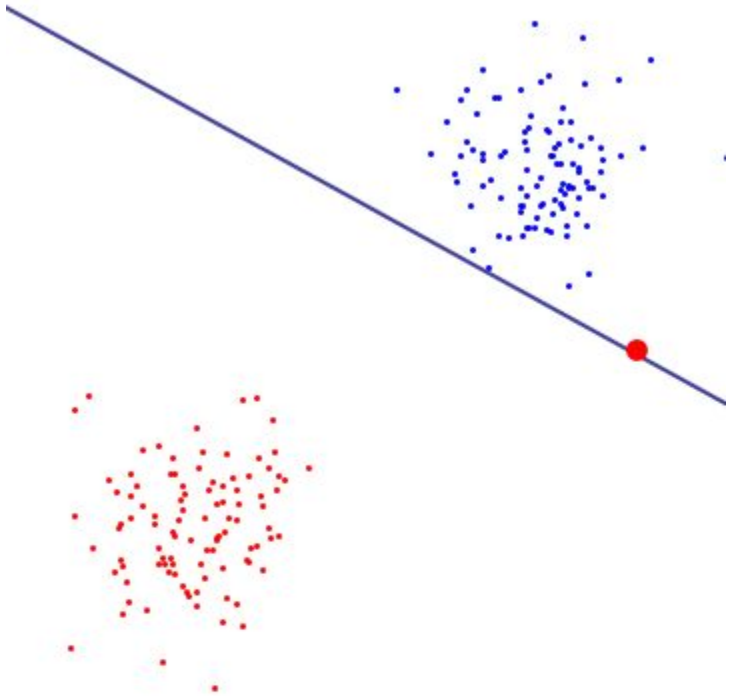

Non-separable training sets

Use linear separation, but admit training errors.

Penalty of error: distance to hyperplane multiplied by *error cost* C.

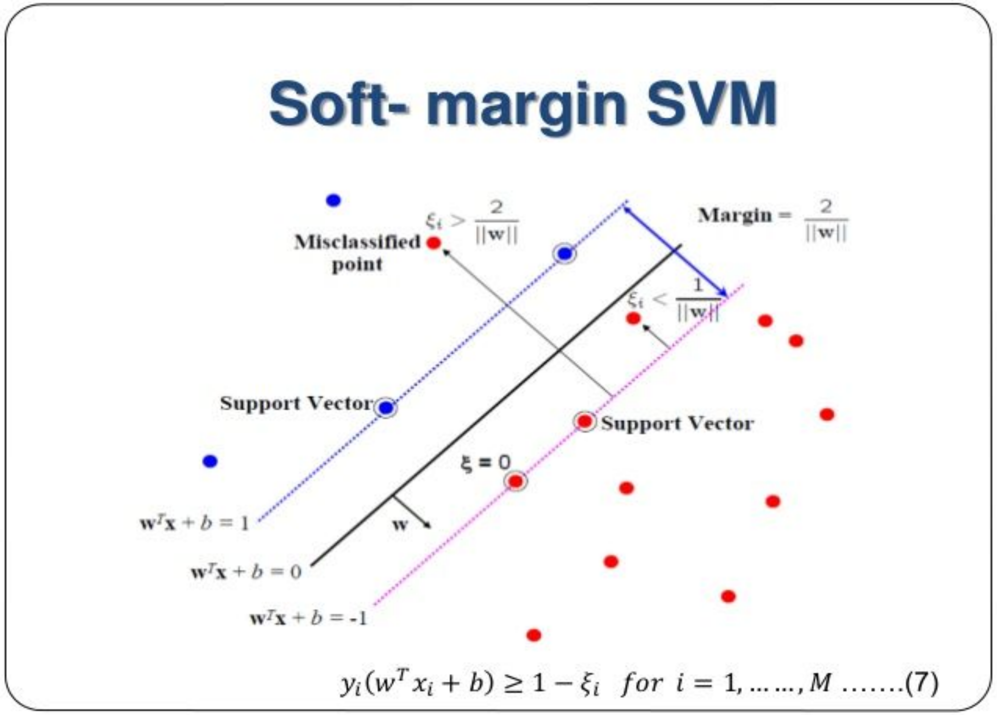
Hard Margins

- High penalty value
- The hyperplane can be dictated by a single outlier



Soft Margins

- Used in non-linearly separable datasets
- Allow for misclassification
- Can account for "dirty" boundaries


Soft- margin SVM
Misclassified point
$\xi_i > \frac{2}{||\mathbf{w}||}$
Margin = $\frac{2}{||\mathbf{w}||}$
$\xi_i < \frac{1}{||\mathbf{w}||}$
Support Vector
Support Vector
$\xi = 0$
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = -1$
$\mathbf{w}$

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad for \ i = 1, \dots\dots, M \ \dots\dots(7)$$

SVM has MANY Hyperparameters

SVM

C

The "penalty cost" for misclassifications (soft margins)

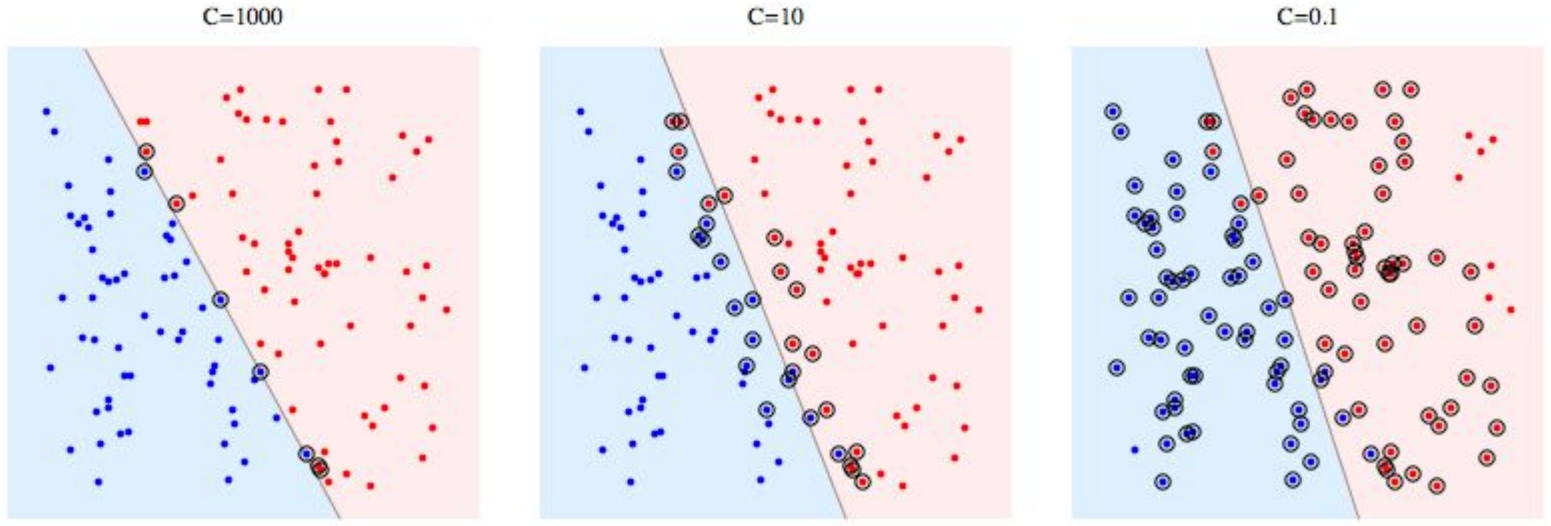
Gamma

How far the influence of a single training example reaches

Kernels

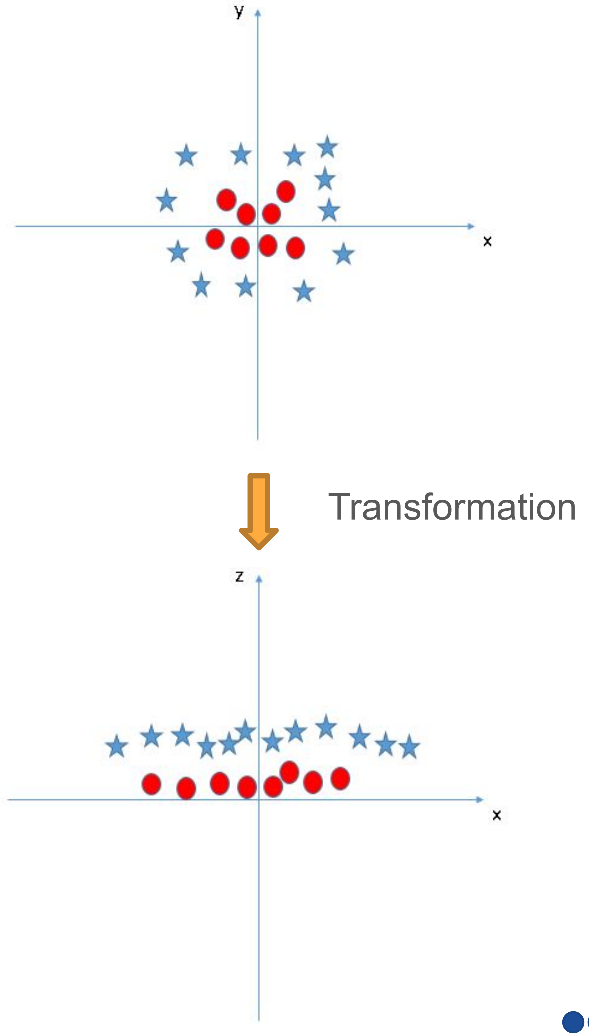
Method of transforming our data set

Misclassification Penalty C



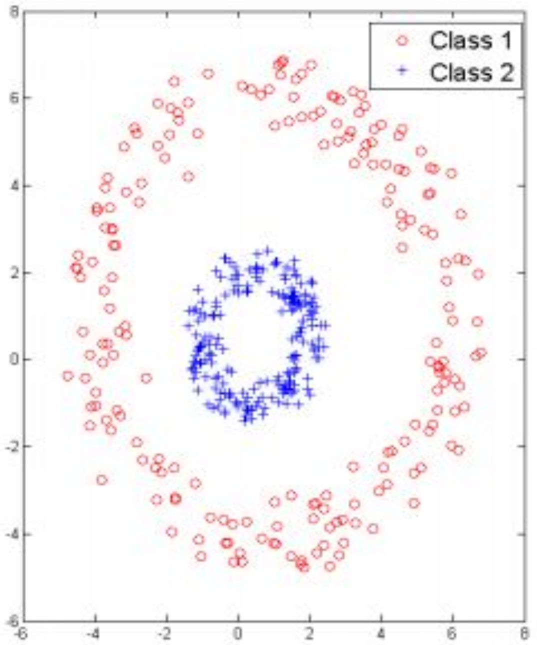
Kernels

- You cannot linearly divide the 2 classes on the *xy* plane at right
- Introduce new feature, $z = x^2 + y^2$ (**radial kernel**)
- Map 2 dimensional data onto 3 dimensional data. Now a hyperplane is easy to find. (Imagine slicing a cone!)

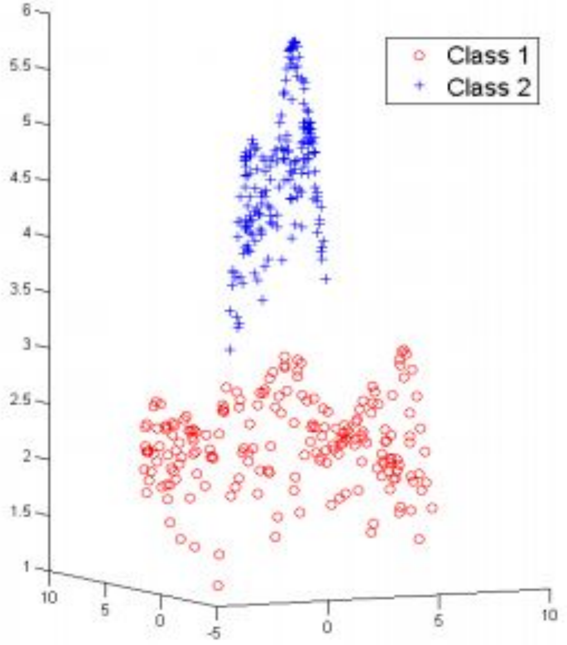


Demo

Kernels

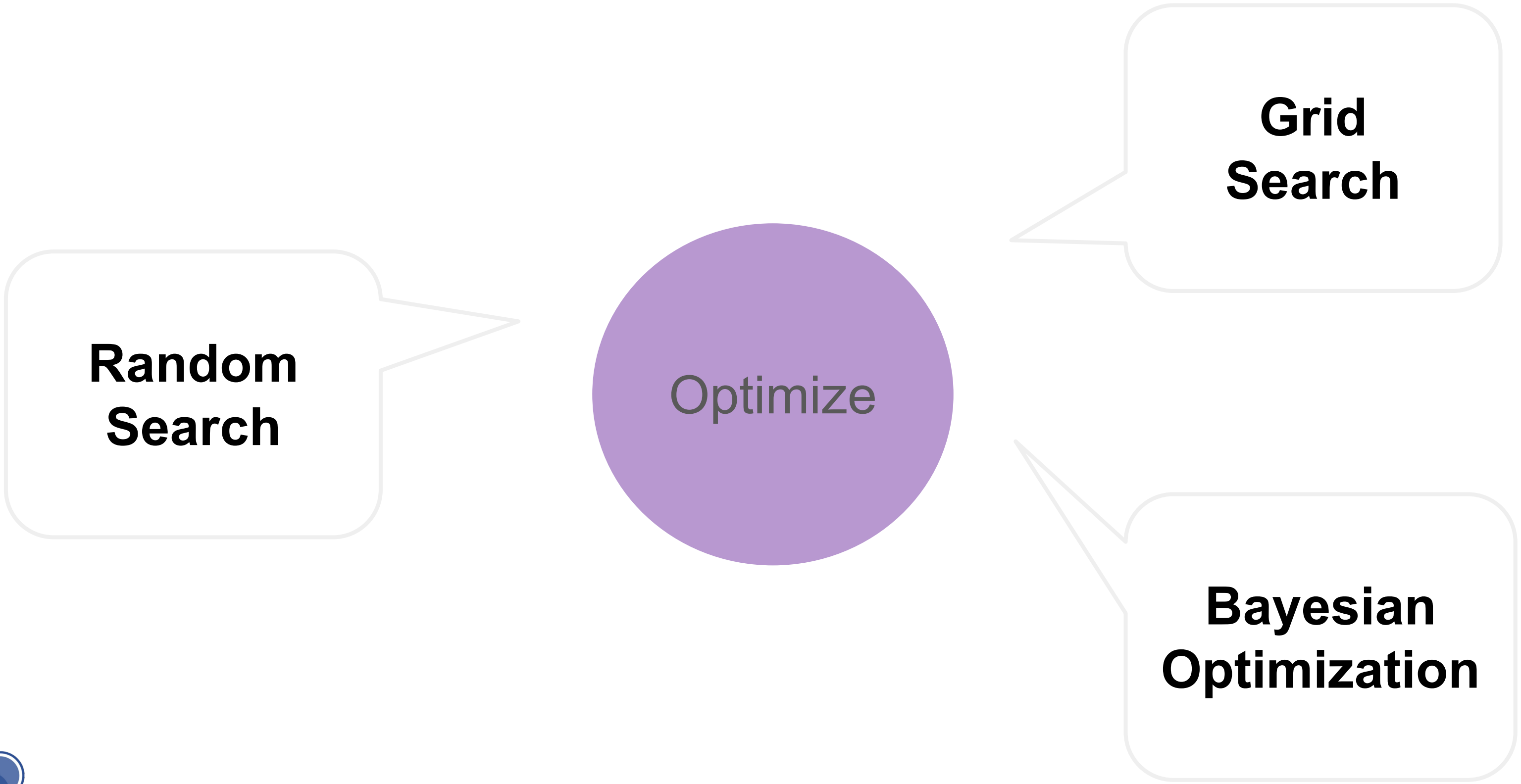


Class 1
Class 2
8
6
4
2
0
-2
-4
-6
-6
-4
-2
0
2
4
6
8



Class 1
Class 2
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
10
5
0
-5
0
5
10

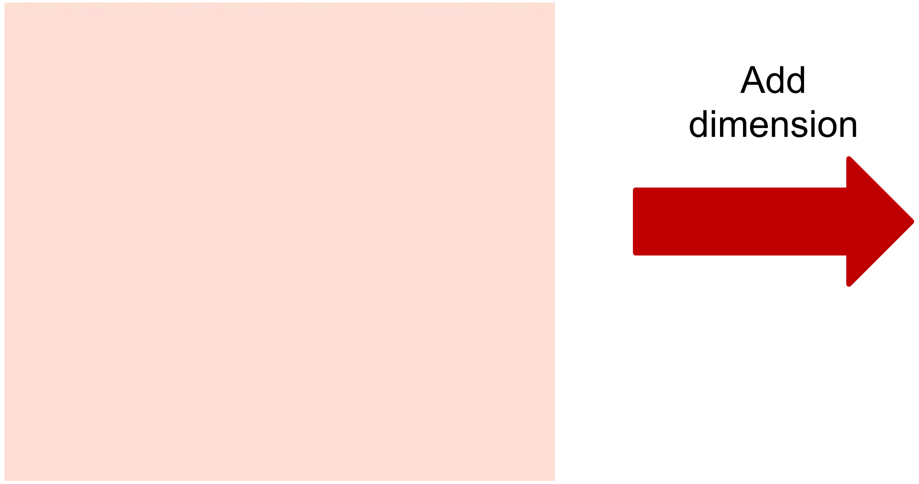
Finding the Best Hyper Parameters

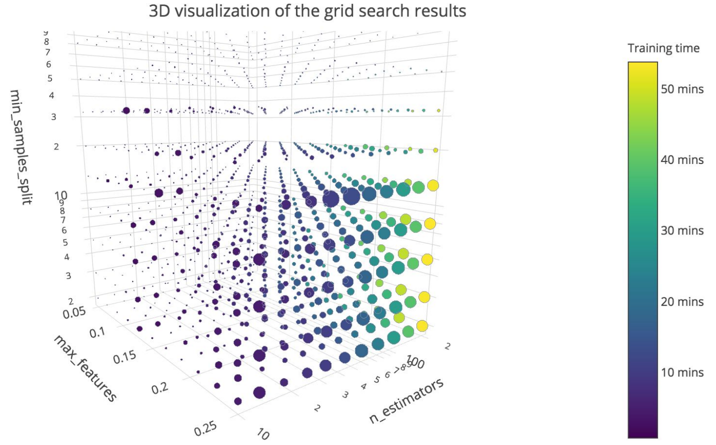


source

Curse of Dimensionality

Our search space for the optimal hyper-parameters increases **exponentially** as the number of hyper parameters we are considering increases





Overview

Perceptron	SVM
• A very simple model • Will perform poorly if data is not linearly separable	• More complex model because we have to choose the "penalty cost" associated with misclassifications • Can transform feature space by choosing a Kernel

Bias-Variance Tradeoff

In regression:

Want to predict y for a fixed test data point x

y is generated from the true model $y = f(x) + \epsilon$
(ϵ has zero mean and independent of everything else)

Estimate using some model $\hat{f}$: $\hat{y} = \hat{f}(x)$

Then:

$$\text{test error} = \mathbb{E}(\hat{f}(x) - y)^2 = \underbrace{\left(\mathbb{E}\hat{f}(x) - f(x)\right)^2}_{\text{bias}^2} + \underbrace{\mathrm{Var}(\hat{f}(x))}_{\text{variance}} + \underbrace{\mathrm{Var}(\epsilon)}_{\text{irreducible error}}$$

Proof (not required, only for reference)

Suppose we have test set data: ***y = f(x) + Ɛ***. The test data *y* is random, since ***Ɛ*** is random. The estimator ***f hat*** is also random, because it is fitted on random training data.
Then from the previous slide "MSE = bias^2 + variance" for unknown quantity ***θ*** and its random estimator ***θ*** hat, we have:

$$
\begin{aligned}
&\mathbb{E}(\hat{f}(x) - y)^2 \\
=&\mathbb{E}\left[\mathbb{E}\left[(\hat{f}(x) - y)^2 | y\right]\right] \\
=&\mathbb{E}\left[(\mathbb{E}\hat{f}(x) - y)^2 + \mathrm{Var}(\hat{f}(x))\right] \\
=&\mathbb{E}(\mathbb{E}\hat{f}(x) - f(x) - \epsilon)^2 + \mathrm{Var}(\hat{f}(x)) \\
=&\mathbb{E}(\mathbb{E}\hat{f}(x) - f(x))^2 + 2\mathbb{E}\left[(\mathbb{E}\hat{f}(x) - f(x))\epsilon\right] + \mathbb{E}\epsilon^2 + \mathrm{Var}(\hat{f}(x))
\end{aligned}
$$

(For some part of this lecture material, credit goes to Professor Damek Davis, Cornell ORIE department.)

Bias-Variance Tradeoff

$$\underbrace{\mathbb{E}(\hat{f}(x) - y)^2}_{\text{test error}} = \underbrace{\left(\mathbb{E}\hat{f}(x) - f(x)\right)^2}_{\text{bias}^2} + \underbrace{\mathrm{Var}(\hat{f}(x))}_{\text{variance}} + \underbrace{\mathrm{Var}(\epsilon)}_{\text{irreducible error}}$$

- Less flexible model:

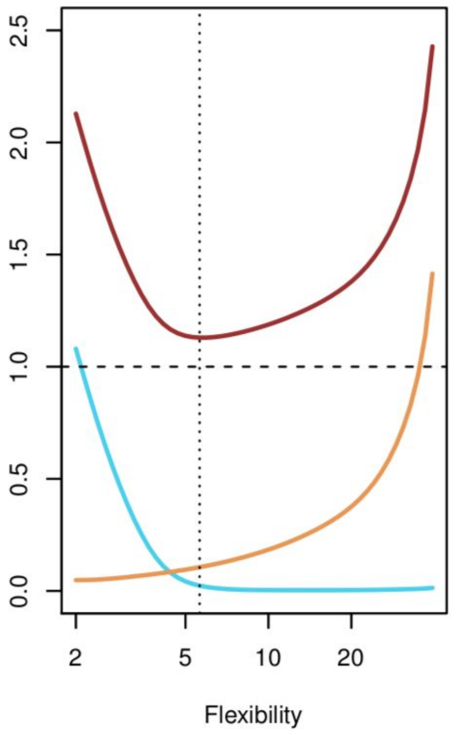
$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$$

bias high, variance low

- More flexible model:

$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \hat{\beta}_3 x^3$$

bias low, variance high



More Validation Techniques

Review: Regression vs. Classification

Regression

- Predict Continuous Data
- "On average, **how wrong** are we?"

Classification

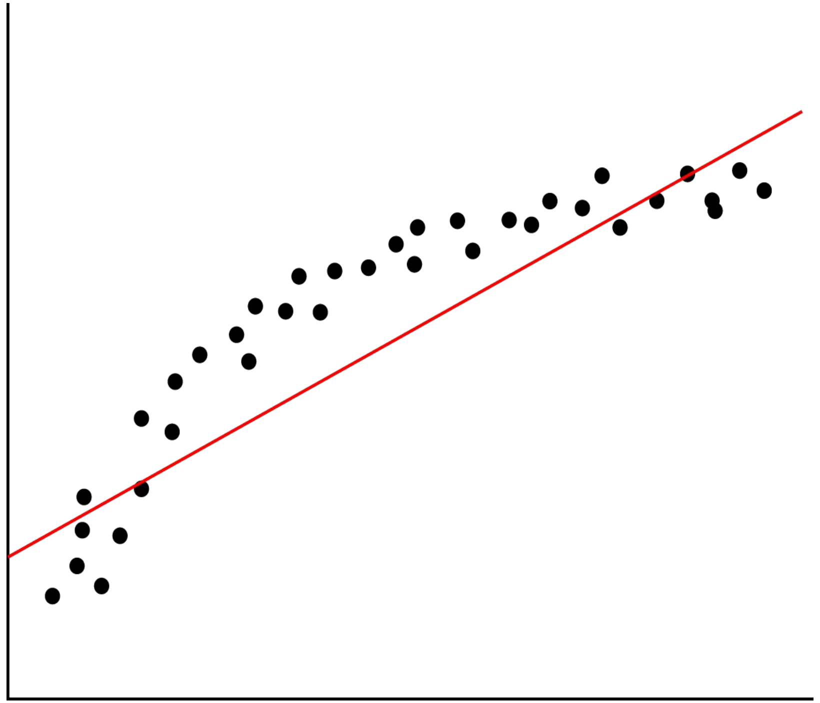
- Predict Discrete or Categorical data
- "**How many** points do we get wrong?"

$\boxed{\text{Numbers}} \neq \boxed{\text{Continuous}}$

Underfitting

Underfitting means we have high bias and low variance.

- Lack of relevant variables/factor
- Imposing limiting assumptions
  - Linearity
  - Assumptions on distribution
  - Wrong values for parameters

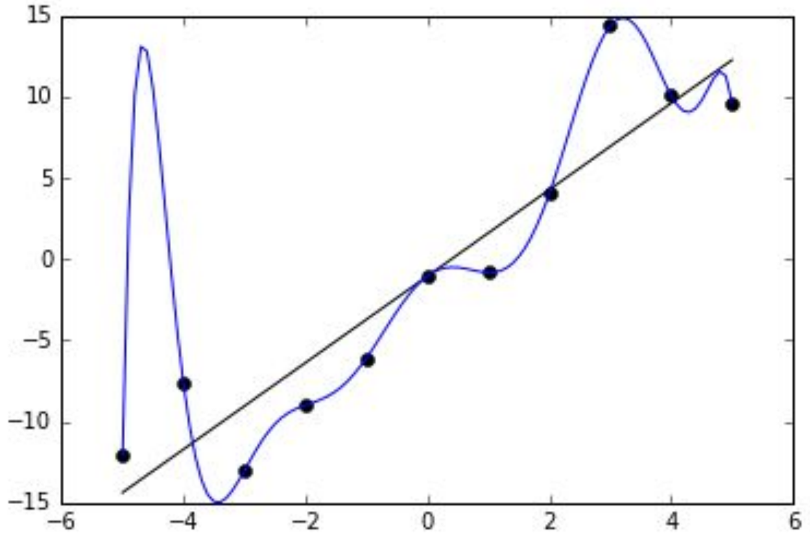


Source

Overfitting

Overfitting means we have low bias and high variance.

- Model fits too well to specific cases
- Model is over-sensitive to sample-specific noise
- Model introduces too many variables/complexities than needed



Source

Bias-Variance Tradeoff

When evaluating the models we are using, one property that we would often refer to is the bias-variance tradeoff.

Bias-variance tradeoff is the property of a set of predictive models where:

- Models with higher flexibility would typically have lower bias and higher variance;
- Models with lower flexibility would typically have higher bias and lower variance.

The graph gives a visualization on the outcomes of different bias and variance if we are doing experiment of throwing darts.

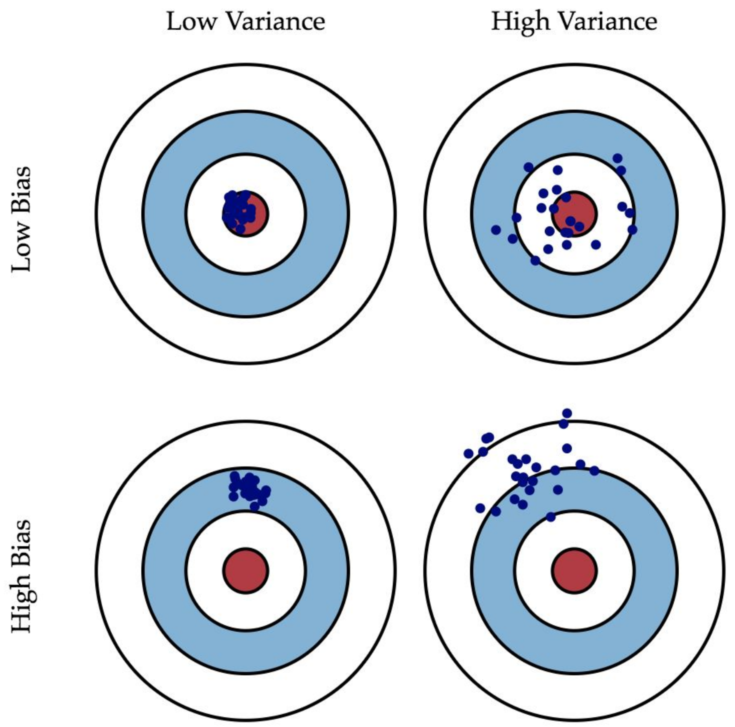


Fig. 1 Graphical illustration of bias and variance.

Graph source: http://scott.fortmann-roe.com/docs/BiasVariance.html

Bias-Variance Tradeoff

Say:

- θ is an unknown quantity we want to estimate
- $\hat{\theta}$ is an estimator of θ (computed from data, random)

The mean squared error (MSE) decomposes as:

$$\underbrace{\mathbb{E}(\hat{\theta}-\theta)^2}_{\text{MSE}} = \underbrace{\left(\mathbb{E}\hat{\theta}-\theta\right)^2}_{\text{bias}^2} + \underbrace{\text{Var}(\hat{\theta})}_{\text{variance}}$$

$$\text{MSE} = \text{bias}^2 + \text{variance}$$

Leave-P-out

```
Let **D** be our whole dataset
Choose a **P**
For every combination of **P** points in **D**:
    Use a train/test split with those **P** points as test, the rest as train
```

Leave-P-out: different from K-fold!

Let’s say **D** has a size of 4. There are four data points: *a*, *b*, *c*, and *d*.

K-fold:

- K = 2.
- Each fold has a size of 2: {*a*,*b*} and {*c*,*d*}
- So, we only have 2 possible test sets:

  {*a*,*b*} and {*c*,*d*}

Leave-P-out:

- P = 2.
- We have 6 possible test sets:

  {*a*,*b*}, {*a*,*c*}, {*a*,*d*}, {*b*,*c*}, {*b*,*d*}, and {*c*,*d*}

Leave-P-out

Pros:

- Dependable (not random)
- Representative — checks all combinations

Cons:

- Slow!
    - Runtime increases with larger datasets
    - Runtime explodes with larger P

Monte Carlo CV

- Getting accuracy **1** time doesn’t tell us much
- Getting accuracy **2** times tells us a bit
- Getting accuracy **3** times tells us a bit more
- …
- Getting accuracy **N** times might be good enough!

Take the average of those **N** times

Monte Carlo CV

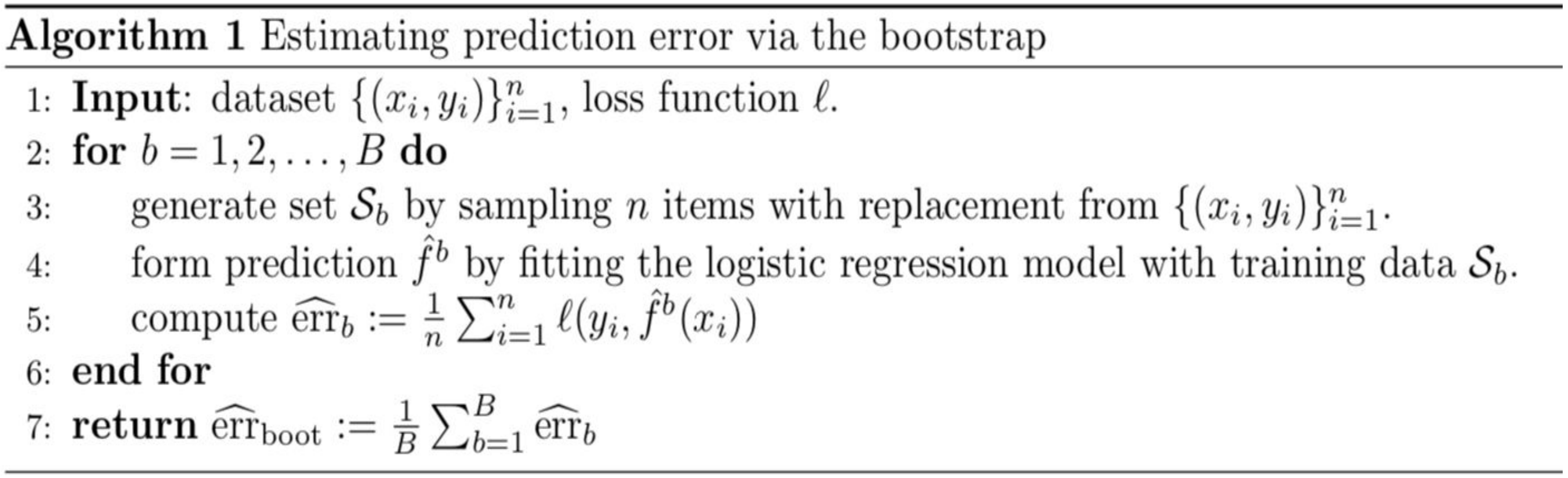
- Need to use **new**, **random** train/test split each time
  - If you use the same train/test split each time, you're not getting any new information!
- Pros:
  - easy to implement
  - easy to make faster/slower by changing number of iterations
- Cons:
  - random -> train/test splits not guaranteed to be representative of dataset
  - harder to calculate how many iterations you need

The Bootstrap

What if we don’t have enough data?

- Use **bootstrap datasets** to approximate the test error
- **Sample with replacement** from the original training dataset (with n samples) to generate **bootstrap datasets** of size n
  - Some data points may appear more than once in the generated data
  - Some data points may not appear
- Estimate of test error = average error among bootstrap datasets

Pipeline of the Bootstrap



Algorithm 1 Estimating prediction error via the bootstrap

```
1: Input: dataset $\{(x_i, y_i)\}_{i=1}^n$, loss function $\ell$.
2: for $b = 1, 2, \ldots, B$ do
3:     generate set $\mathcal{S}_b$ by sampling $n$ items with replacement from $\{(x_i, y_i)\}_{i=1}^n$.
4:     form prediction $\hat{f}^b$ by fitting the logistic regression model with training data $\mathcal{S}_b$.
5:     compute $\widehat{\text{err}}_b := \frac{1}{n} \sum_{i=1}^n \ell(y_i, \hat{f}^b(x_i))$
6: end for
7: return $\widehat{\text{err}}_{\text{boot}} := \frac{1}{B} \sum_{b=1}^B \widehat{\text{err}}_b$
```

- ❖ Does this give very good estimate of the prediction error?
- ❖ If **no,** why not? Does it **underestimate** or **overestimate** the prediction error?

Pipeline of the Bootstrap

No.

Since the bootstrap is sampling with replacement for B validation folds, each fold would have significant overlap with the original data used for training. Approximately, **2/3** of the training data would appear in each validation fold.

This leads to significant **underestimation** of the prediction error.

- ❖ Why **2/3**?

Proof

Suppose there are n samples *(yi, xi)* in the training set data, then for each of the bootstrap validation fold b, every sample has probability **1/n** of being selected into b. The probability of each data sample *(yi, xi)* not being in fold b is **(1 - 1/n)**, respectively.

Probability of **avoid selecting** all n data points in fold b: **(1 - 1/n)^n**.

When n gets large, we have this probability converges to 1/e.
(Why this converges? Probably should review Calc I, or see: https://math.stackexchange.com/questions/882741/limit-of-1-x-nn-when-n-tends-to-infinity.)

Therefore, the fraction of overlapping data points in each fold is **(1-1/e)**, which is about **2/3**.

How we fix the problem

- ❖ Requires some thoughts when generating validation set, especially for real-world complex data.
- ❖ Can partly fix this problem by only using predictions for those observations that did not (by chance) occur in the current bootstrap sample.
- ❖ But the method gets complicated.

Bootstrap vs. k-fold

In K-fold validation, each of the K folds is distinct from the other (K − 1) folds used for training: there is **no overlap**.

This is crucial for its success in estimating prediction error.

Why we still use bootstrap

- Bootstrap allows us to use a computer to mimic the process of obtaining new data sets.
- Can be used to quantify the uncertainty associated with a given estimator or statistical learning method.
- Provides an estimate of the standard error of a coefficient, or a confidence interval for that coefficient.
    - i.e., the variability of the model!

Bagging (Bootstrap Aggregating)

What if we don't have enough data?

- Bagging is a common technique that builds on Bootstrapping
- Main Idea: Do Bootstrapping a bunch and make a classifier for each bootstrap, then majority prediction wins.
- Many weak learners aggregated typically outperform a single learner over the entire set, and overfits less.
    - Principle behind Random Forests

Measures of overfitting

- Compare *train* accuracy with *test* accuracy!
- Method 1: **Generalization error**
  - (Generalization error) = (test error) - (train error)
  - Error introduced from trying to *generalize* to new data
- Method 2: **Prediction variation**
  - VAR(predictions for a specific point x)
  - How much do predictions change for point x, based on the training data?

Reminder: don't mix up error and accuracy! They're nearly equivalent measures, but confusing them in your code can cause problems

Coming Up

- **Assignment 8**: Due at 5:30pm, Apr. 28
- **Next Lecture**: Applications of *Un*supervised Learning

CDS Education

We explore, learn, and educate big minds.