# Machine Learning

# Max Margin Classification And Multi-class SVM

Dan Goldwasser

dgoldwas@purdue.edu

# Learning with Regularization

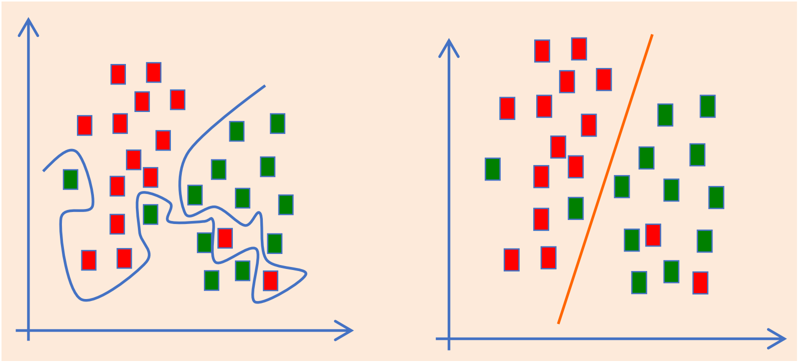
- ***A form of inductive bias – we prefer simpler functions!***
- A very popular choice of regularization term is to minimize the norm of the weight vector
  - For convenience: ½ squared norm

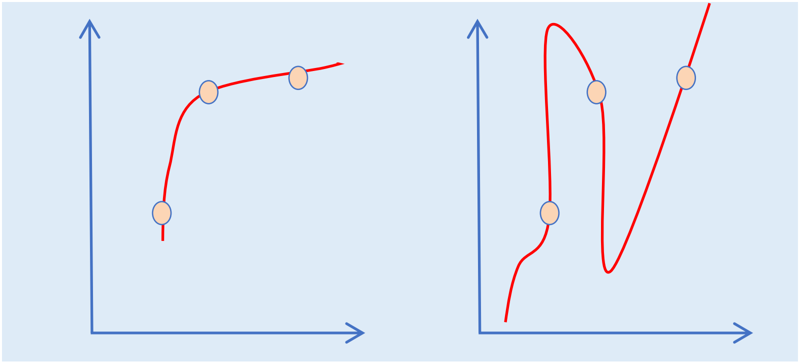
$$\min_{\mathrm{w}} \quad \sum_n loss(y_n, \mathrm{w}_n) + \frac{\lambda}{2}||\mathrm{w}||^2$$

# Regularization

- We have the ability to ”blow up” the representation space (implicitly define highly non-linear classifiers in the original space)
- *We can control this complexity by tuning the regularizer weight, and potentially using other techniques.*

$$\min_{\mathrm{w}} = \sum_{n} loss(y_n, \mathrm{w}_n) + \lambda R(\mathrm{w})$$





# *Quick Detour: understanding Overfitting*

- Easy to define very powerful learners
  - Move to a higher dimension
- ***Is it always a good idea?***
- Simple (simplistic) approach:
- Learning doesn't work– *our model is not powerful enough!*
  - *Move to a richer representation*
- Learning doesn't work – *our model overfits!*
  - *Get more data! Learn a lower degree model!*
- ***How can we tell?***

# *Quick Detour: understanding Overfitting*

- Easy to define very powerful learners
  - Move to a higher dimension
- ***Is it always a good idea?***

Learning doesn't work– *our model is not powerful enough!*

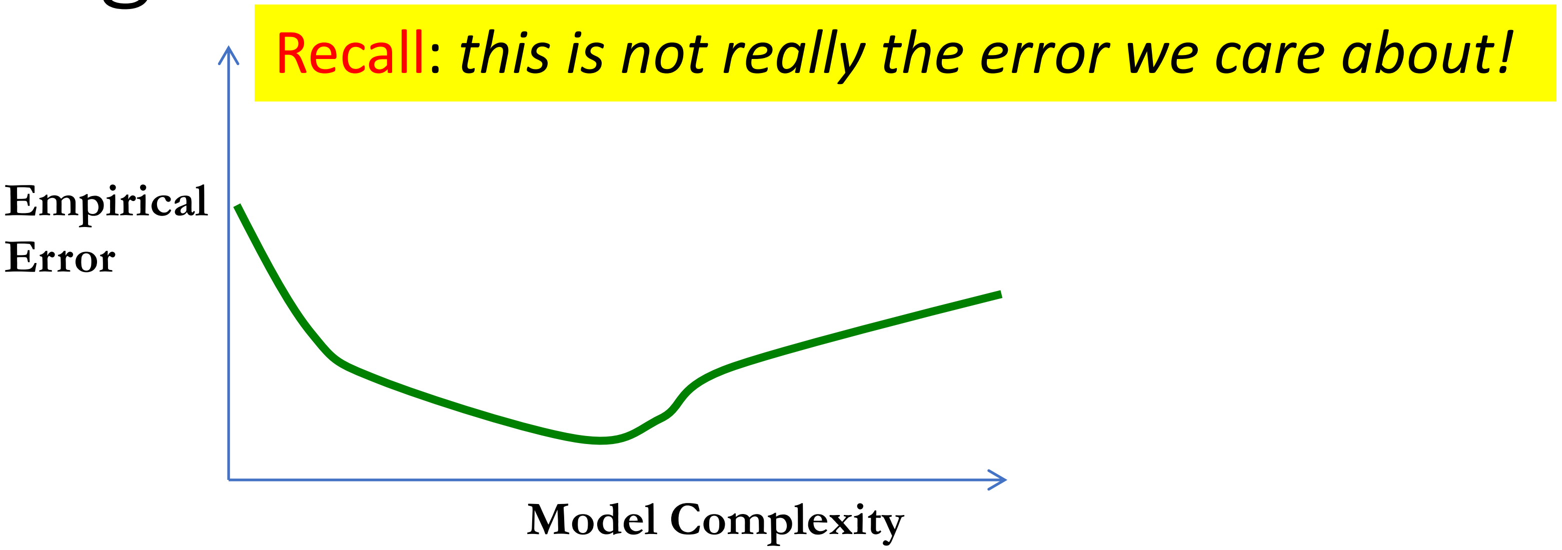
*Move to a richer representation*

Learning doesn't work – *our model overfits!*

*Get more data! Learn a lower degree model!*

- ***How can we tell?***

# Overfitting



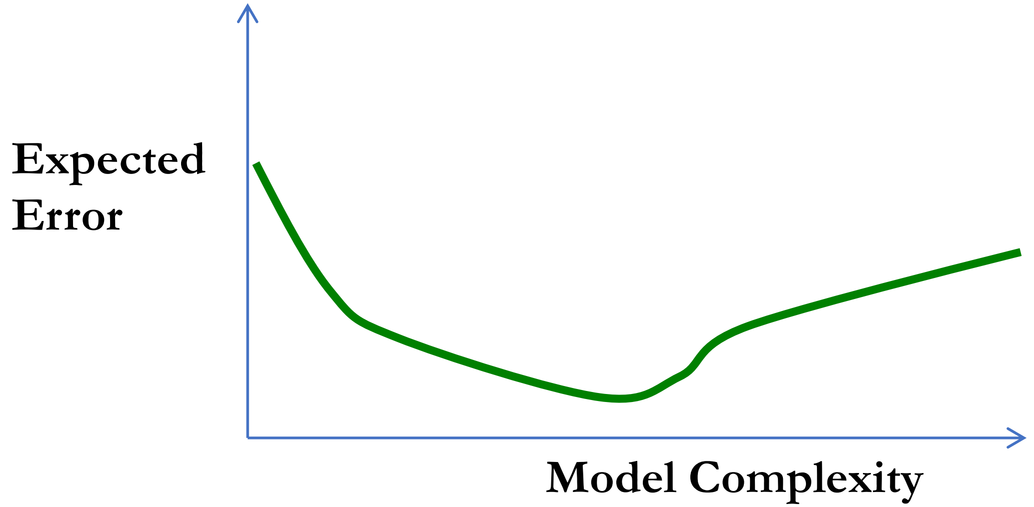

- **Model complexity:** *"how many parameters are learned?"*
  - *Decision tree: depth, Linear models: features, degree of a polynomial kernel*
- ***Empirical Error:*** *For a given dataset, what is the percentage misclassified items by the learned classifier?*

# Data Generating Distribution

- We assume that a distribution $P(x,y)$ from which the dataset is sampled
  - *Data generating distribution*
- Training and testing examples are drawn from the **same** distribution
  - The examples are **identically** distributed
- Each example is drawn **independently**
  - We say that the train/test examples are ***independently and identically distributed*** (i.i.d)
- We care about our performance over any sample from that distribution
  - *not just the one we observe during training*

# Overfitting

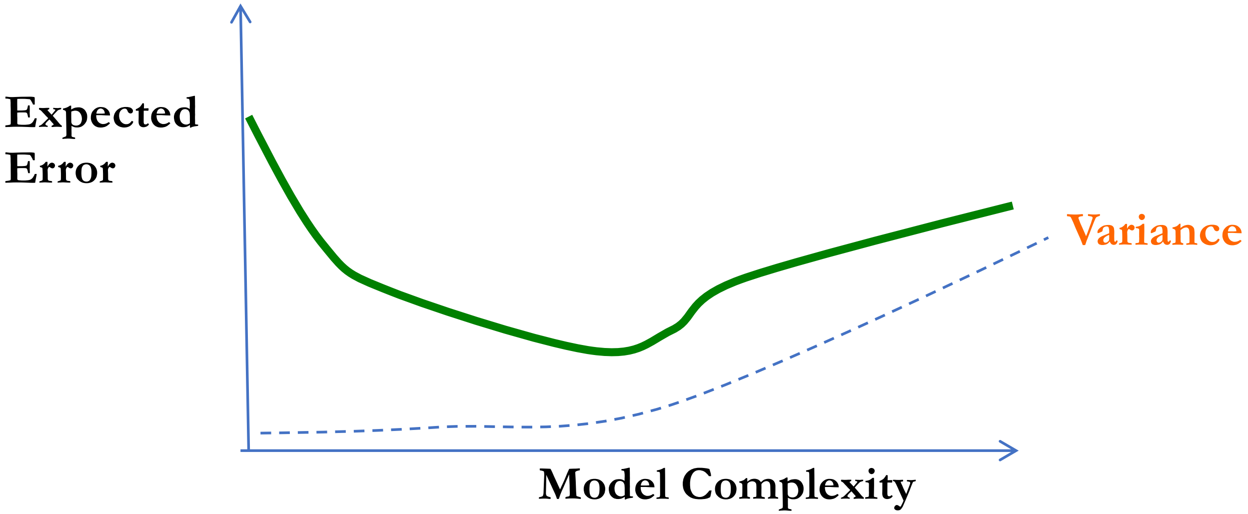
**Expected Error**: *what percentage of items drawn from P(x,y) do we expect to be misclassified by the learned classifier?*




let's consider **different** samples from the data distribution. You can get closer to the **expected loss** by considering the expectation of the empirical loss on (all/many) different data samples

***Let's continue the discussion with that in mind!***

# The Variance of the Learner

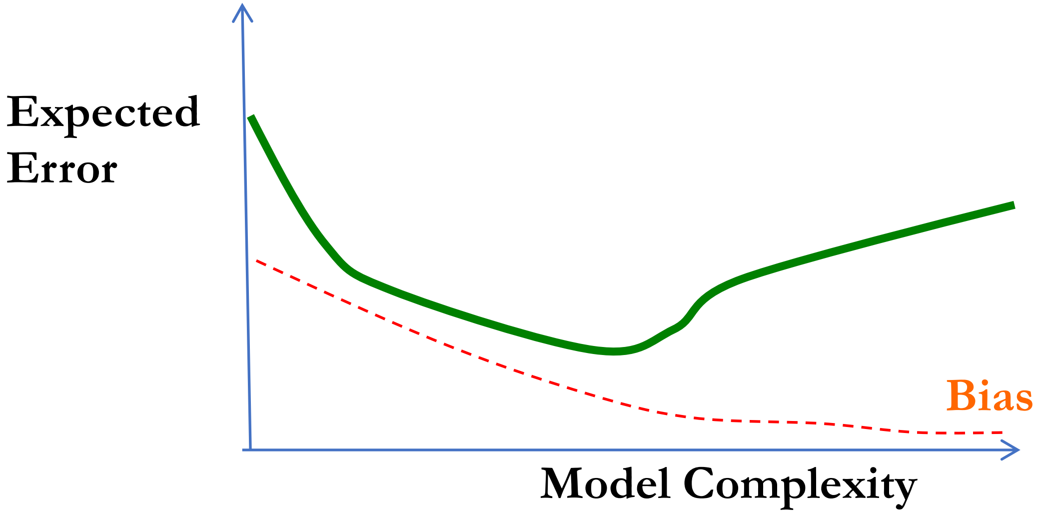



**How susceptible is the learner to minor changes in the training data?**
(i.e., different samples from P(x,y))

- You can think about the testing data as another sample
  - *Overfitting*

***Variance increases with model complexity!***

# Bias of the Learner

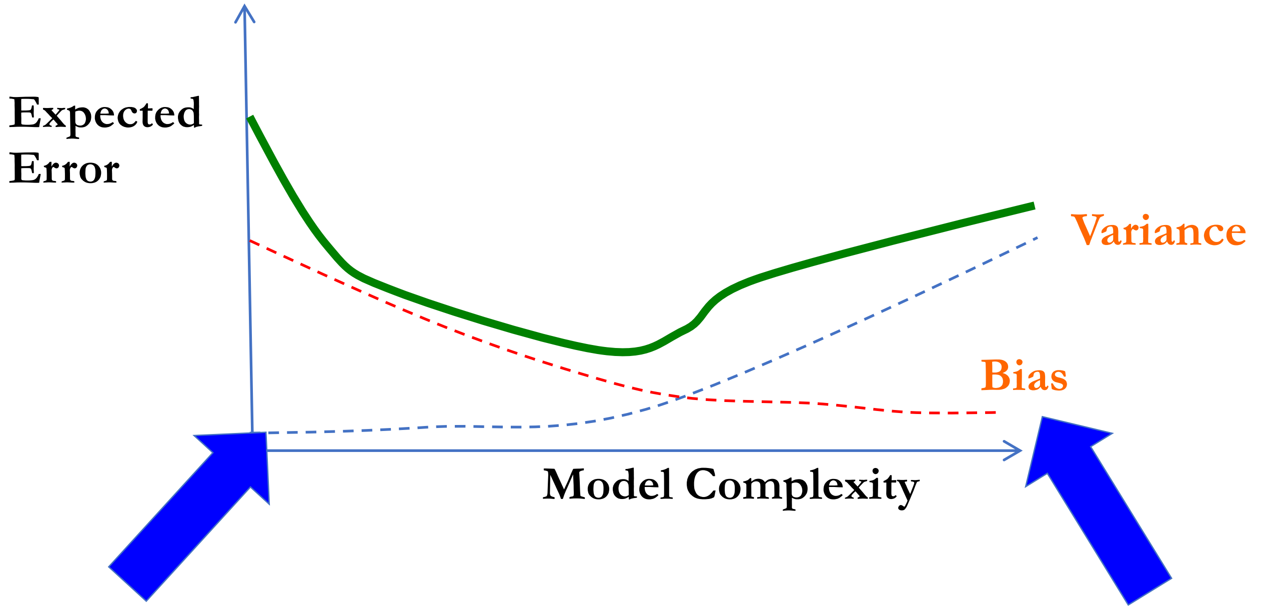



**How likely is the learner to identify the target hypothesis?**

- What will happen if we test on a different sample?
  - **Underfitting**
- **What will happen if we train on a different sample?**

**Bias is high when the model is too simple**

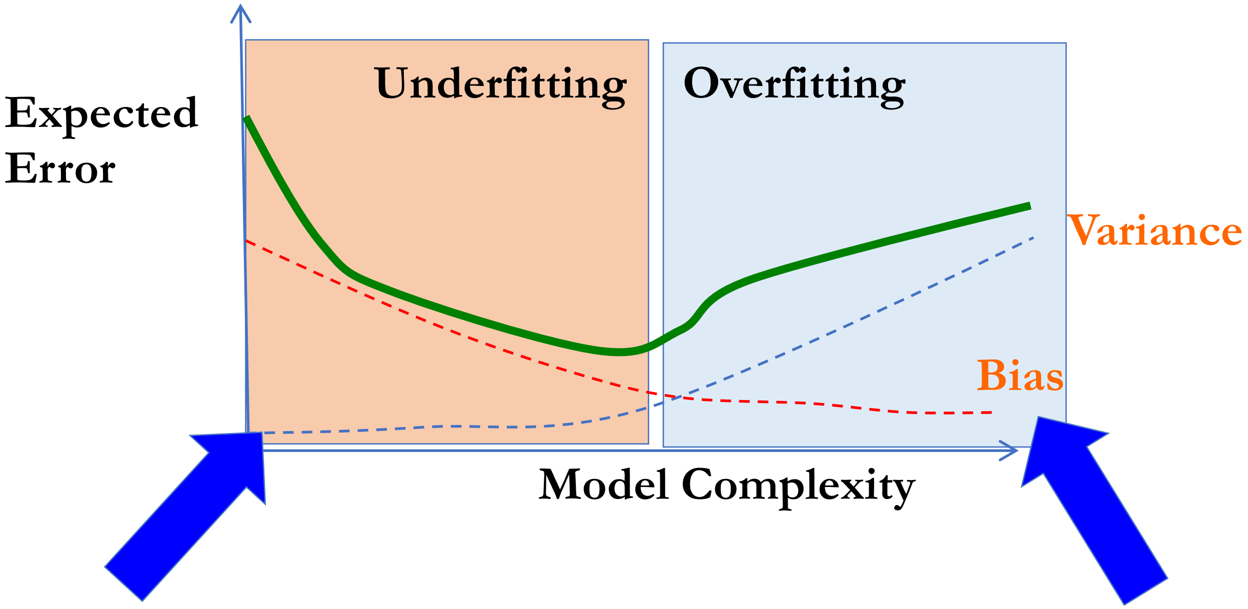
# Model Complexity




Simple models:
High bias and
low variance

Complex models:
High variance
and low bias

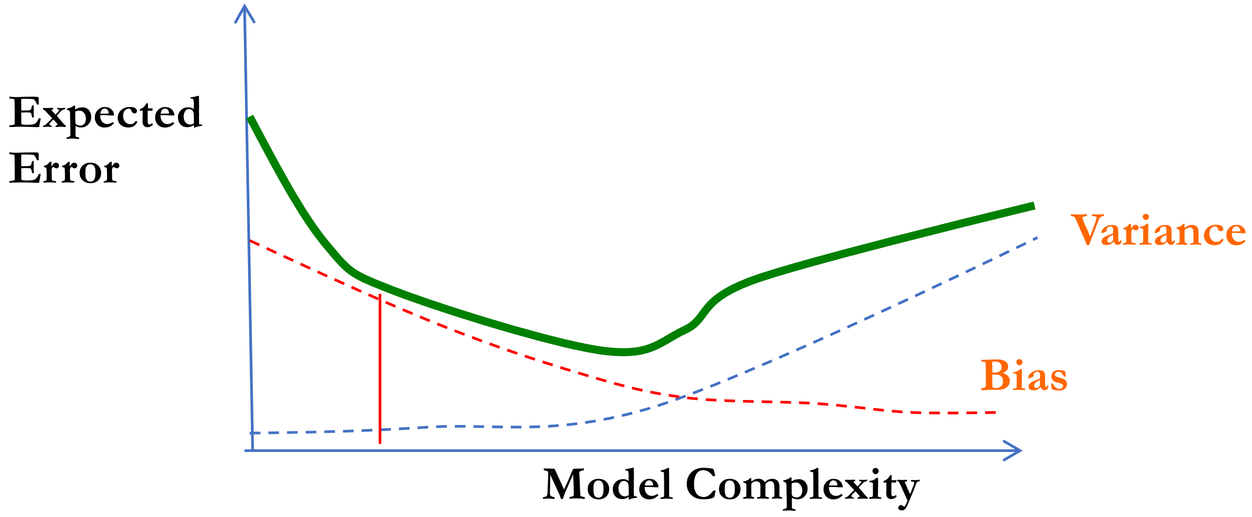
# Model Complexity




**Simple models:**
**High bias** and
low variance

**Complex models:**
**High variance**
and low bias

# Impact of bias and variance




Expected Error ≈ Bias + Variance

# Bias-Variance Tradeoff

Sample different sets from P(x,y) and train (each $\mathbf{x}$ is a learned model)

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

How would the learned models “behave” when trained over different datasets from the distribution?

Bias


High bias
High variance

Low bias
High variance

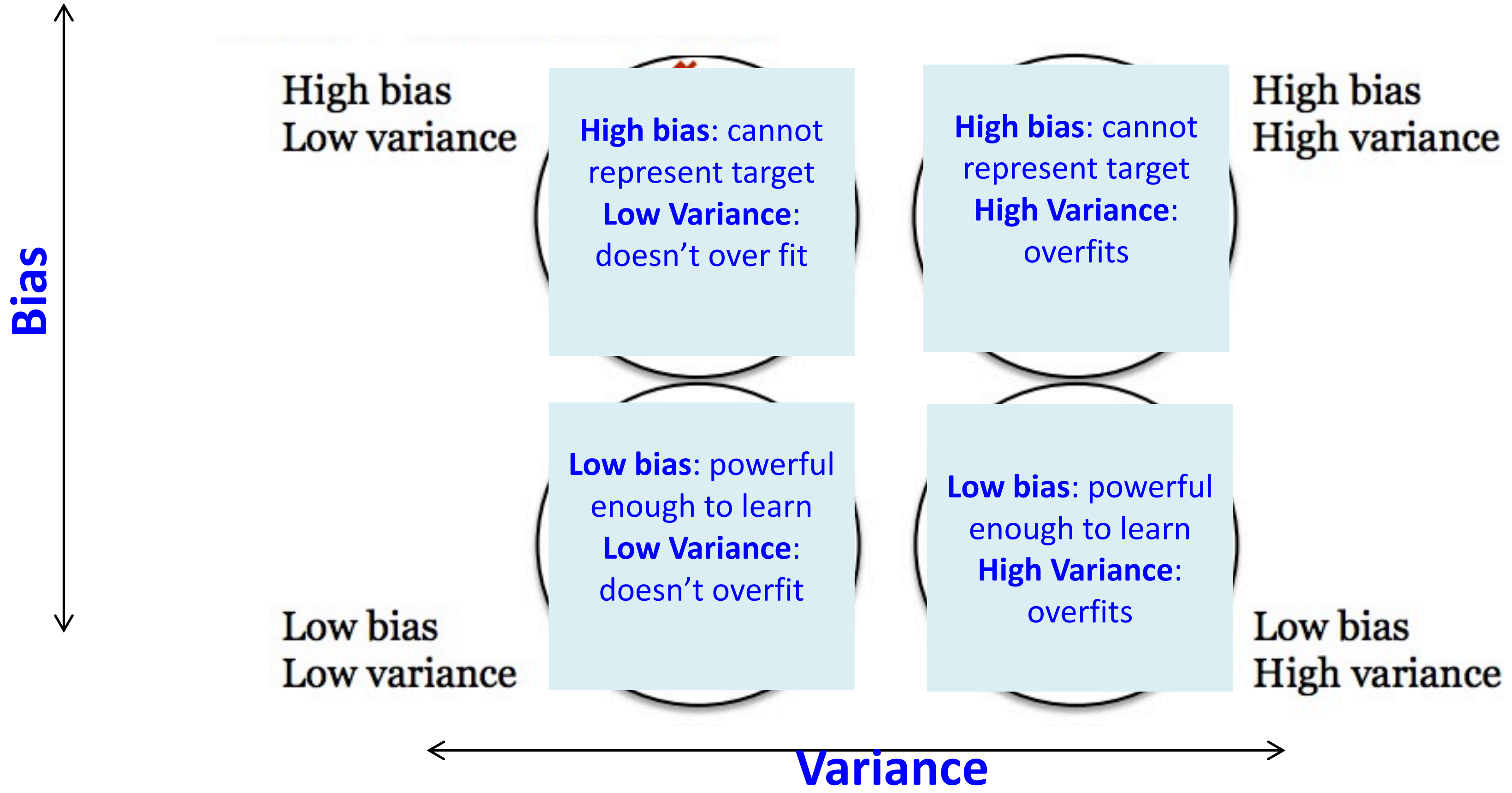
Variance

# Bias-Variance Tradeoff

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

Sample different sets from P(x,y) and train (each **x** is a learned model)

How would the learned models "behave" when trained over different datasets from the distribution?



# Bias-Variance Tradeoff

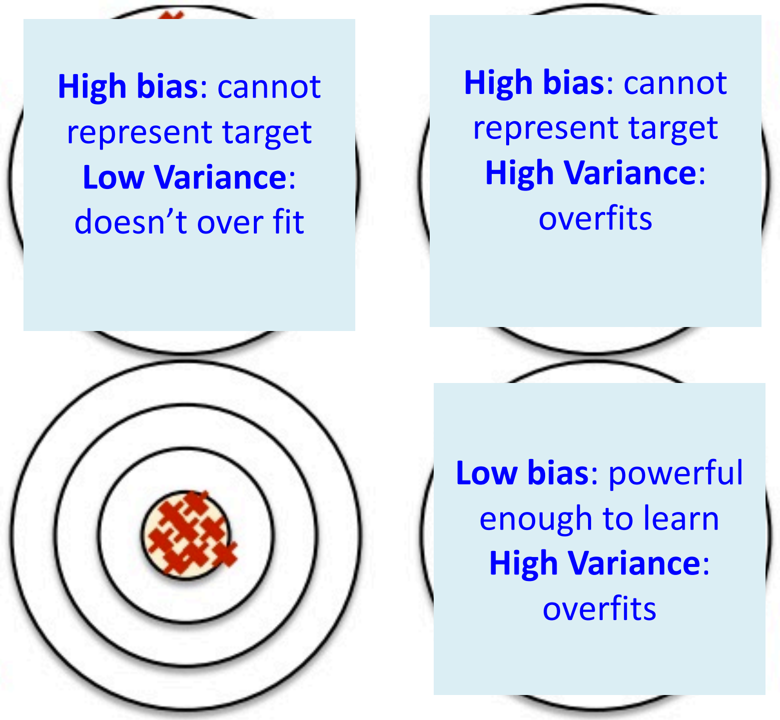
Sample different sets from P(x,y) and train (each **x** is a learned model)

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

How would the learned models “behave” when trained over different datasets from the distribution?

**Bias**

High bias
Low variance



**High bias**: cannot represent target
**Low Variance**: doesn’t over fit

**High bias**: cannot represent target
**High Variance**: overfits

High bias
High variance

Clustered together (low variance) in a “good” place (low bias)

**Low bias**: powerful enough to learn
**High Variance**: overfits

Low bias
Low variance

Low bias
High variance

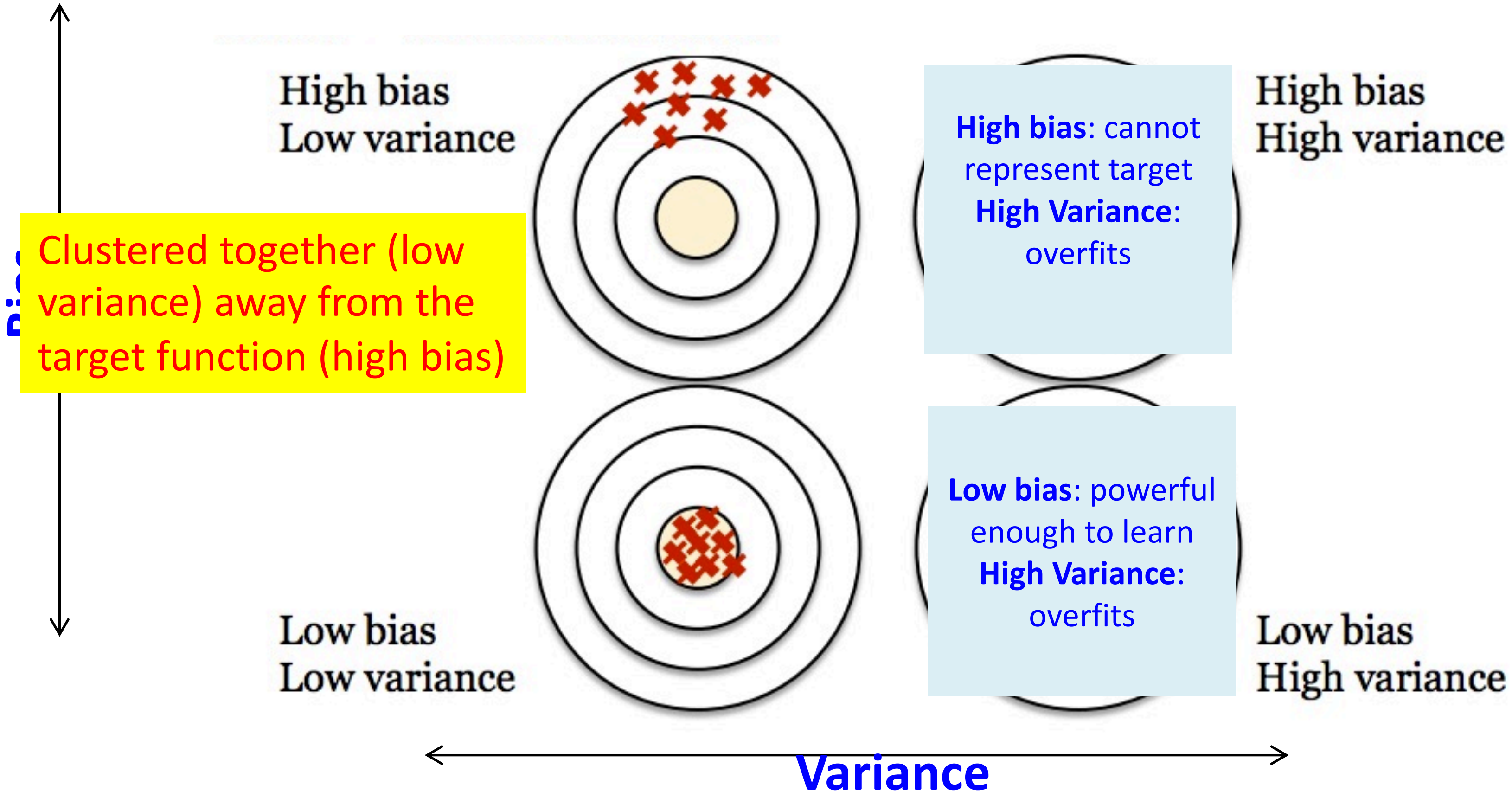
**Variance**

# Bias-Variance Tradeoff

Sample different sets from P(x,y) and train (each $\mathbf{x}$ is a learned model)

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

How would the learned models “behave” when trained over different datasets from the distribution?




Clustered together (low variance) away from the target function (high bias)

**High bias**: cannot represent target
**High Variance**: overfits

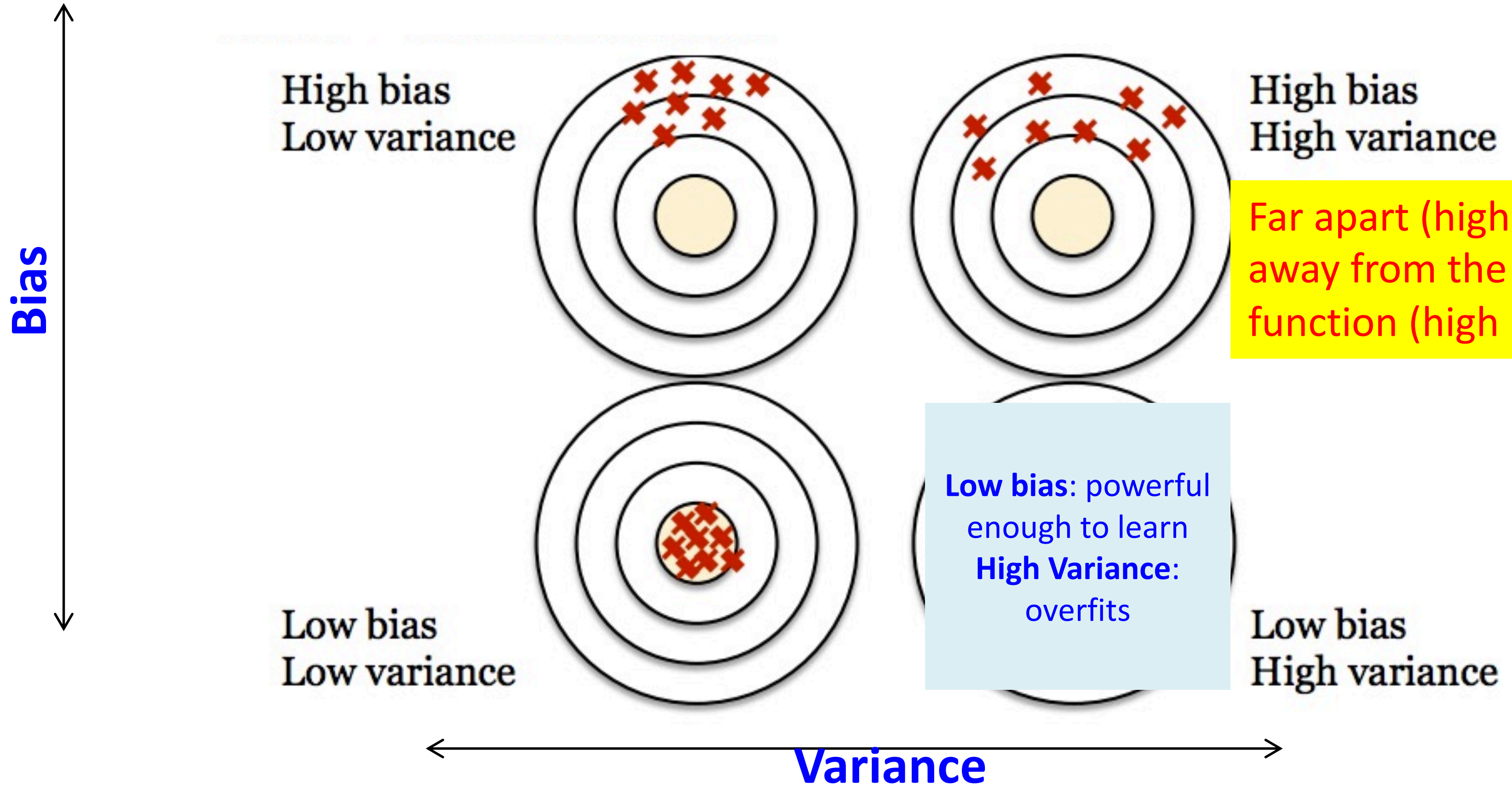
**Low bias**: powerful enough to learn
**High Variance**: overfits

# Bias-Variance Tradeoff

Sample different sets from P(x,y) and train (each $\mathbf{x}$ is a learned model)

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

How would the learned models "behave" when trained over different datasets from the distribution?




Far apart (high variance) away from the target function (high bias)

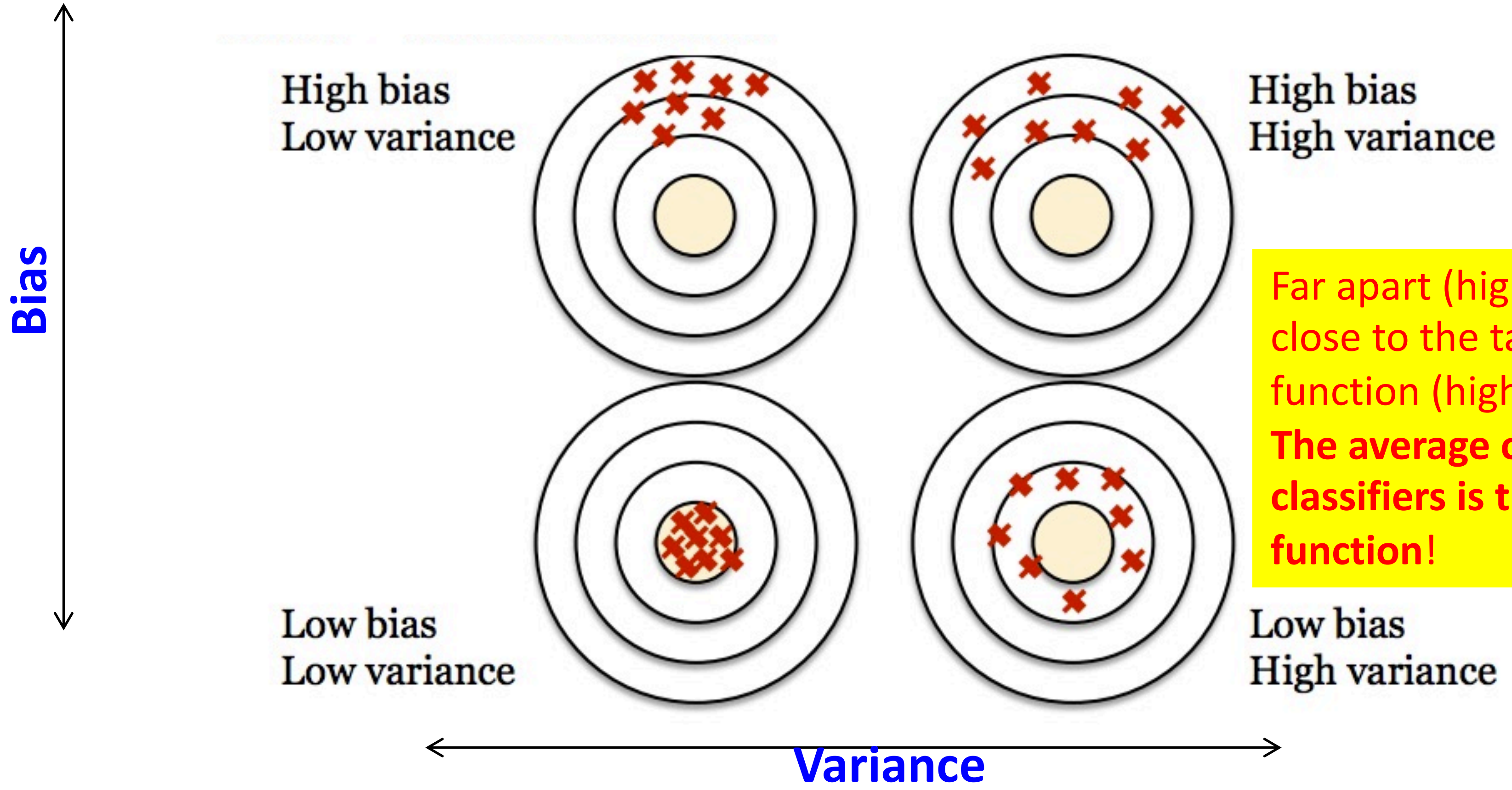
**Low bias**: powerful enough to learn
**High Variance**: overfits

# Bias-Variance Tradeoff

Dartboard = hypothesis space
Bullseye = target function
Darts = learned models

Sample different sets from P(x,y) and train (each **x** is a learned model)

How would the learned models "behave" when trained over different datasets from the distribution?




Far apart (high variance) close to the target function (high bias)
**The average of the classifiers is the target function**!

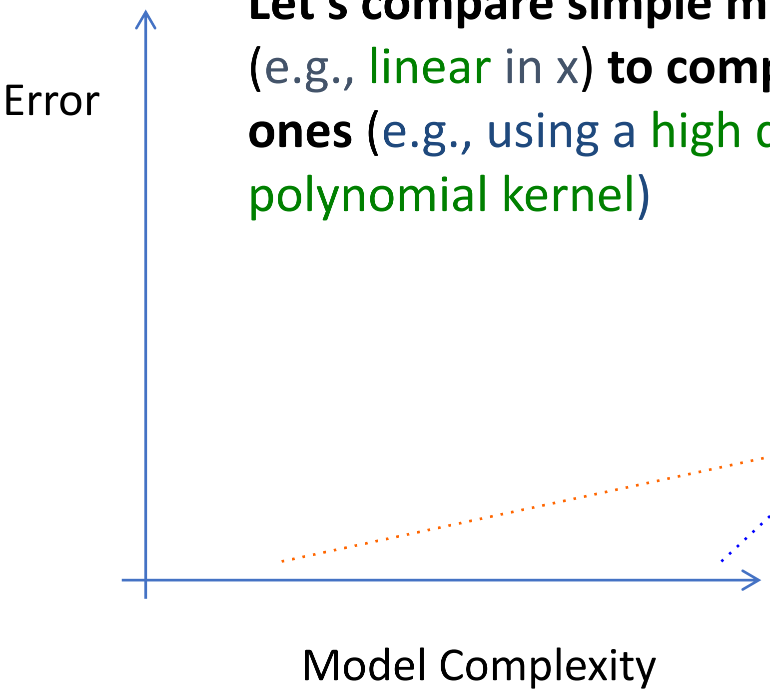
# Bias-Variance Tradeoff in Practice

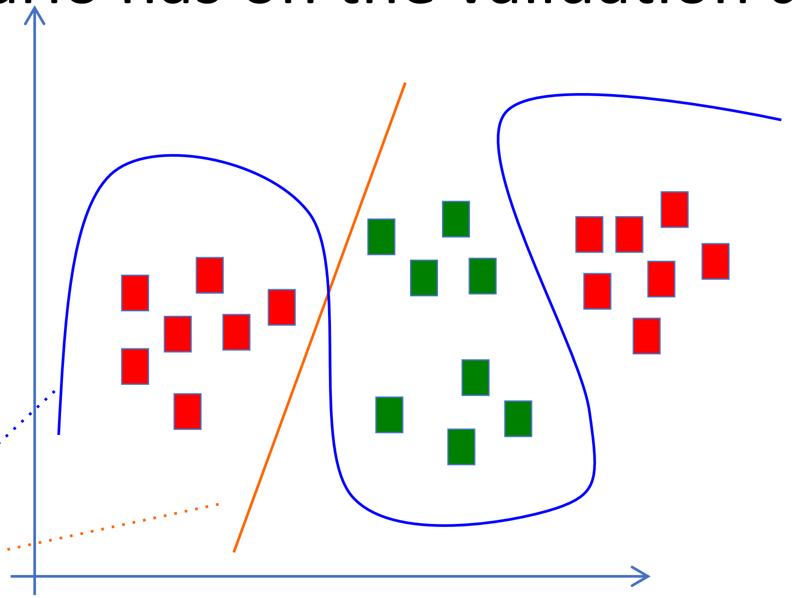
- We saw that the classification error can be expressed in terms of bias and variance
  - Similar claims could be made for classification loss
- Reducing the bias/variance can reduce expected error!
- Different scenarios can lead to different actions for reducing the error
  - High **bias**: add more features
  - High **variance**: simplify the model, add more examples
- How can we diagnose each one of these scenarios?

# Bias-Variance Analysis

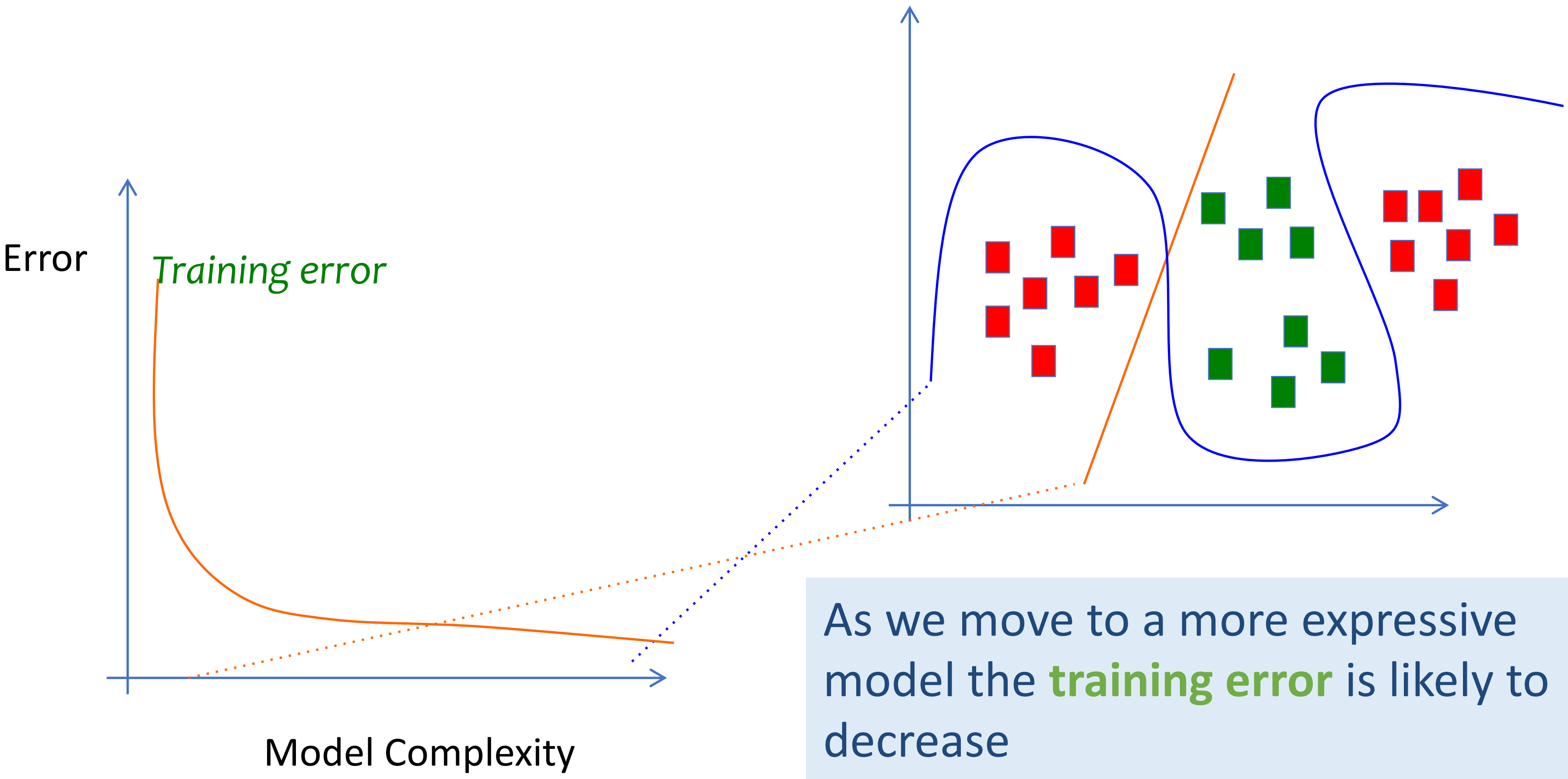
- Let’s look at the influence each scenario has on the validation and training errors

**Let’s compare simple models** (e.g., linear in x) **to complex ones** (e.g., using a high degree polynomial kernel)



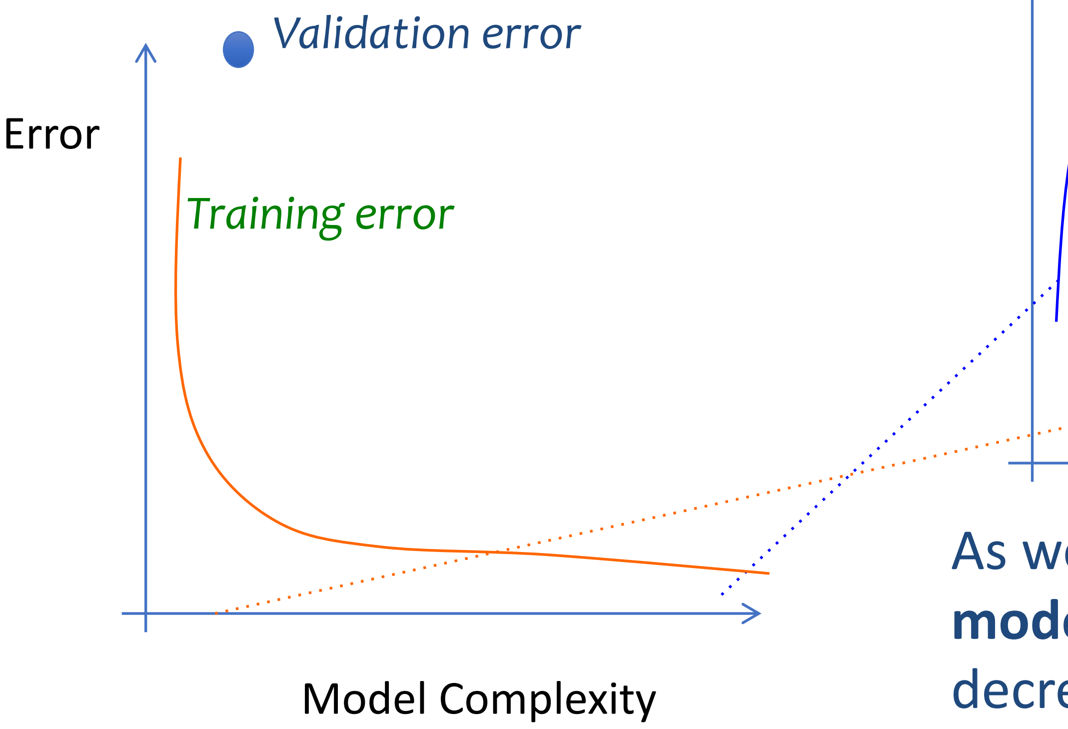
# Bias-Variance Analysis

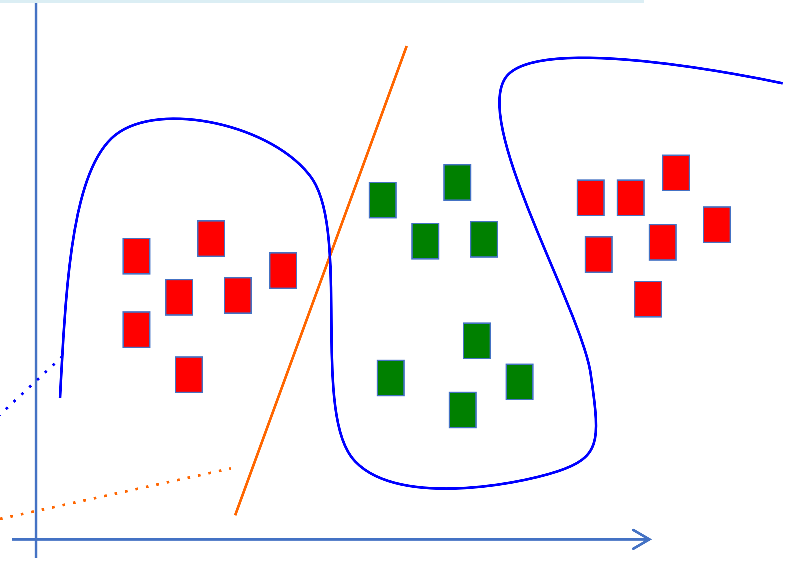



As we move to a more expressive model the **training error** is likely to decrease

# Bias-Variance Analysis

*When we are using a* ***simple model****, it's very likely we'll underfit resulting in a* ***high validation*** *error*

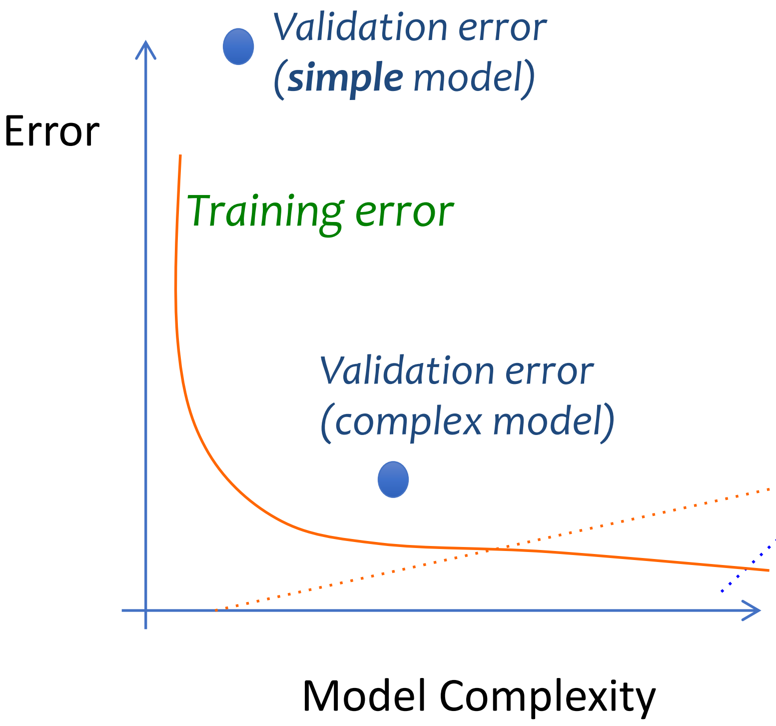


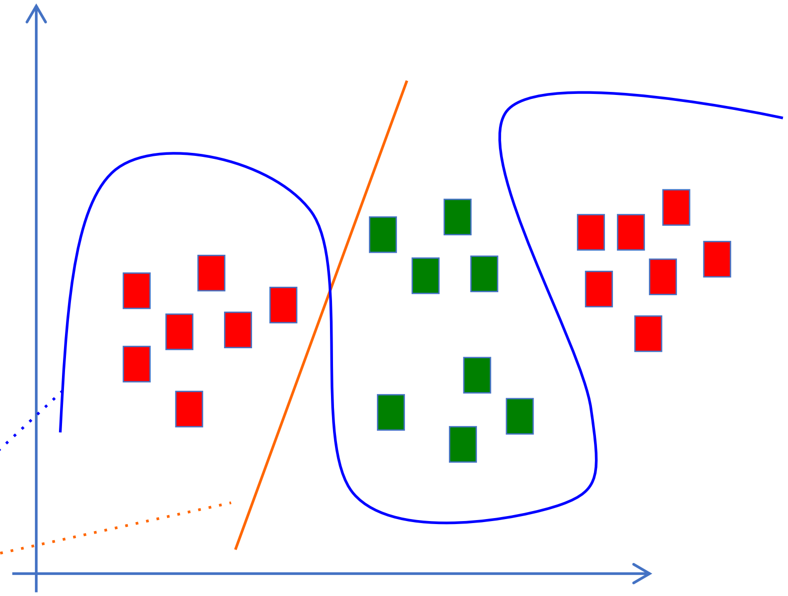



As we move to a **more expressive model** the **training error** is likely to decrease

# Bias-Variance Analysis

*When we are using a **simple model,** it’s very likely we’ll underfit resulting in a **high validation** error*

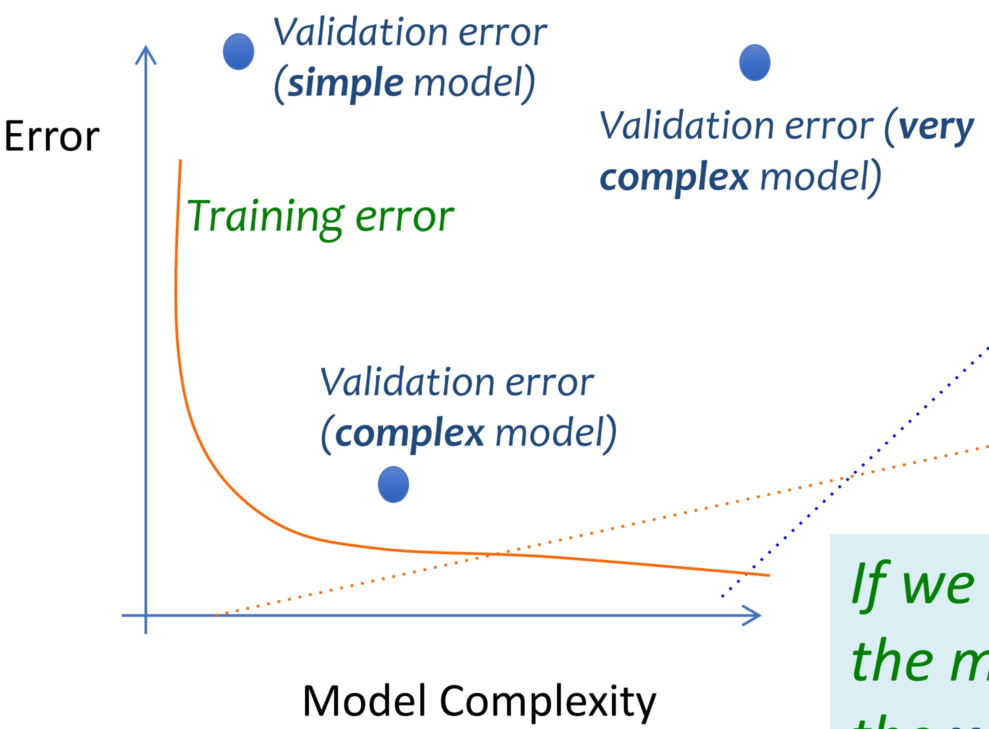


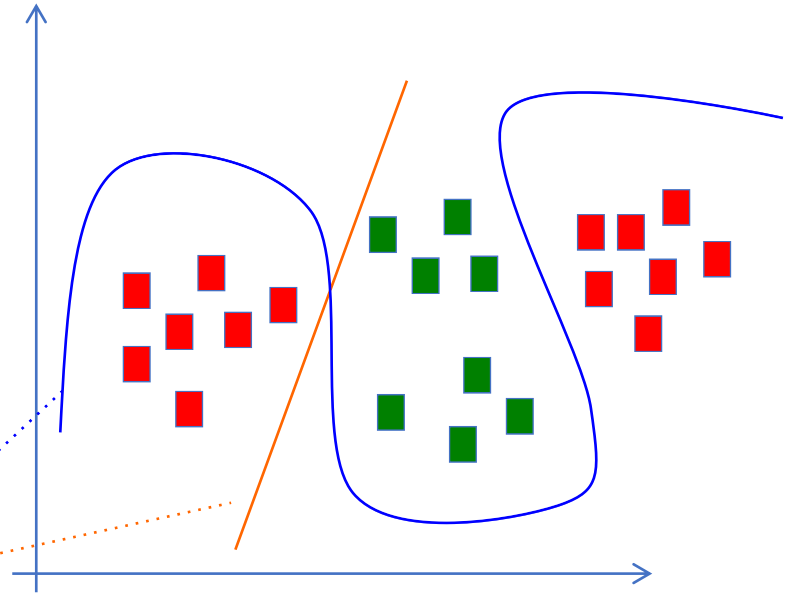



***If we increase the complexity, we can do better on the validation data***

# Bias-Variance Analysis

*When we are using a **simple model,** it's very likely we'll underfit resulting in a **high validation** error*

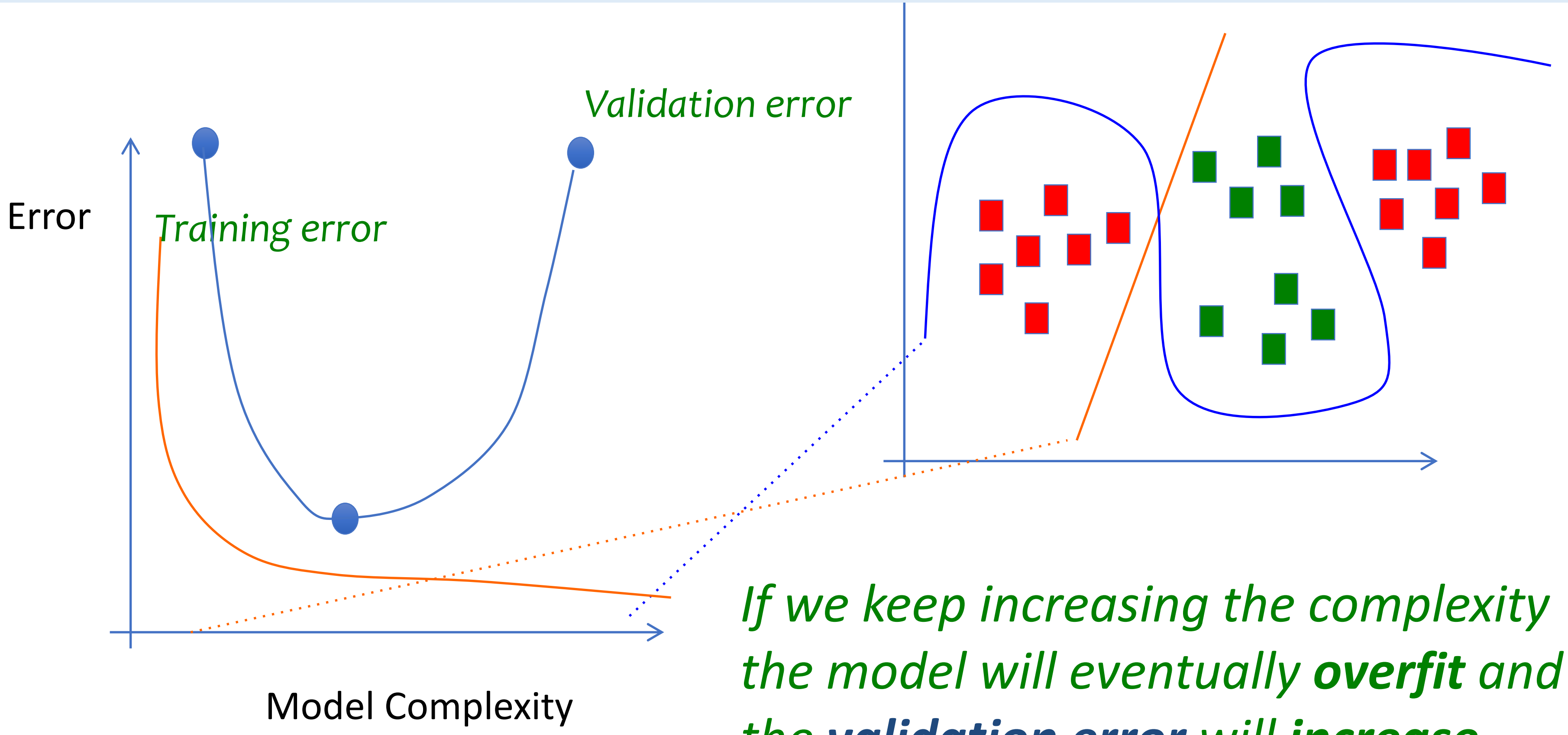


*If we keep increasing the complexity the model will eventually **overfit** and the **validation error** will **increase***

# Bias-Variance Analysis

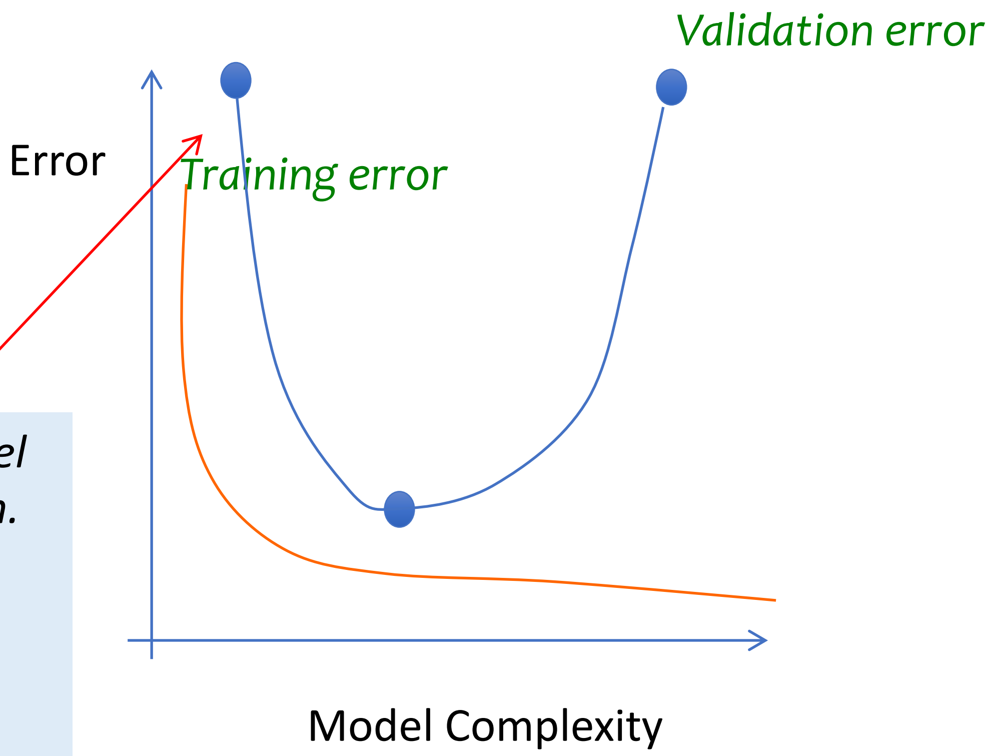
*Interpolating over the points: two curves that we can use for **diagnosis***




*If we keep increasing the complexity the model will eventually **overfit** and the **validation error** will **increase***

# Bias-Variance Analysis

*Interpolating over the points: two curves that we can use for* ***diagnosis***




***Bias problem :*** *the model is not expressive enough.*

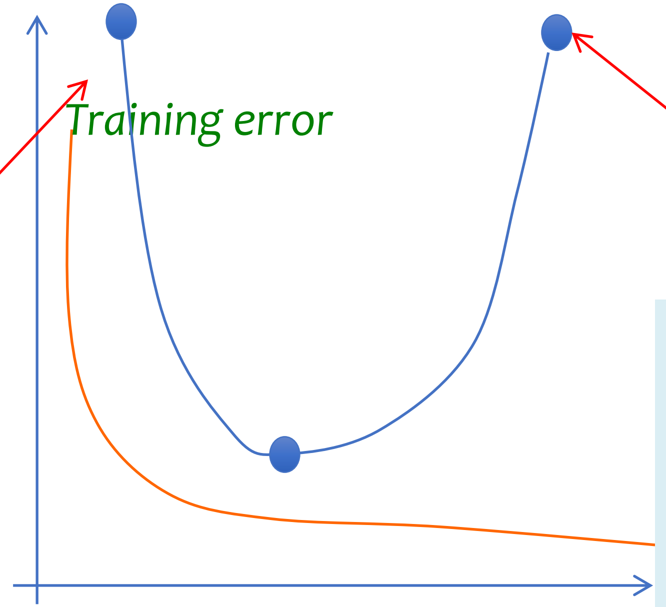
***Underfitting***

*(1) high training error*

*(2) validation error ≈ training error*

# Bias-Variance Analysis

*Interpolating over the points: two curves that we can use for* ***diagnosis***




***Bias problem :*** *the model is not expressive enough.*

***Underfitting***

*(1) high training error*

*(2) validation error ≈ training error*

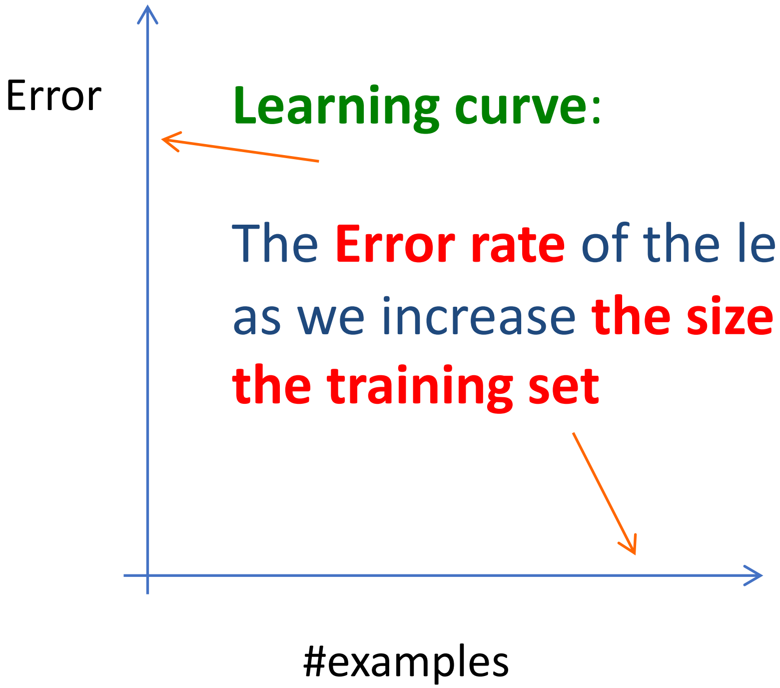
***Variance problem:*** *the model is too expressive for the data set that we have* ***overfitting***

*(1) low training error,*

*(2) validation error > training error.*

# Learning Curve

Plotting the learning curve of an algorithm on a given data set is also a useful way to diagnose the algorithm's performance.

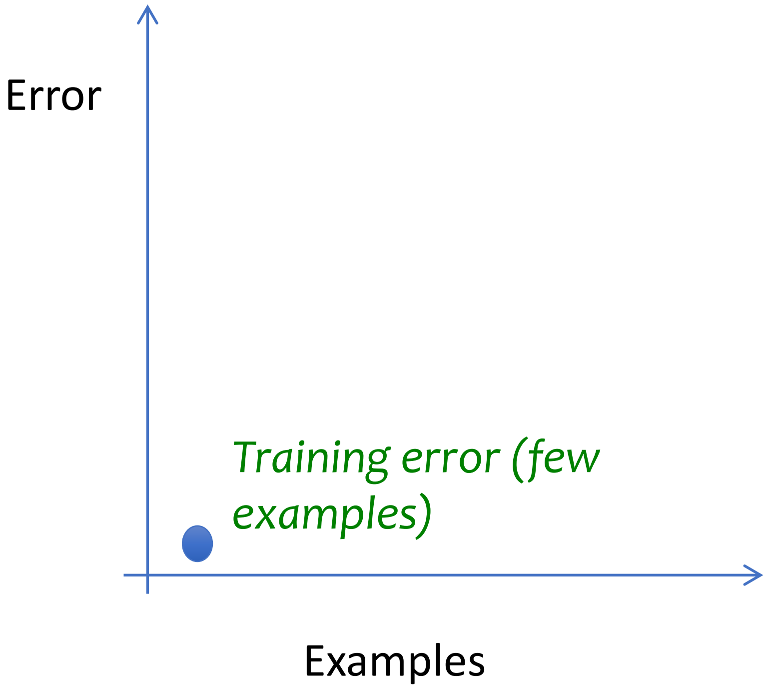


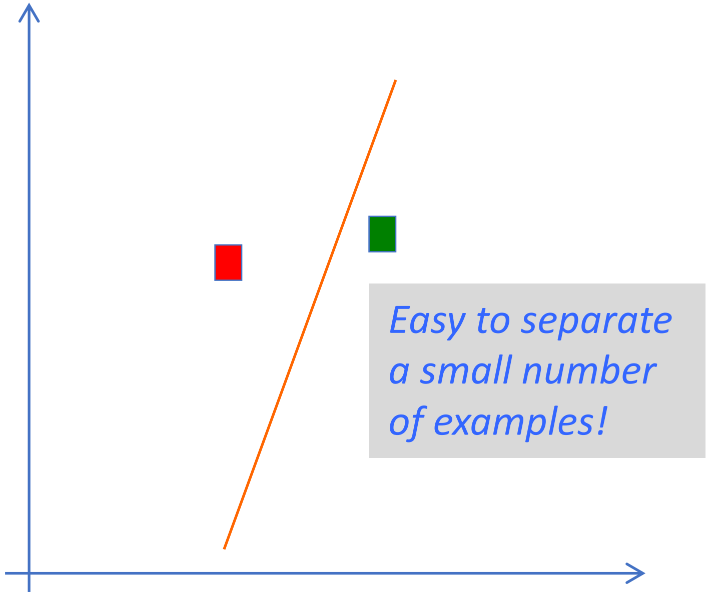

**Learning curve**:

The **Error rate** of the learner as we increase **the size of the training set**

# Learning Curve

Plotting the learning curve of an algorithm on a given data set is also a useful way to diagnose the algorithm's performance.

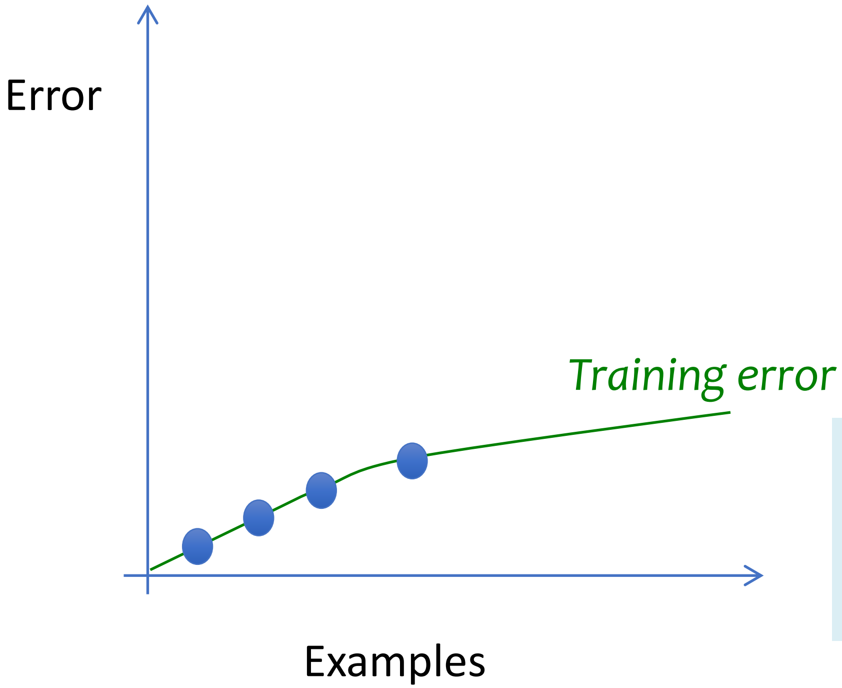


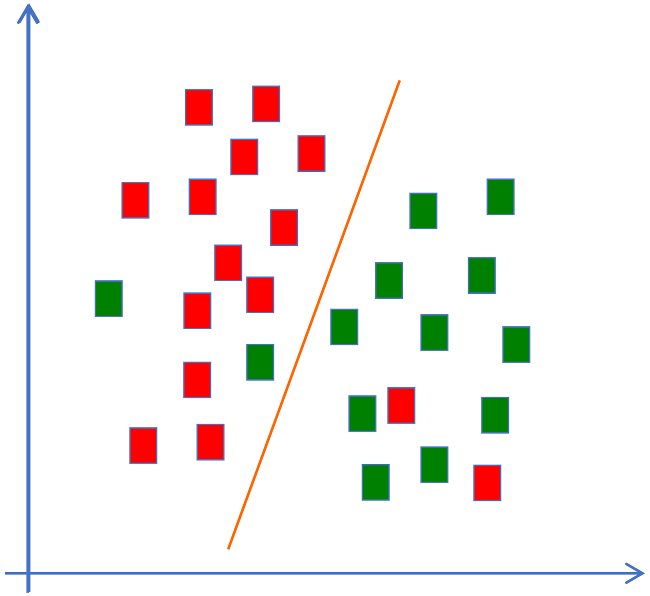




*When the algorithm has access to a **few examples** – training error will be very **low***

# Learning Curve

Plotting the learning curve of an algorithm on a given data set is also a useful way to diagnose the algorithm's performance.

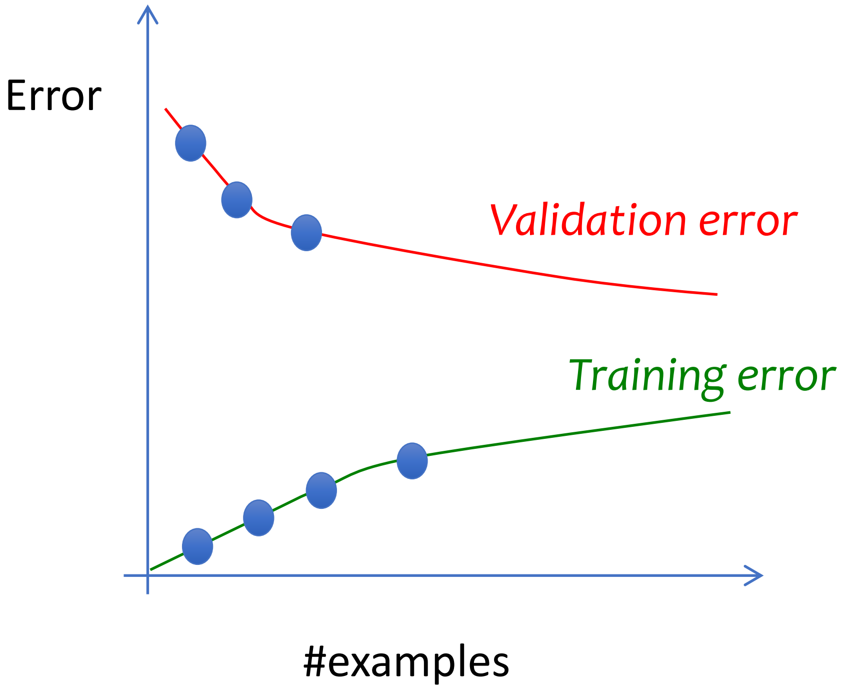


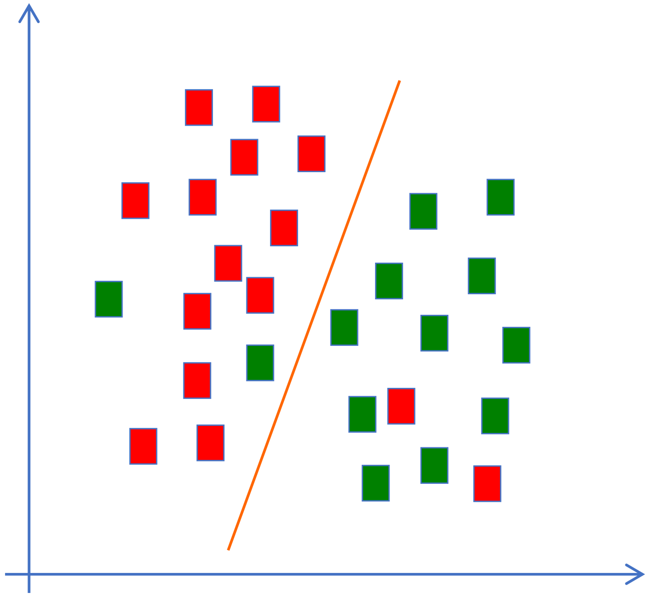



As **more examples** are sampled, it is more difficult to separate the data and **the training error will increase**

# Learning Curve

Plotting the learning curve of an algorithm on a given data set is also a useful way to diagnose the algorithm's performance.

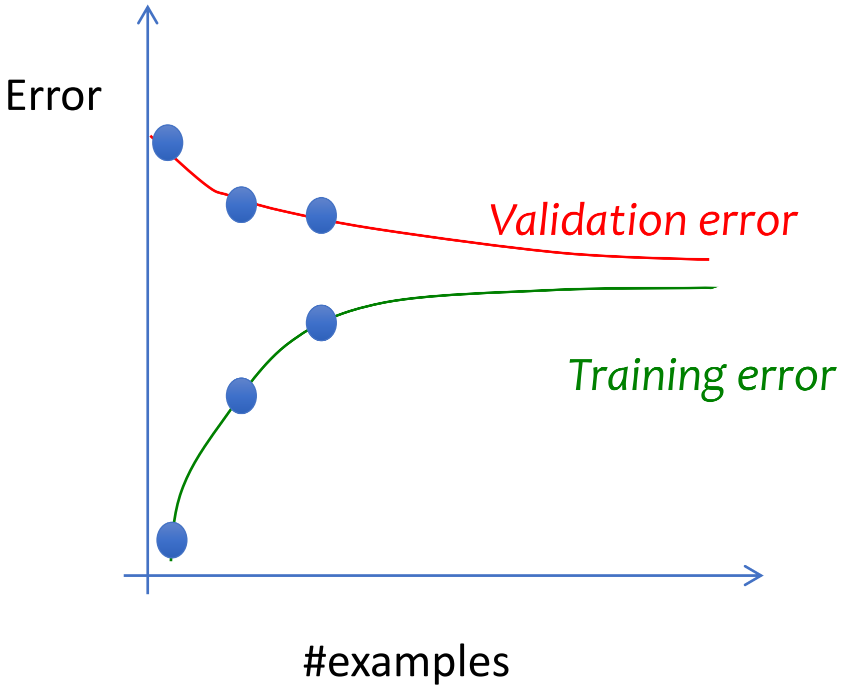


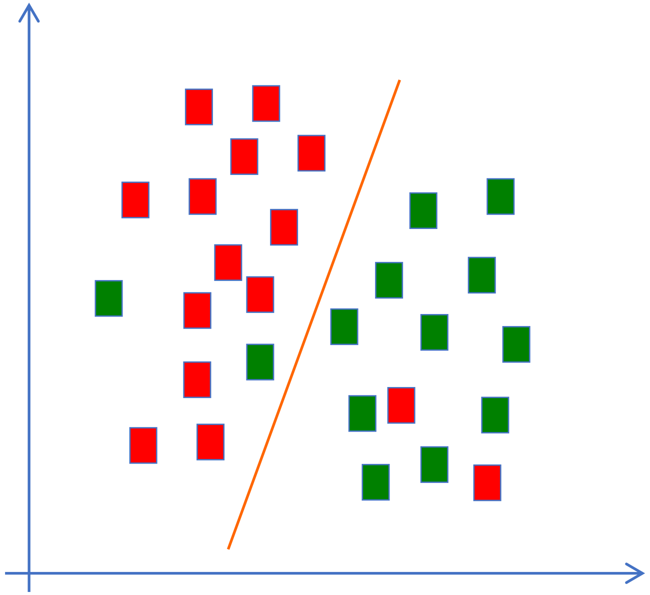



On the other hand, with more data the **validation error is likely to decrease**

# Learning Curve

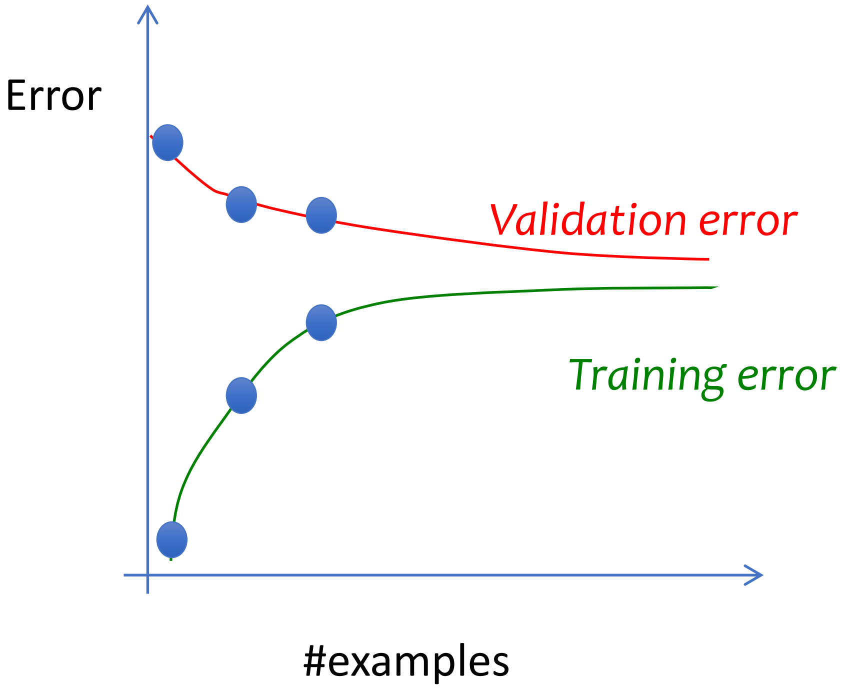
**High bias and high variance models react differently when presented with more examples**

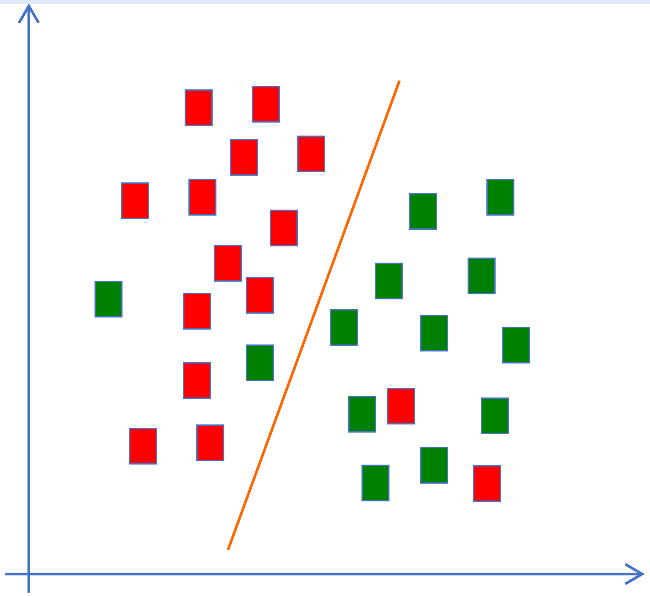


# Learning Curve (**High Bias**)

**High Bias**: We train a simple model *(e.g.,* linear classifier *over the original feature space)*

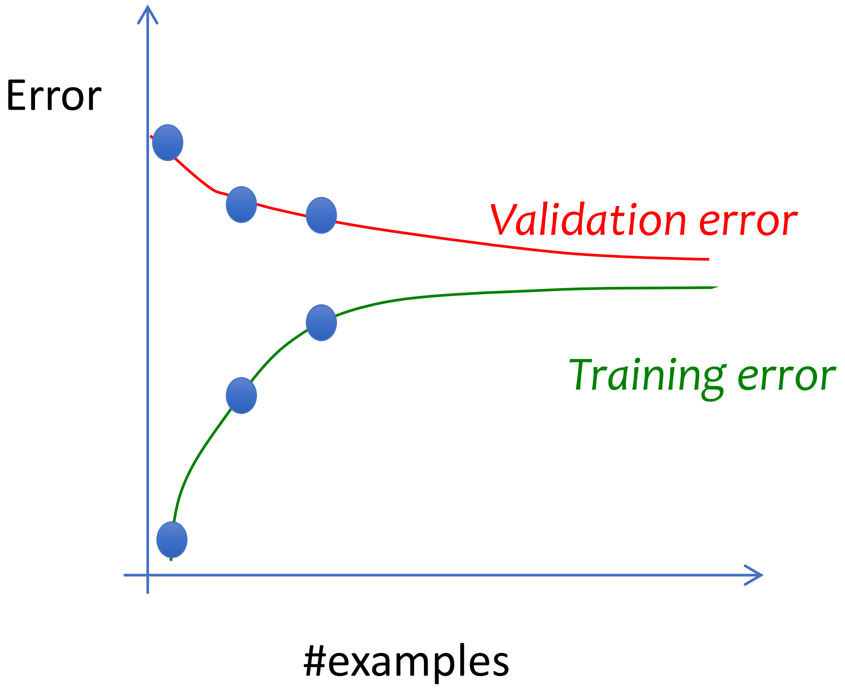


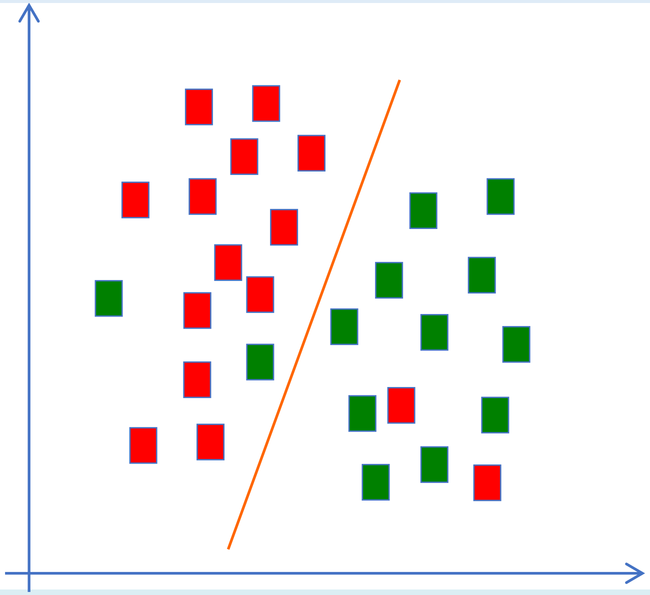



***Training error***: ***increasing the number of examples*** *will* ***increase the training error*** *since the simple model cannot account for the "noise" introduced*

# Learning Curve (**High Bias**)

**High Bias**: We train a simple model *(e.g.,* linear classifier *over the original feature space)*



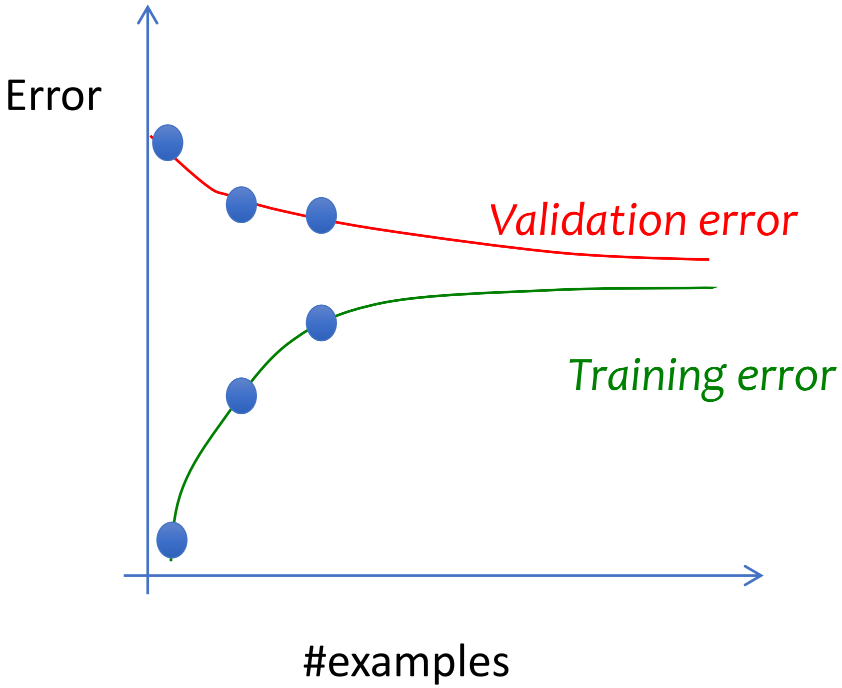



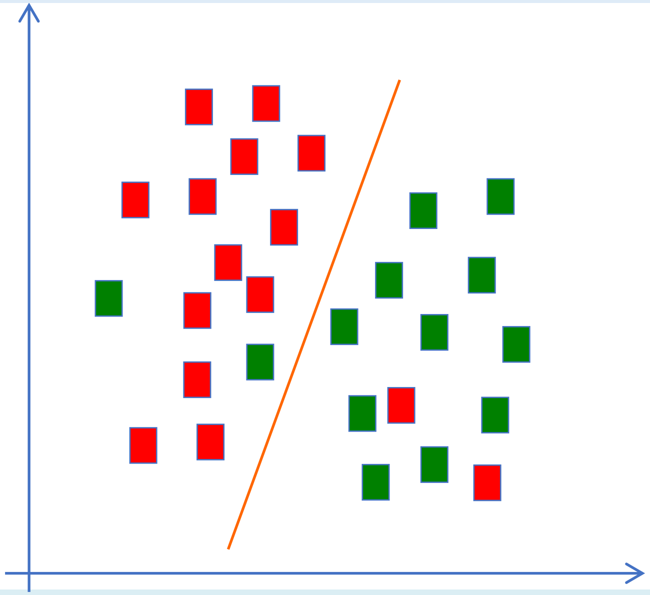
Adding more examples will not change the learned model (just increase the training error)

➔ Adding more examples will not reduce the validation error significantly

# Learning Curve (**High Bias**)

**High Bias**: We train a simple model *(e.g.,* linear classifier *over the original feature space)*

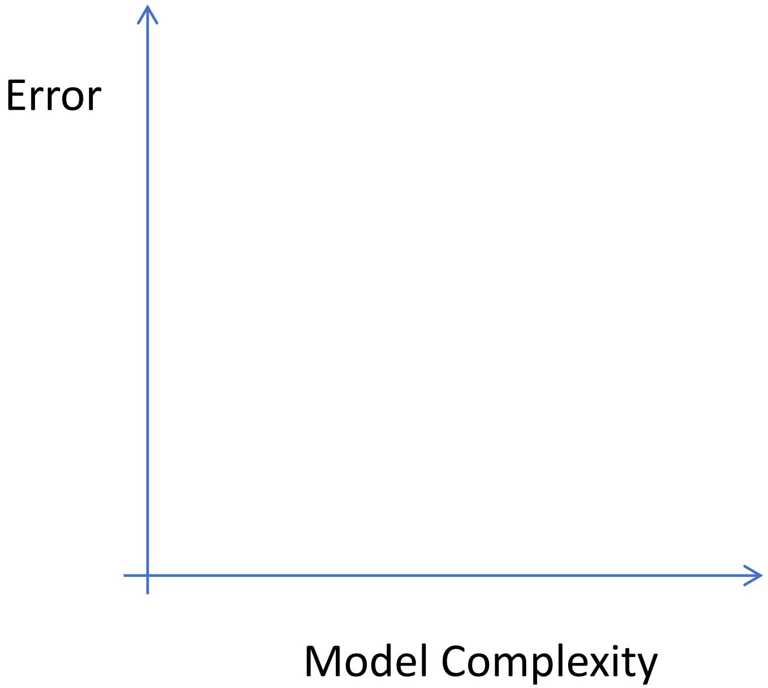


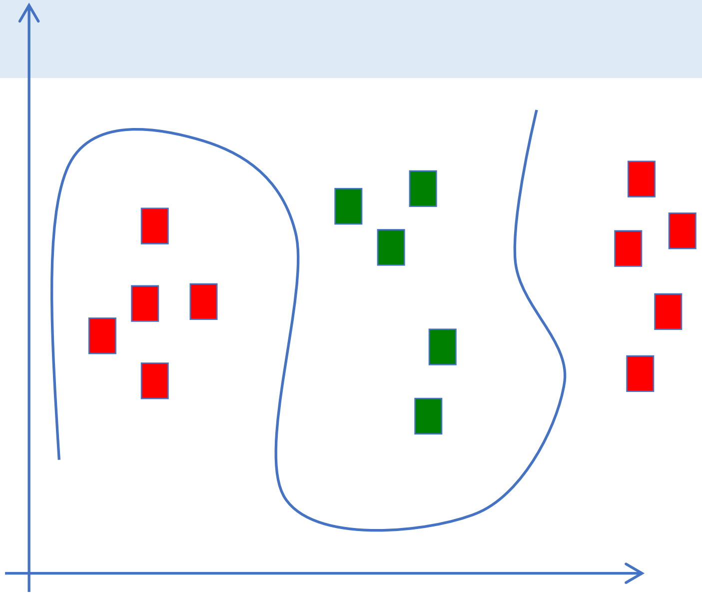



- ➔ Simple hypotheses converge quickly (converges after a few examples, adding more will not help)
- ➔ Feature engineering: carefully selecting a **subset** of features **will not help**; **instead add more relevant attributes**!

# Learning Curve (High Variance)

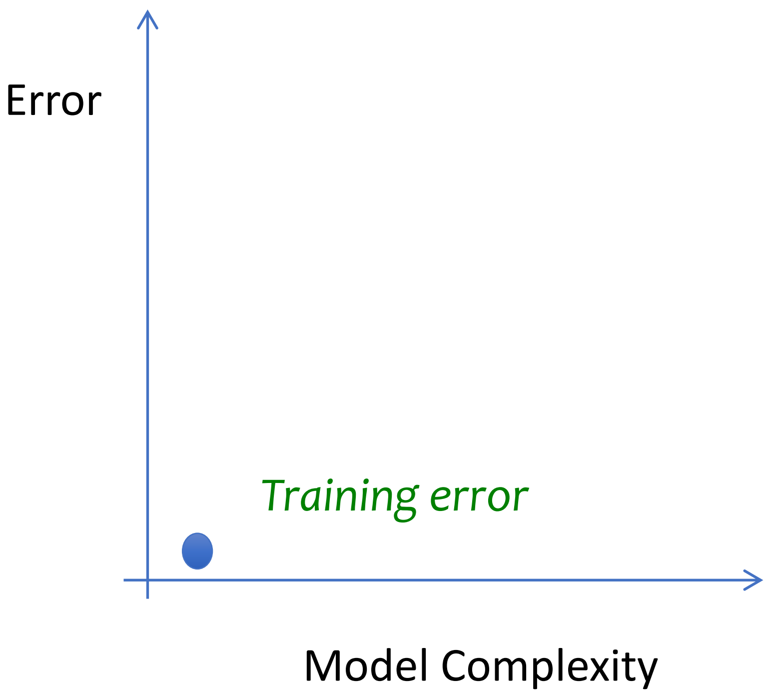
In the **high variance** case, let's assume we blow up the feature space, and we have a very expressive function (e.g., high degree polynomial kernel)

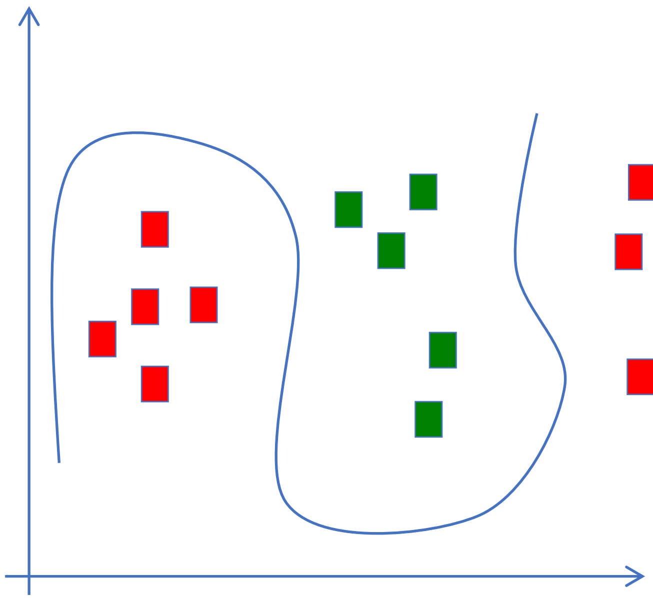


# Learning Curve (High Variance)

In the high variance case, let’s assume we blow up the feature space, and we have a very expressive function (e.g., high degree polynomial kernel)

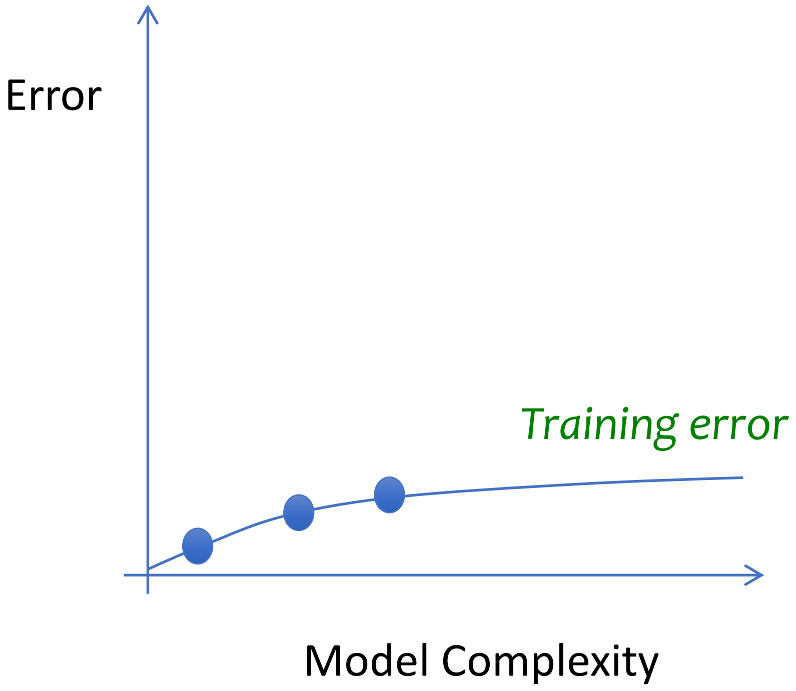


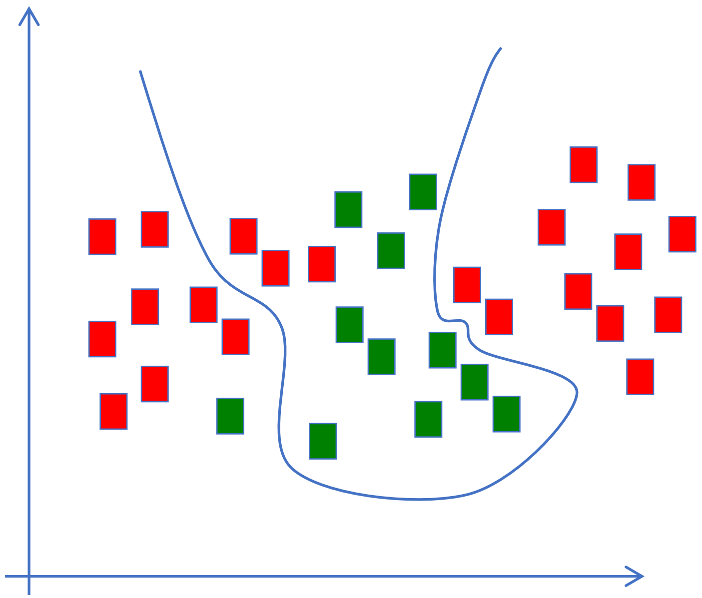



When we have few examples, the model can easily overfit and have a small training error

# Learning Curve (High Variance)

In the high variance case, let's assume we blow up the feature space, and we have a very expressive function (e.g., high degree polynomial kernel)

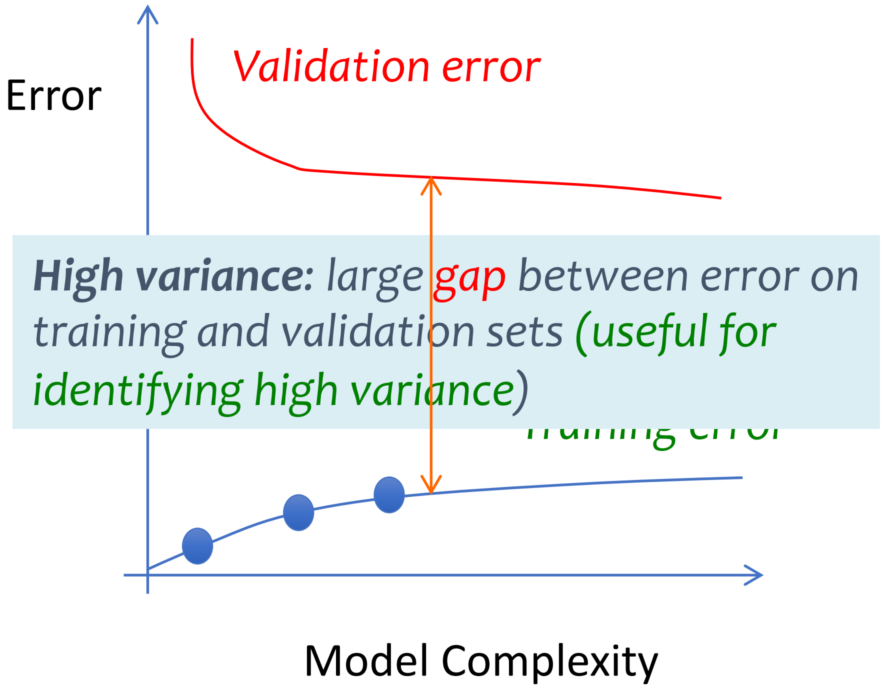


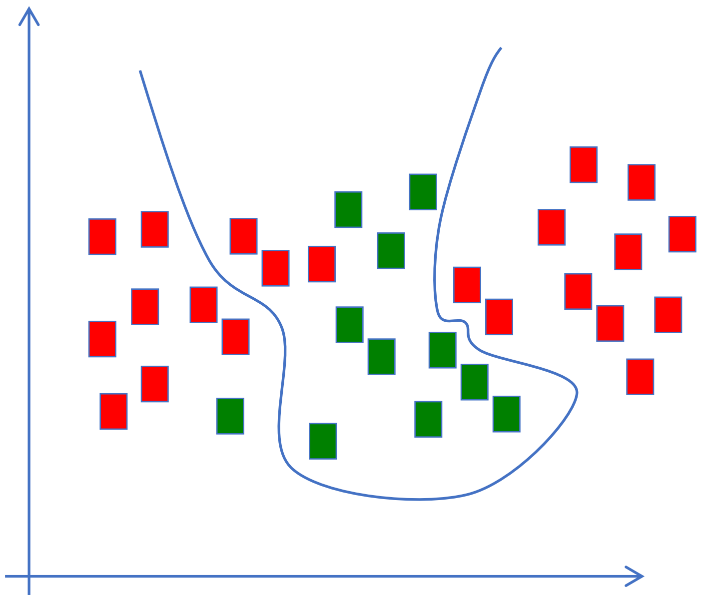
As we add more examples, it might be more difficult to fit the data, **so the training error will increase** (the model might not account for all the "noise")

# Learning Curve (High Variance)

In the high variance case, let's assume we blow up the feature space, and we have a very expressive function (e.g., high degree polynomial kernel)



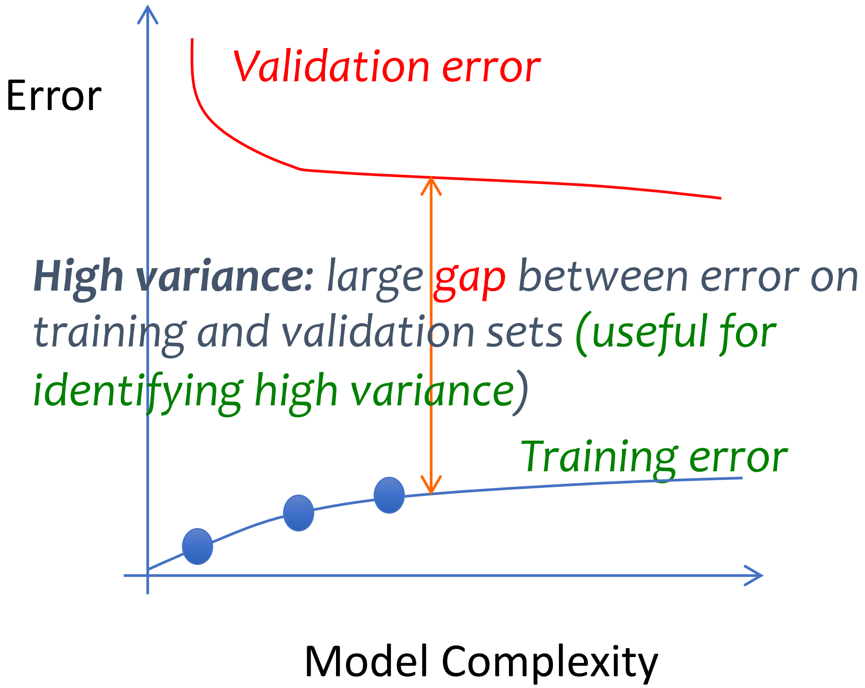

***High variance:*** *large gap between error on training and validation sets (useful for identifying high variance)*



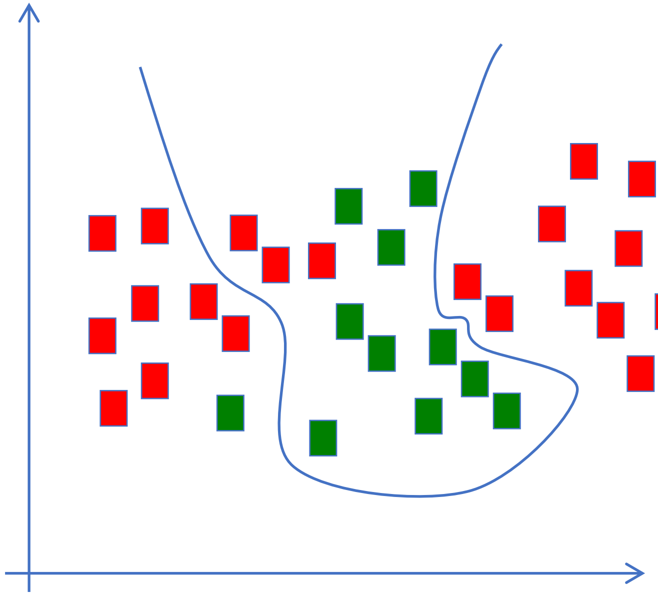
The validation error will be high (the model is overfitting)

# Learning Curve (High Variance)

In the high variance case, let's assume we blow up the feature space, and we have a very expressive function (e.g., high degree polynomial kernel)




***High variance:** large gap between error on training and validation sets (useful for identifying high variance)*



1) **Adding more examples** will change the resulting classifier and decrease the gap *(reduce validation error!)*
2) Simplifying the model (less features) might also help

# Summary

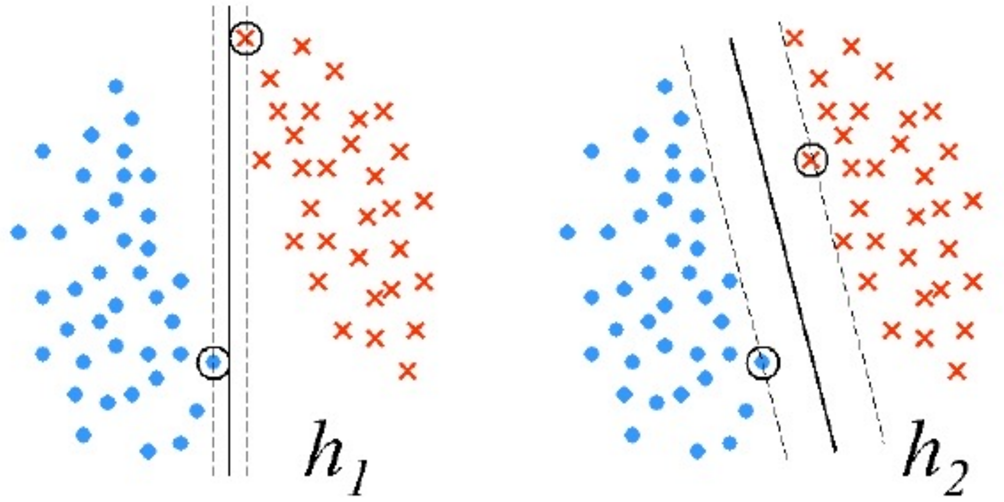
- Bias/Variance: convenient way to analyze a learning system performance
  - Identify underfitting/overfitting
  - Both high bias and high variance can happen

# Reminder: Margin of a classifier

- Distance between a separator (hyperplane) and an example (point)

$$\frac{y(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})}{||\boldsymbol{w}||}$$

- Margin: the value that minimizes that distance for a given dataset.
- Larger margin can be indicative of better generalization



# Hard SVM Intuition

The margin of a classifier: *the distance of the nearest point*

*Recall:* $\dfrac{y(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})}{||\boldsymbol{w}||}$

*We want to find the max margin classifier:* $\mathrm{argmax}_w\, [y(w^T x) \;/\; ||w||]$

If we fix $||w|| = 1$, we can focus on maximizing the functional margin

$$w^* = \mathrm{argmax}_{||w||=1}\, \min_{(x,y) \in S}\, \overbrace{y\,(w^T x)}$$

*Or, fix the functional margin* $y(w^T x) \geq 1$, *and focus on minimizing* $||w||$

$$w^* = \mathrm{argmin}\, ||w||$$
$$\text{s.t. } y(w^T x) \geq 1$$

# Hard SVM Optimization

- We have shown that the sought-after weight vector **w** is the solution of the following optimization problem:

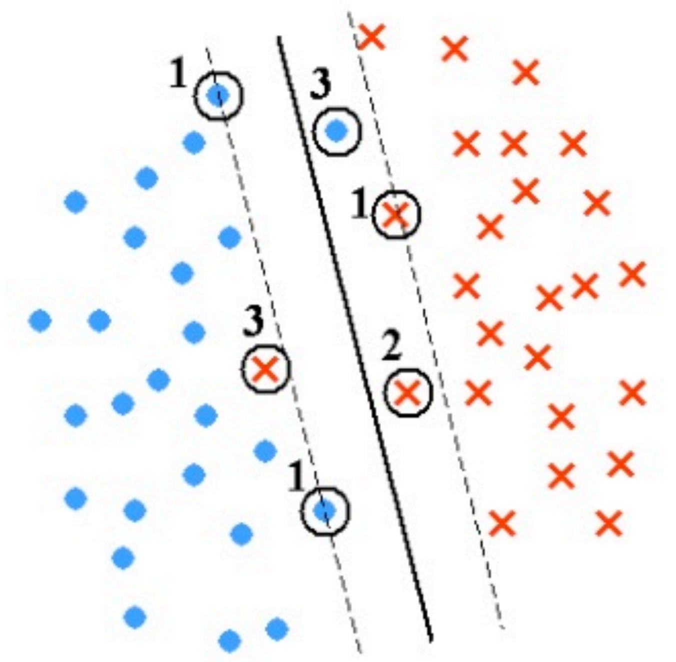
**SVM Optimization:**

Minimize: $\frac{1}{2}\, ||w||^2$

Subject to: $\forall\,(x,y) \in S$: $\quad y\, w^T x \geq 1$

- This is an optimization problem in (n+1) variables, with |S|=m inequality constraints.

# Visualizing Solution in the non-Separable Case




1. Margin support vectors $\xi_i = 0$ Correct
2. Non-margin support vectors $\xi_i < 1$ Correct (in margin)
3. Non-margin support vectors $\xi_i > 1$ Error

# Soft SVM

- Notice that the relaxation of the constraint: $y_i\, w_i^t x_i \geq 1$ can be done by introducing a slack variable $\xi$ (per example) and requiring:

$$y_i\, w_i^t x_i \geq 1 - \xi_i \quad ; \quad \xi_i \geq 0$$

- Now, we want to solve:

$$\text{Min } ½\, ||w||^2 + c \sum_i \xi_i \quad \text{subject to } \xi_i \geq 0$$

- ***Which can be written as:***

$$\text{Min } ½\, ||w||^2 + c \sum_i \max(0,\, 1 - y_i\, w_i^t x_i).$$

# SVM Objective Function

*General Form of a learning algorithm:*

- **Minimize** empirical loss, and **Regularize** (to avoid over fitting)

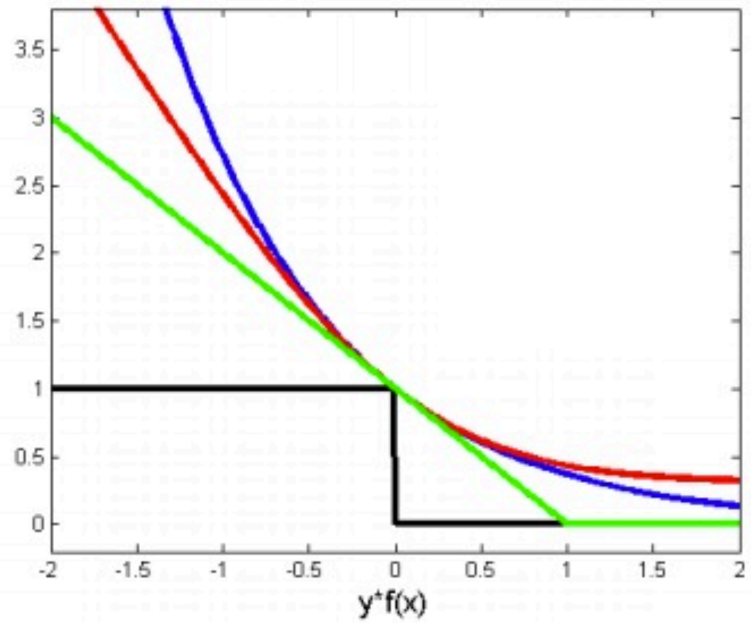
$$\text{Min} \ \underbrace{\tfrac{1}{2}\,||w||^2}_{\text{Regularization term}} + \ c \ \underbrace{\Sigma \max(0, 1 - y_i\, wx_i)}_{\text{Empirical loss}}$$

Can be replaced by other **regularization functions**

Can be replaced by other **loss functions**

# Surrogate Loss functions

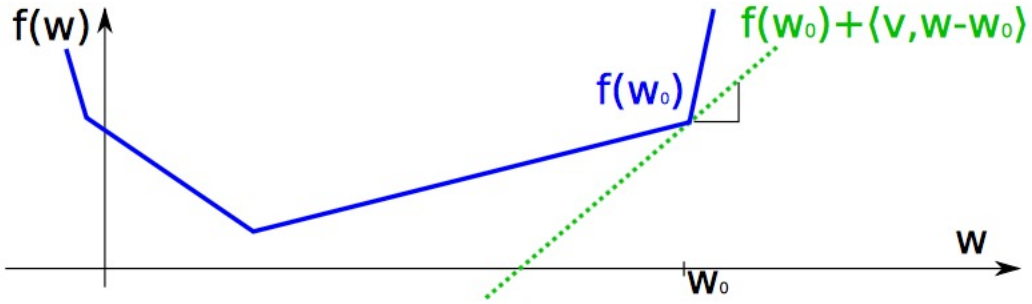
- Surrogate loss function: smooth approximation to the 0-1 loss
  - Upper bound to 0-1 loss



# Subgradient descent

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of $f$ at $w_0$, if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$

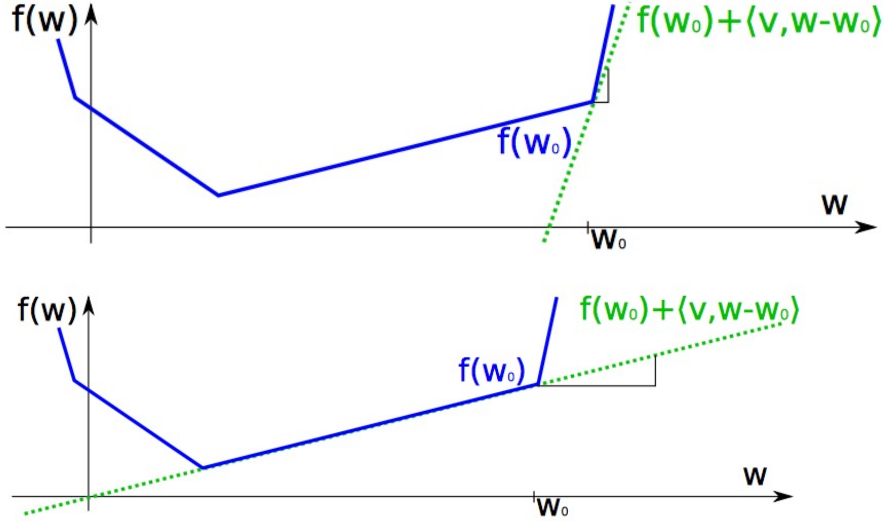



Slides by Sebastian Nowozin and Christoph H. Lampert “structured models in computer vision” tutorial CVPR 2011

# Subgradient descent

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of $f$ at $w_0$, if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



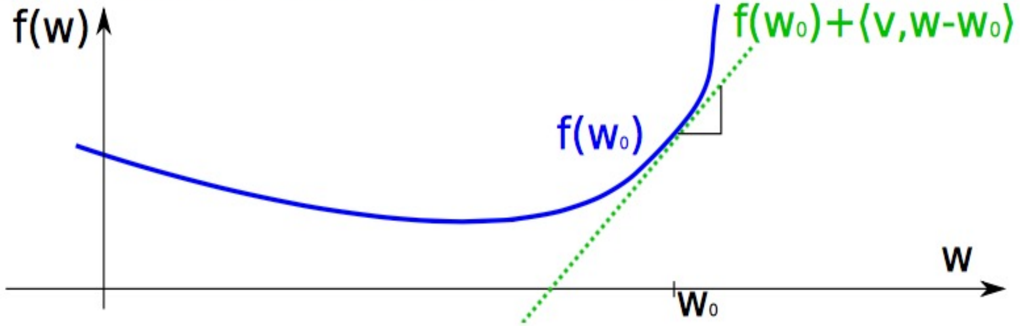

Slides by Sebastian Nowozin and Christoph H. Lampert “structured models in computer vision” tutorial CVPR 2011

# Subgradient descent

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of $f$ at $w_0$, if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$

For differentiable $f$, the gradient $v = \nabla f(w_0)$ is the only subgradient.




Slides by Sebastian Nowozin and Christoph H. Lampert "structured models in computer vision" tutorial CVPR 2011

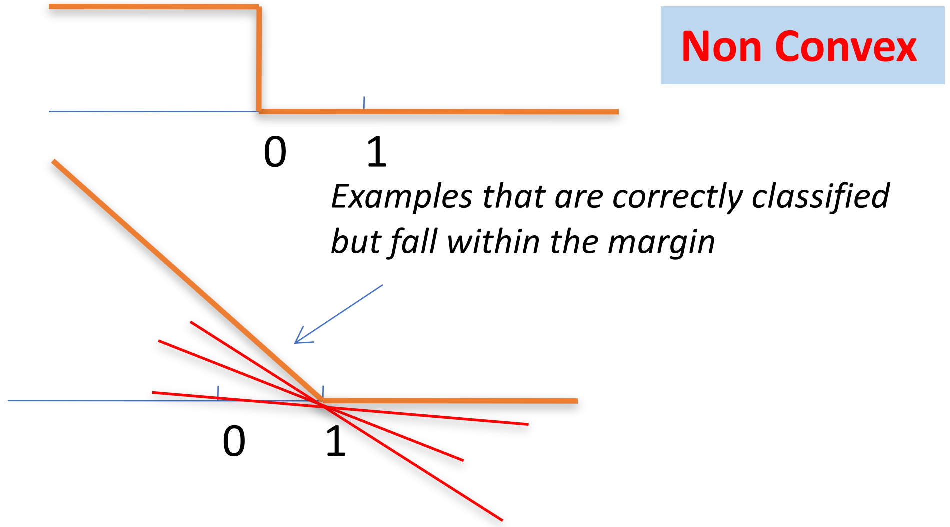
# Sub-Gradient

**Standard 0/1 loss**

Penalizes all incorrectly classified examples with the same amount

**Hinge loss**

Penalizes incorrectly classified examples and correctly classified examples that lie within the margin




**Convex,**
**but not differentiable at x=1**

**Solution:** *subgradient*

*The* ***sub-gradient*** *of a function c at $x_0$ is any vector v such that:* $\forall x : c(x) - c(x_0) \geq v \cdot (x - x_0).$

*At* ***differentiable*** *points this set only contains the gradient at $x_0$*

***Intuition****: the set of all tangent lines (lines under c, touching c at $x_0$)*

$$
\begin{aligned}
&\boldsymbol{\partial}_w \max\{0, 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\} \\
&= \boldsymbol{\partial}_w \begin{cases} 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases} \\
&= \begin{cases} \boldsymbol{\partial}_w 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ \boldsymbol{\partial}_w y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases} \\
&= \begin{cases} \boldsymbol{0} & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ y_n \boldsymbol{x}_n & \text{otherwise} \end{cases}
\end{aligned}
$$

Image from CIML

# Multiclass SVM

- **Single classifier optimizing a global objective**
  - *Extend the SVM framework to the multiclass settings*

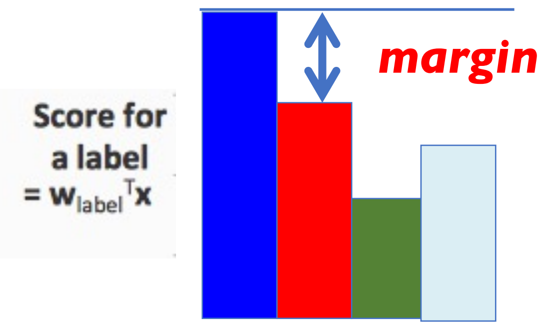
- **Binary SVM:**
  - *Minimize ||W|| such that the closest points to the hyperplane have a score of +/- 1*

- **Multiclass SVM**
  - *Each label has a* ***different*** *weight vector*
  - *Maximize* ***multiclass margin***

# Margin in the Multiclass case

***Revise the definition for the multiclass case:***

- The difference between the score of the correct label and the scores of competing labels




Colors indicate different labels

<u>**SVM Objective**</u>: Minimize total norm of weights s.t. the ***true label is scored at least 1 more than the second best*****.**

# Hard Multiclass SVM

Regularization

$$\min_{\mathbf{w}_1,\mathbf{w}_2,\cdots,\mathbf{w}_K} \quad \frac{1}{2}\sum_k \mathbf{w}_k^T\mathbf{w}_k$$

$$\text{s.t.} \quad \mathbf{w}_{\mathbf{y}_i}^T\mathbf{x} - \mathbf{w}_k^T\mathbf{x} \geq 1 \qquad \forall(\mathbf{x}_i,\mathbf{y}_i)\in D,$$
$$k\in\{1,2,\cdots,K\}, k\neq \mathbf{y}_i,$$

The score of the true label has to be higher than 1, for any label

K. Crammer, Y. Singer: "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines", JMLR, 2001

# Soft Multiclass SVM

**Regularizer**

**Slack Variables**

$$\min_{\mathbf{w}_1,\mathbf{w}_2,\cdots,\mathbf{w}_K,\xi} \quad \frac{1}{2}\sum_k \mathbf{w}_k^T\mathbf{w}_k + C \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in D} \xi_i$$

$$\text{s.t.} \quad \mathbf{w}_{\mathbf{y}_i}^T\mathbf{x} - \mathbf{w}_k^T\mathbf{x} \geq 1 - \xi_i, \quad \forall(\mathbf{x}_i,\mathbf{y}_i)\in D,$$

$$k \in \{1,2,\cdots,K\}, k \neq \mathbf{y}_i,$$

$$\xi_i \geq 0, \quad \forall i.$$

**The score of the true label should have a margin of 1-$\xi_i$**

**Relax hard constraints using slack variables**

**Positive slack**

K. Crammer, Y. Singer: "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines", JMLR, 2001

# Alternative Notation

- For examples with label *i* we want: $w_i^T x > w_j^T x$
- **Alternative notation**: *Stack all weight vectors*

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_K \end{bmatrix}_{nK \times 1} \qquad \phi(\mathbf{x}, i) = \begin{bmatrix} \mathbf{0}_n \\ \vdots \\ \mathbf{x} \\ \vdots \\ \mathbf{0}_n \end{bmatrix}_{nK \times 1}$$

**x** in the i[th] block, zeros everywhere else

$\mathbf{w}^T \phi(\mathbf{x}, i) > \mathbf{w}^T \phi(\mathbf{x}, j)$ *is equivalent to* $w_i^T x > w_j^T x$

# Multiclass classification so far

- **Lea**

Solve: $$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\xi^n$$

subject to, for $i = 1, \dots, n$,

$$\langle w, \phi(x^n, y^n)\rangle - \langle w, \phi(x^n, y)\rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

- **Prediction**

$$f(x) = \text{argmax}_{y \in \mathcal{Y}} \ \langle w, \phi(x, y)\rangle$$

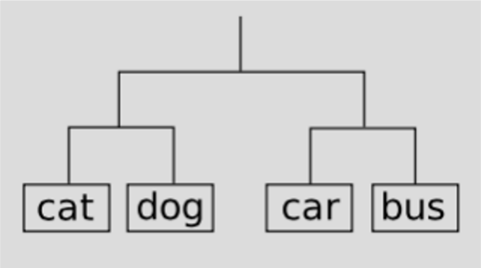
# Cost Sensitive Multiclass Classification

- Sometime we are willing to “tolerate” some mistakes more than others




# Cost Sensitive Multiclass Classification

- We can think about it as a hierarchy:




- Define a distance metric:
  - Δ(y,y') = tree distance between y and y'

We would like to incorporate that into our learning model

Solve: $$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \xi^n$$

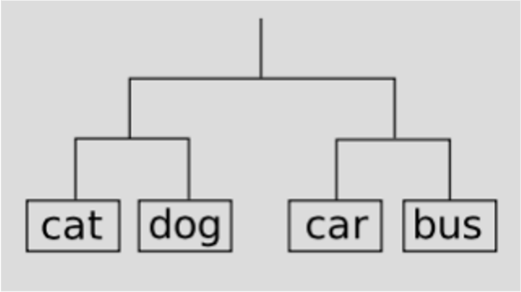
subject to, for $i = 1, \ldots, n$,

$$\langle w, \phi(x^n, y^n)\rangle - \langle w, \phi(x^n, y)\rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

what should we change?

# Cost Sensitive Multiclass Classification

- We can think about it as a hierarchy:
- Define a distance metric:
  - Δ(y,y’) = tree distance between y and y’




We would like to incorporate that into our learning model

$\Delta(y^n, y)$

Solve: $$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \xi^n$$

subject to, for $i = 1, \ldots, n$,

$$\langle w, \phi(x^n, y^n)\rangle - \langle w, \phi(x^n, y)\rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

# Cost Sensitive Multiclass Classification

Solve: $$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\xi^n$$

subject to, for $i = 1, \ldots, n$,

$$\langle w, \phi(x^n, y^n)\rangle - \langle w, \phi(x^n, y)\rangle \geq \Delta(y^n, y) - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

*Instead, we can have an unconstrained version -*

Solve: $$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} l(\mathbf{w};(\mathbf{x}^n, y^n))$$

**Question**: What is sub-gradient of this loss function?

$$l(\mathbf{w};(\mathbf{x}^n, y^n)) = \max_{y' \in Y} \Delta(y, y') - \langle \mathbf{w}, \phi(\mathbf{x}^n, y^n)\rangle + \langle \mathbf{w}, \phi(\mathbf{x}^n, y')\rangle$$

# Subgradient for the MC case

**Computing a subgradient:**

$$\min_w \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$

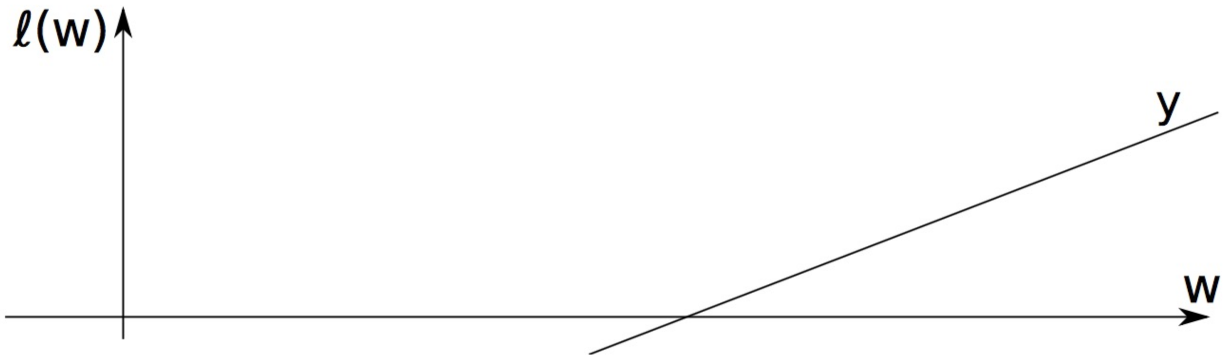
# Subgradient for the MC case

**Computing a subgradient:**

$$\min_w \ \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$




For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

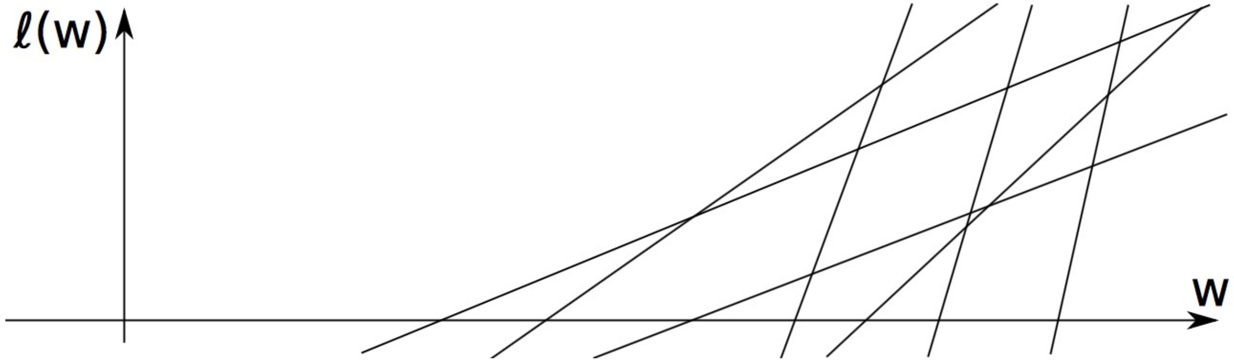
# Subgradient for the MC case

**Computing a subgradient:**

$$\min_w \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$




For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

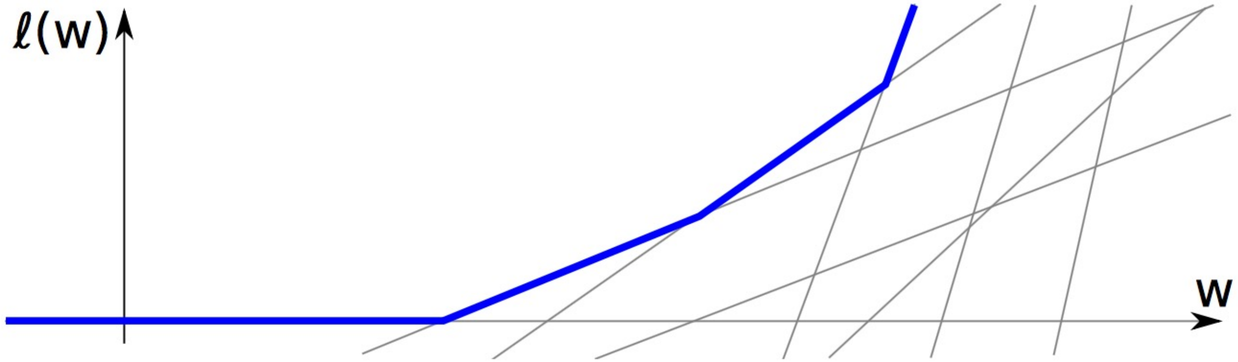
# Subgradient for the MC case

**Computing a subgradient:**

$$\min_w \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$




$\ell(w) = \max_y \ell_y(w)$: maximum over all $y \in \mathcal{Y}$.

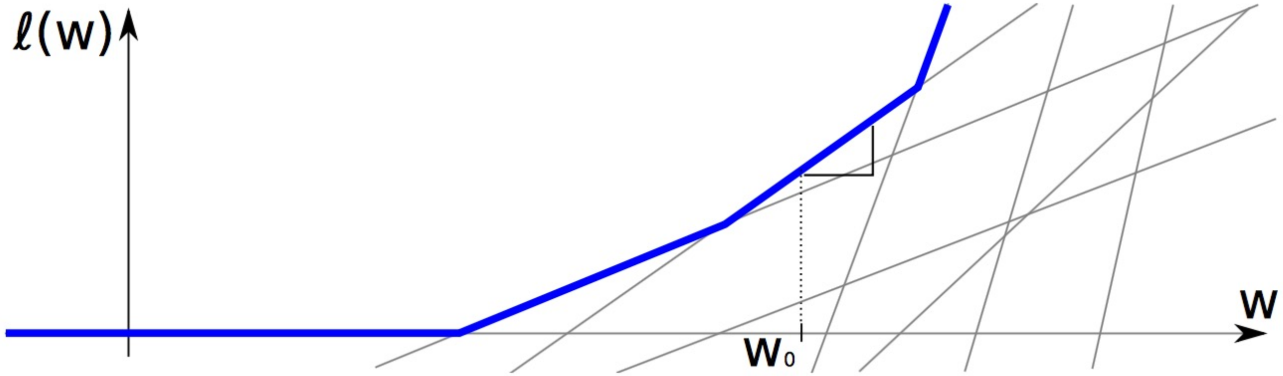
# Subgradient for the MC case

**Computing a subgradient:**

$$\min_w \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell_y^n(w)$, and

$$\ell_y^n(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$




Subgradient of $\ell^n$ at $w_0$: find maximal (active) $y$, use $v = \nabla \ell_y^n(w_0)$.

# Subgradient descent for the MC case

**Subgradient Descent S-SVM Training**

**input** training pairs $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
**input** feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer $C$,
**input** number of iterations $T$, stepsizes $\eta_t$ for $t = 1, \ldots, T$

1: $w \leftarrow \vec{0}$
2: **for** t=1,. . . ,T **do**
3:     **for** i=1,. . . ,n **do**
4:         $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$
5:         $v^n \leftarrow \phi(x^n, \hat{y}) - \phi(x^n, y^n)$
6:     **end for**
7:     $w \leftarrow w - \eta_t (w - \frac{C}{N} \sum_n v^n)$
8: **end for**

**output** prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

Observation: each update of $w$ needs 1 argmax-prediction per example.

# Stochastic SbGD for the MC case

**Stochastic Subgradient Descent S-SVM Training**

**input** training pairs $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
**input** feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer $C$,
**input** number of iterations $T$, stepsizes $\eta_t$ for $t = 1, \dots, T$

1: $w \leftarrow \vec{0}$
2: **for** t=1,...,T **do**
3: $(x^n, y^n) \leftarrow$ randomly chosen training example pair
4: $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$
5: $w \leftarrow w - \eta_t (w - \frac{C}{N}[\phi(x^n, \hat{y}) - \phi(x^n, y^n)])$
6: **end for**

**output** prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

**Question:** *What is the difference between this algorithm and the perceptron variant for multiclass classification?*

# From multiclass to structures

- So far the number of classes was small.
- **That's not always the case…**

Label = DOG

Label = 2 DOGS

Label = 3 DOGS

***Can we still think about this scenario as multiclass classification?***

Solve: $$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \xi^n$$

subject to, for $i = 1, \ldots, n$,

$$\langle w, \phi(x^n, y^n)\rangle - \langle w, \phi(x^n, y)\rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

# Summary

- Introduced an optimization framework for learning:
  - Minimization problem
  - Objective: data loss term and regularization cost term
  - Separate learning objective from learning algorithm
    - Many algorithms for minimizing a function
- Can be used for regression and classification
  - Different loss function
  - GD and SGD algorithms
- Classification: use surrogate loss function
  - Smooth approximation to 0-1 loss