Machine Learning

CMPT 726

Mo Chen
SFU School of Computing Science
2022-11-22

Why Machine Learning?

- AI is hard because the inner workings of our brain are not well understood.
- We don’t know *how* our brain converts input to output, so we can’t write a program to do so.
- While we don’t know *how* our brain converts input to output, we know what the output should be for every input.
- We can use this knowledge to teach the machine.
- The goal of machine learning is to predict outputs from inputs.

Review:
General Machine Learning Principles

General Framework

Given: Dataset of input-output pairs

Goal: Predict the output corresponding to an unseen input

How do we achieve this?

- Devise a model, which takes in the input and produces a prediction
- Devise a loss function, which measures how good the predictions are compared to the target output
- Train the model, that is, minimize the loss function w.r.t. the model parameters
- Test the model, that is, use the model with optimal parameters to generate predictions on unseen inputs

General Framework

Given: Dataset of input-output pairs

Goal: Predict the output corresponding to an unseen input

How do we achieve this?

- **Devise a model**, which takes in the input and produces a prediction
- Devise a loss function, which measures how good the predictions are compared to the target output
- Train the model, that is, minimize the loss function w.r.t. the model parameters
- Test the model, that is, use the model with optimal parameters to generate predictions on unseen inputs

Model

Two kinds of models: deterministic models and probabilistic models

Deterministic model: Given an input, produces an output

- Example: $\hat{y} = \vec{w}^{\top}\vec{x}$

Probabilistic model: Given an input, produces a distribution over possible outputs.

- Example: $y|\vec{x}, \vec{w}, \sigma \sim \mathcal{N}(\vec{w}^{\top}\vec{x}, \sigma^2)$

General Framework

Given: Dataset of input-output pairs

Goal: Predict the output corresponding to an unseen input

How do we achieve this?

- Devise a model, which takes in the input and produces a prediction
- Devise a loss function, which measures how good the predictions are compared to the target output
- Train the model, that is, minimize the loss function w.r.t. the model parameters
- Test the model, that is, use the model with optimal parameters to generate predictions on unseen inputs

General Framework

Given: Dataset of input-output pairs

Goal: Predict the output corresponding to an unseen input

How do we achieve this?

- Devise a model, which takes in the input and produces a prediction
- **Devise a loss function**, which measures how good the predictions are compared to the target output
- Train the model, that is, minimize the loss function w.r.t. the model parameters
- Test the model, that is, use the model with optimal parameters to generate predictions on unseen inputs

Loss Function

For a deterministic model: Can be any function that assigns high values to incorrect outputs and low values to correct outputs.

For a probabilistic model:

- Maximum likelihood (MLE): Loss function is the negative log-likelihood $\log p\left(\mathcal{D}\middle|\vec{\theta}\right)$
- Maximum a posteriori (MAP): Loss function is the negative log-posterior $\log p\left(\vec{\theta}\middle|\mathcal{D}\right) = \log\left(\frac{p(\vec{\theta})p\left(\mathcal{D}\middle|\vec{\theta}\right)}{p(\mathcal{D})}\right)$

General Framework

Given: Dataset of input-output pairs

Goal: Predict the output corresponding to an unseen input

How do we achieve this?

- Devise a model, which takes in the input and produces a prediction
- Devise a loss function, which measures how good the predictions are compared to the target output
- Train the model, that is, minimize the loss function w.r.t. the model parameters
- Test the model, that is, use the model with optimal parameters to generate predictions on unseen inputs

General Framework

Given: Dataset of input-output pairs

Goal: Predict the output corresponding to an unseen input

How do we achieve this?

- Devise a model, which takes in the input and produces a prediction
- Devise a loss function, which measures how good the predictions are compared to the target output
- **Train the model**, that is, minimize the loss function w.r.t. the model parameters
- Test the model, that is, use the model with optimal parameters to generate predictions on unseen inputs

Training

Training involves finding the model parameters that minimize the loss. Several approaches:

- Set the gradient to zero and solve for the optimal parameters analytically.
  - Examples: OLS, ridge regression, multiple output linear regression
- If there is no closed-form solution for the optimal parameters:
  - If optimization problem is unconstrained: use iterative gradient-based optimization methods.
    - If loss is convex and Lipschitz, guaranteed to find the global minimum.
    - If loss is non-convex but Lipschitz, may get stuck in local minimum and gradients may vanish.
    - If loss is non-Lipschitz, gradients may explode, leading to divergence.

Training

- If optimization problem is constrained:
  - Can check if Slater’s condition holds.
    - If it holds, strong duality holds. So can choose to take the dual and solve the dual instead.
  - Sometimes can massage the problem into an unconstrained problem.
  - If this is not possible, can use dedicated solvers when the objective and constraints are of a particular form (e.g.: linear, quadratic, etc.).

General Framework

Given: Dataset of input-output pairs

Goal: Predict the output corresponding to an unseen input

How do we achieve this?

- Devise a model, which takes in the input and produces a prediction
- Devise a loss function, which measures how good the predictions are compared to the target output
- Train the model, that is, minimize the loss function w.r.t. the model parameters
- Test the model, that is, use the model with optimal parameters to generate predictions on unseen inputs

General Framework

Given: Dataset of input-output pairs

Goal: Predict the output corresponding to an unseen input

How do we achieve this?

- Devise a model, which takes in the input and produces a prediction
- Devise a loss function, which measures how good the predictions are compared to the target output
- Train the model, that is, minimize the loss function w.r.t. the model parameters
- **Test the model**, that is, use the model with optimal parameters to generate predictions on unseen inputs

Testing

In some cases, take the model’s output directly as prediction.

- Example: Regression methods

In other cases, need to process the model output.

- Example: Classification methods, where the processing takes the form of thresholding

Model Selection

Often we have a family of related models which differ in hyperparameter values.

In this case, the previous procedure becomes the inner loop; the outer loop selects the best model.

Goal is to minimize the expected error on an **unseen** example.

Bias-Variance Tradeoff

Expected Error = Bias Squared + Variance + Irreducible Error

Bias: Difference between model prediction when trained on the unseen example and the target output

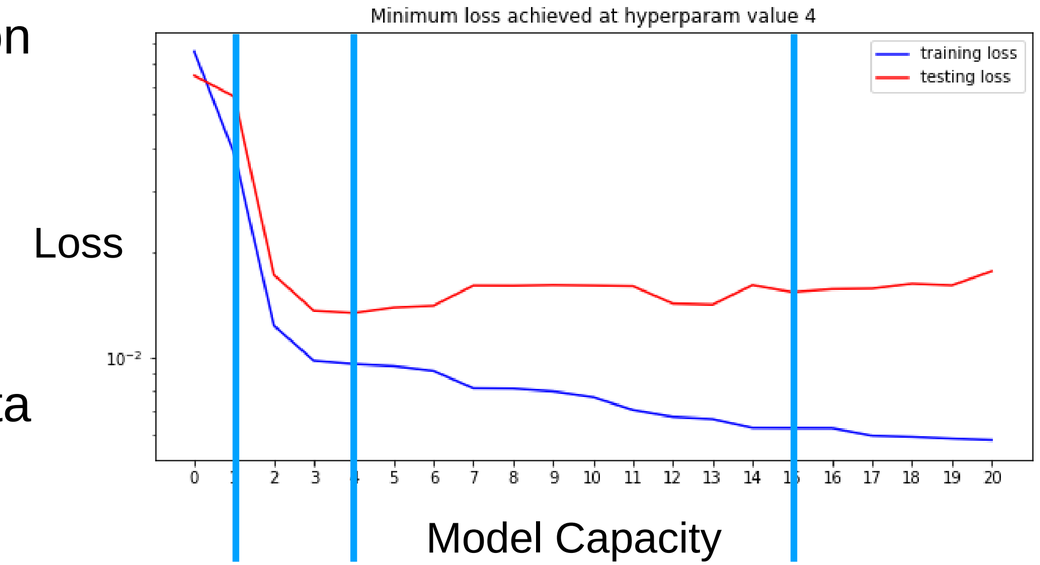
Variance: How much the model prediction varies when trained on different datasets

More expressive models achieve low bias (fits training data better)

Less expressive models achieve low variance (generalizes to unseen data better)

To achieve low expected error, we must strike the right balance in model expressiveness.

Just right



Underfitting

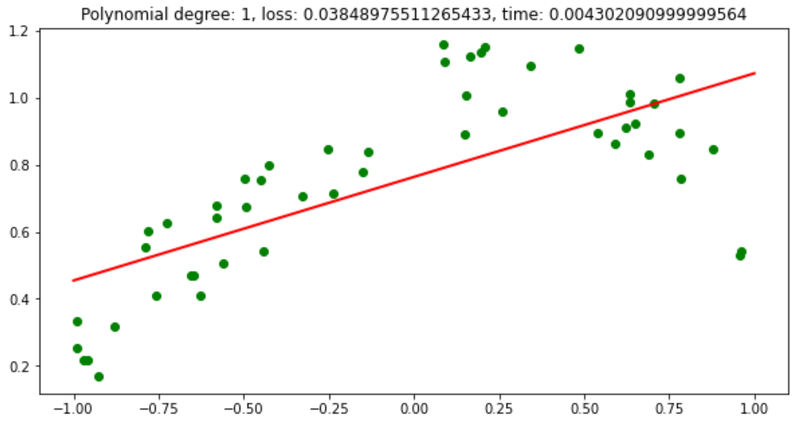
Overfitting

Overfitting / Underfitting

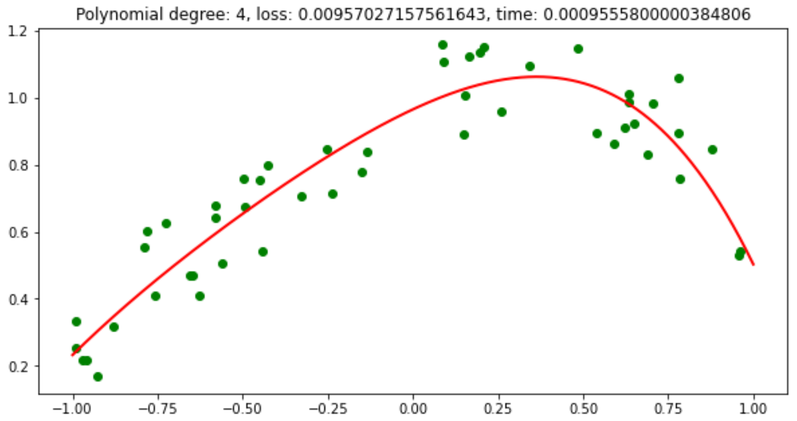
Overfitting: Low bias, high variance

Underfitting: High bias, low variance

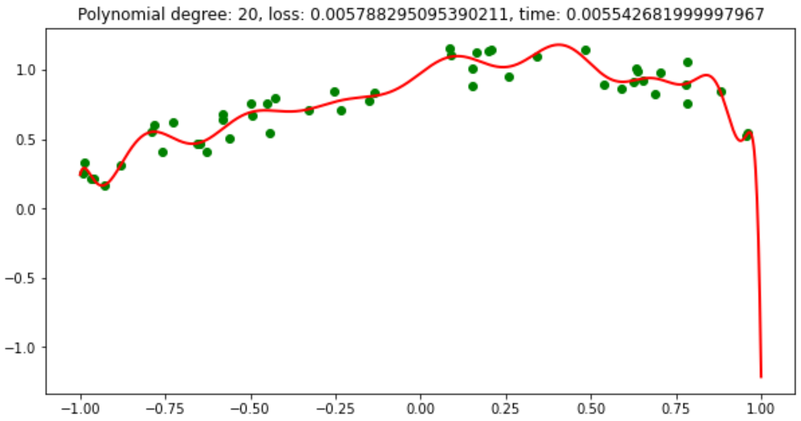
Just right: Low bias, low variance



Underfitting



Just right



Overfitting

Avoiding Overfitting

Three ways:

- Reduce model capacity (e.g.: reduce degree of polynomial features, decrease the number of hidden units in neural networks)
- Add regularizer or increase regularizer strength (e.g.: increase λ in ridge regression)
- Early stopping (e.g.: stopping the training of neural networks when the validation loss starts to increase)

Review:
Machine Learning Methods

Regression vs. Classification

When you are given a machine learning problem, what should you look for?

- The first thing is to decide which kind of machine learning problem it is.

There are two major types of problems:

- Regression problems
- Classification problems

How do you tell?

- If the goal is to predict a **value within a continuous interval**, the problem is a regression problem.
- If the goal is to predict **one of a discrete set of labels**, the problem is a classification problem.

Examples:

- Predicting tomorrow's temperature: regression
- Predicting whether it rains tomorrow: classification

Regression vs. Classification

When you are given a machine learning problem, what should you look for?

- The first thing is to decide which kind of machine learning problem it is.

There are two major types of problems:

- **Regression problems**
- Classification problems

How do you tell?

- If the goal is to predict a **value within a continuous interval**, the problem is a regression problem.
- If the goal is to predict **one of a discrete set of labels**, the problem is a classification problem.

Examples:

- Predicting tomorrow's temperature: regression
- Predicting whether it rains tomorrow: classification

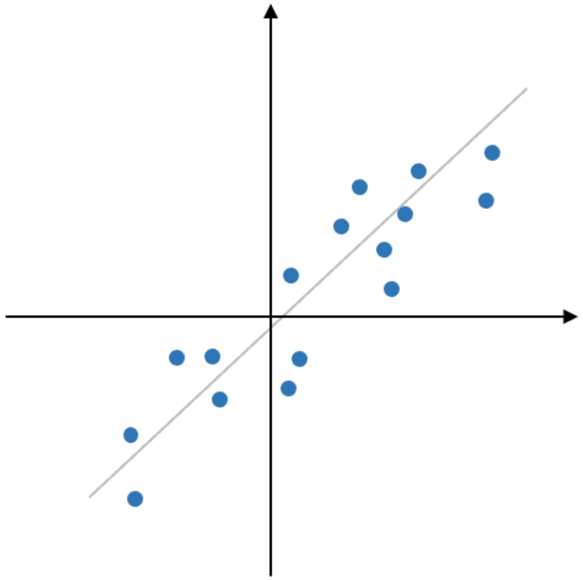
Regression

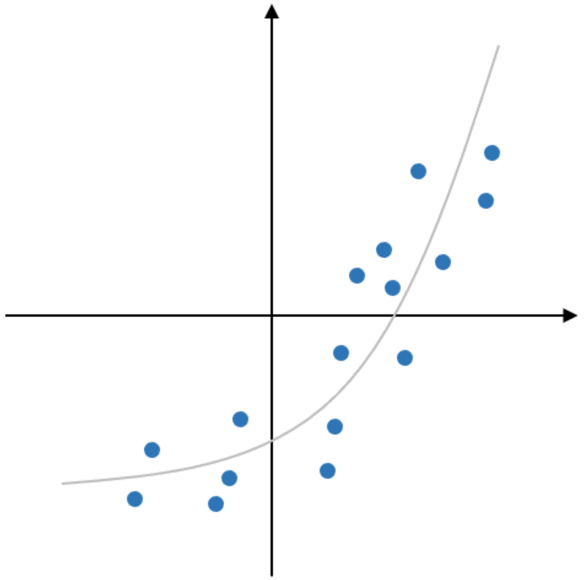
The next step is to decide which model to use. There are two kinds of models:

- Linear regression models
- Non-linear regression models

How do you decide?

- If you expect the label to be linear in the input, use a linear regression model
  - E.g.: if an increase in an input dimension is expected to result in a proportional increase in the label, regardless of the starting value
- If you expect the label to be non-linear in the input, use a non-linear regression model
  - E.g.: if the label can be similar for both extremes of an input dimension
- If you don't know, try both and see which achieves better validation loss. This process is known as **model selection**.





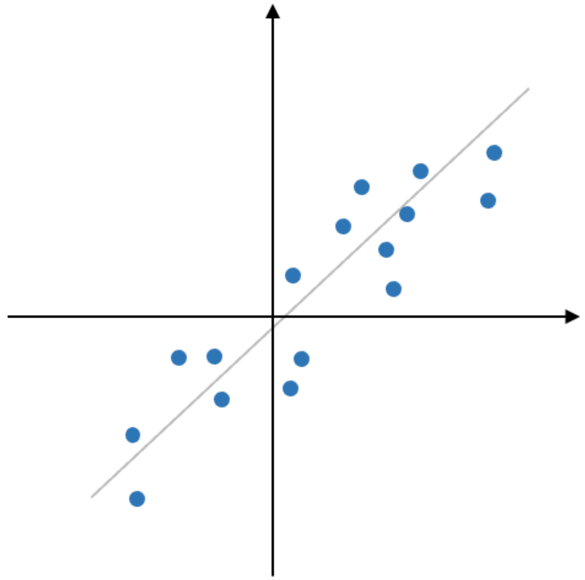
Regression

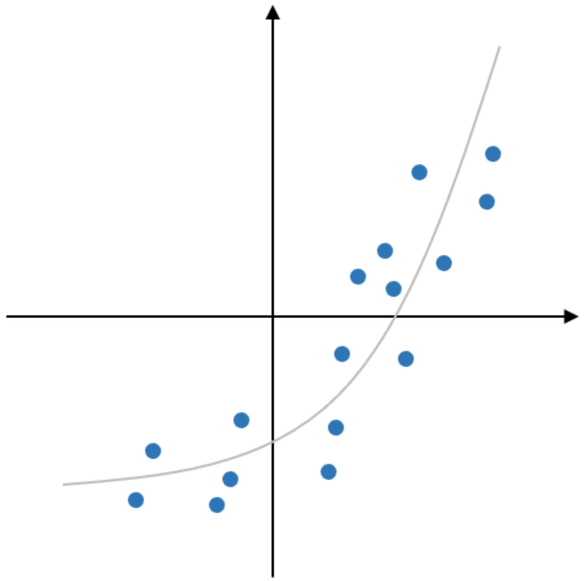
The next step is to decide which model to use. There are two kinds of models:

- **Linear regression models**
- Non-linear regression models

How do you decide?

- If you expect the label to be linear in the input, use a linear regression model
  - E.g.: if an increase in an input dimension is expected to result in a proportional increase in the label, regardless of the starting value
- If you expect the label to be non-linear in the input, use a non-linear regression model
  - E.g.: if the label can be similar for both extremes of an input dimension
- If you don't know, try both and see which achieves better validation loss. This process is known as **model selection**.





Linear Regression

The next step is to decide whether to use:

- Single output linear regression
- Multiple output linear regression

How do you decide?

- If you would like to predict a scalar value, use single output linear regression
- If you would like to predict a vector, use multiple output linear regression

Linear Regression

The next step is to decide whether to use:

- **Single output linear regression**
- Multiple output linear regression

How do you decide?

- If you would like to predict a scalar value, use single output linear regression
- If you would like to predict a vector, use multiple output linear regression

Single Output Linear Regression

Model: $\hat{y} = \vec{w}^{\top}\vec{x}$

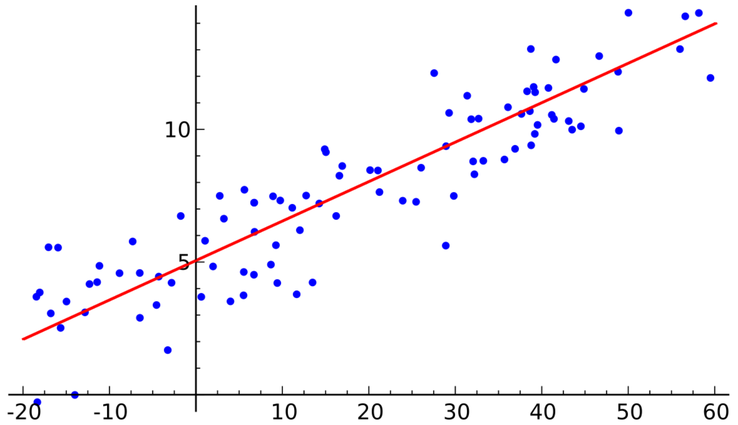
Parameters:$\vec{w}$

Loss:

$$L(\vec{w}) = \sum_{i=1}^{N}(y_i - \vec{w}^{\top}\vec{x}_i)^2 + \lambda\left\|\vec{w}\right\|_2^2 = \left\|\vec{y} - X\vec{w}\right\|_2^2 + \lambda\|\vec{w}\|_2^2$$

Training: Calculate optimal parameters using closed-form formula:

$$\vec{w}^* \coloneqq \arg\min_{\vec{w}} L(\vec{w}) = (X^{\top}X + \lambda I)^{-1}X^{\top}\vec{y}$$



Testing: $\hat{y} = \vec{w}^{*\top}\vec{x}$

Two methods:

- Ordinary least squares (OLS): $\lambda = 0$
- Ridge regression: $\lambda > 0$

OLS vs. Ridge Regression (Deterministic Interpretation)

Model: $\vec{y} = \vec{w}^\top \vec{x}$; Parameters: $\vec{w}$

OLS:

Loss function:

$$L(\vec{w}) = \sum_{i=1}^{N} (y_i - \vec{w}^\top \vec{x}_i)^2$$
$$= \|\vec{y} - X\vec{w}\|_2^2$$

Optimal Parameters:

$$\vec{w}^* := \arg\min_{\vec{w}} L(\vec{w}) = (X^\top X)^{-1} X^\top \vec{y}$$

Ridge Regression:

Loss function:

$$L(\vec{w}) = \sum_{i=1}^{N} (y_i - \vec{w}^\top \vec{x}_i)^2 + \lambda \|\vec{w}\|_2^2$$
$$= \|\vec{y} - X\vec{w}\|_2^2 + \lambda \|\vec{w}\|_2^2$$

Optimal Parameters:

$$\vec{w}^* := \arg\min_{\vec{w}} L(\vec{w}) = (X^\top X + \lambda I)^{-1} X^\top \vec{y}$$

OLS vs. Ridge Regression (Probabilistic Interpretation)

Probabilistic Model: $y|\vec{x},\vec{w},\sigma \sim \mathcal{N}(\vec{w}^\top\vec{x},\sigma^2)$; Prior: $\vec{w}|\sigma \sim \mathcal{N}\left(\vec{0},\frac{\sigma^2}{\lambda}I\right)$

Parameter of Interest: $\vec{w}$
Nuisance Parameter: σ

OLS:

MLE estimate:

$$\hat{\vec{w}}_{\text{MLE}} = \arg\min_{\vec{w}}(y_i - \vec{w}^\top\vec{x})^2$$
$$= \arg\min_{\vec{w}}\|\vec{y} - X\vec{w}\|_2^2$$
$$= (X^\top X)^{-1}X^\top\vec{y}$$

Ridge Regression:

MAP estimate:

$$\hat{\vec{w}}_{\text{MAP}} = \arg\min_{\vec{w}}(y_i - \vec{w}^\top\vec{x})^2 + \lambda\|\vec{w}\|_2^2$$
$$= \arg\min_{\vec{w}}\|\vec{y} - X\vec{w}\|_2^2 + \lambda\|\vec{w}\|_2^2$$
$$= (X^\top X + \lambda I)^{-1}X^\top\vec{y}$$

Linear Regression

The next step is to decide whether to use:

- Single output linear regression
- Multiple output linear regression

How do you decide?

- If you would like to predict a scalar value, use single output linear regression
- If you would like to predict a vector, use multiple output linear regression

Linear Regression

The next step is to decide whether to use:

- Single output linear regression
- **Multiple output linear regression**

How do you decide?

- If you would like to predict a scalar value, use single output linear regression
- If you would like to predict a vector, use multiple output linear regression

Multiple Output Linear Regression

Model: $\hat{\vec{v}} = W\vec{x}$, where $W \coloneqq \begin{pmatrix} \vec{w}_1^\top \\ \vdots \\ \vec{w}_m^\top \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1n} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ w_{m1} & \cdots & w_{mn} & b_m \end{pmatrix}$ and $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix}$

Parameters: W

Loss:

$$L(W) = \sum_{i=1}^{N} \|\vec{v}_i - W\vec{x}_i\|_2^2 = \|Y - XW^\top\|_F^2, \text{where } X = \begin{pmatrix} \vec{x}_1^\top \\ \vdots \\ \vec{x}_N^\top \end{pmatrix}, Y = \begin{pmatrix} \vec{v}_1^\top \\ \vdots \\ \vec{v}_N^\top \end{pmatrix} \coloneqq \begin{pmatrix} \vec{y}_1 & \cdots & \vec{y}_m \end{pmatrix}$$

Training: Calculate optimal parameters using closed-form formula: $W^{*\top} = (X^\top X)^{-1} X^\top Y$

Testing: $\hat{\vec{v}} = W^*\vec{x}$

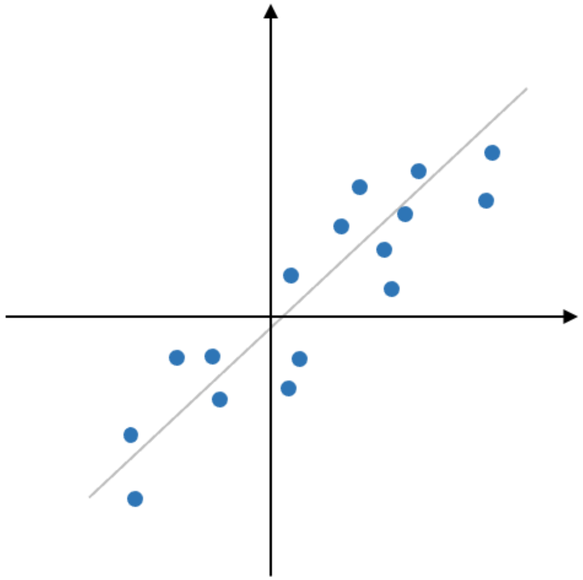
Regression

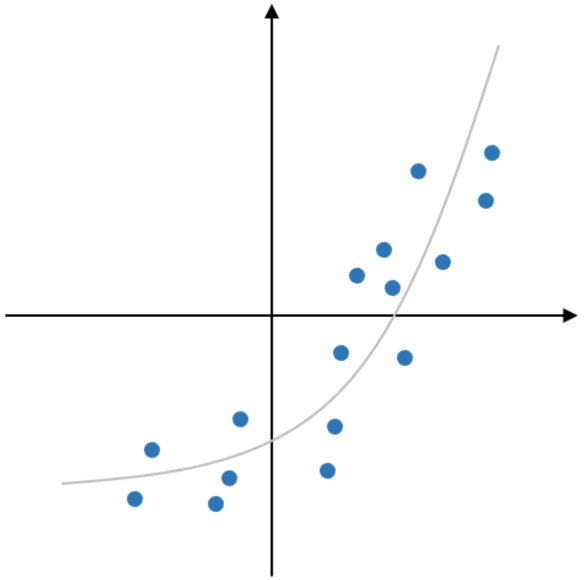
The next step is to decide which model to use. There are two kinds of models:

- Linear regression models
- Non-linear regression models

How do you decide?

- If you expect the label to be linear in the input, use a linear regression model
  - E.g.: if an increase in an input dimension is expected to result in a proportional increase in the label, regardless of the starting value
- If you expect the label to be non-linear in the input, use a non-linear regression model
  - E.g.: if the label can be similar for both extremes of an input dimension
- If you don't know, try both and see which achieves better validation loss. This process is known as **model selection**.





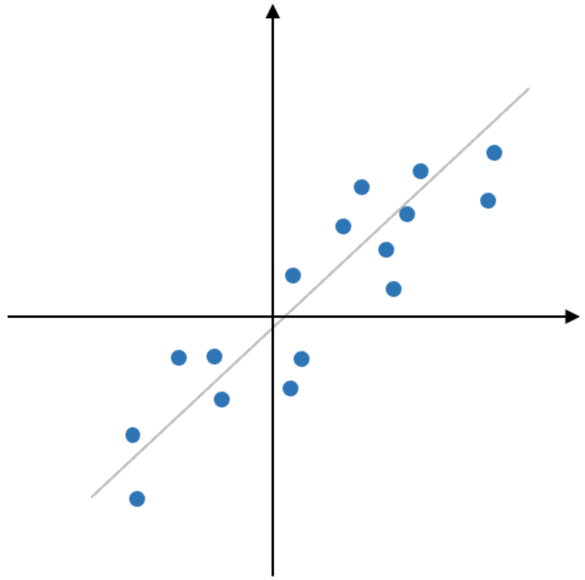
Regression

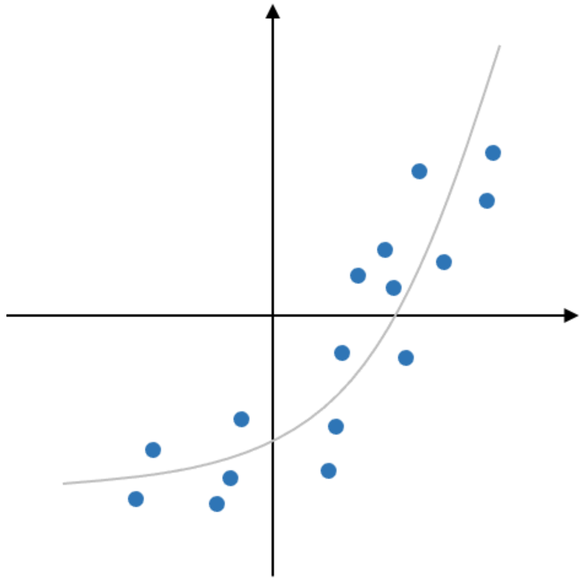
The next step is to decide which model to use. There are two kinds of models:

- Linear regression models
- **Non-linear regression models**

How do you decide?

- If you expect the label to be linear in the input, use a linear regression model
  - E.g.: if an increase in an input dimension is expected to result in a proportional increase in the label, regardless of the starting value
- If you expect the label to be non-linear in the input, use a non-linear regression model
  - E.g.: if the label can be similar for both extremes of an input dimension
- If you don't know, try both and see which achieves better validation loss. This process is known as **model selection**.





Non-Linear Regression

Two approaches:

- Linear regression on features: replace the raw data points $\vec{x}_i$ with features $\phi(\vec{x}_i)$ in linear regression.
  - Pro: Can use the same optimization method as in linear regression, i.e.: computing optimal parameters with closed-form formula.
  - Con: Features are fixed, so model is less expressive.
  - Con: Features can be high-dimensional and are computationally expensive to compute.
- Non-linear least squares: use a model that is non-linear in the parameters.
  - Pro: Can learn features (in the case of neural networks), so model can be more expressive.
  - Pro: Don’t have to use high-dimensional features for the model to be expressive.
  - Con: Non-convex loss, so may not be able to find global minimum.

Non-Linear Regression

Two approaches:

- **Linear regression on features**: replace the raw data points $\vec{x}_i$ with features $\phi(\vec{x}_i)$ in linear regression.
  - Pro: Can use the same training procedure as in linear regression, i.e.: computing optimal parameters with closed-form formula.
  - Con: Features are fixed, so model is less expressive.
  - Con: Features can be high-dimensional and are computationally expensive to compute.
- Non-linear least squares: use a model that is non-linear in the parameters.
  - Pro: Can learn features (in the case of neural networks), so model can be more expressive.
  - Pro: Don’t have to use high-dimensional features for the model to be expressive.
  - Con: Non-convex loss, so may not be able to find global minimum.

Linear Regression on Features

A feature map $\phi: \mathbb{R}^n \to \mathbb{R}^{n'}$) where $n' \geq n$ typically) converts raw data into features, which are used in place of the raw data as input to a model.

Model: $\hat{y} = \vec{w}^\top \phi(\vec{x})$

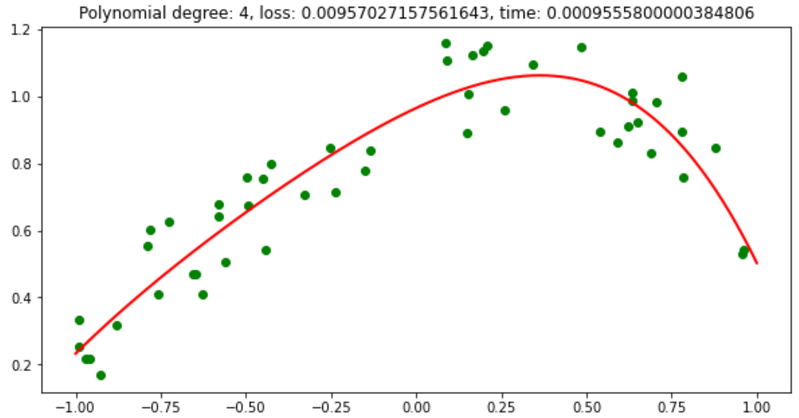
Parameters:$\vec{w}$

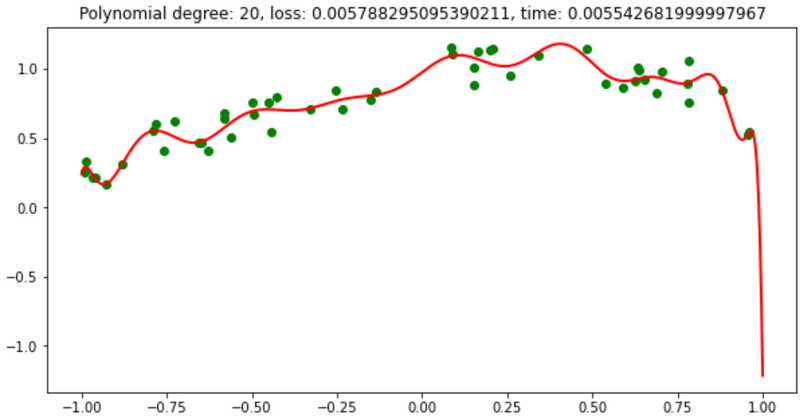
$$L(\vec{w}) = \sum_{i=1}^{N} (y_i - \vec{w}^\top \phi(\vec{x}_i))^2 + \lambda \|\vec{w}\|_2^2$$
$$= \|\vec{y} - \Phi\vec{w}\|_2^2 + \lambda \|\vec{w}\|_2^2$$

Training: Calculate optimal parameters using closed-form formula:

$$\vec{w}^* := \arg\min_{\vec{w}} L(\vec{w}) = (\Phi^\top \Phi)^{-1} \Phi^\top \vec{y}$$

Testing: $\hat{y} = \vec{w}^{\top *} \phi(\vec{x})$





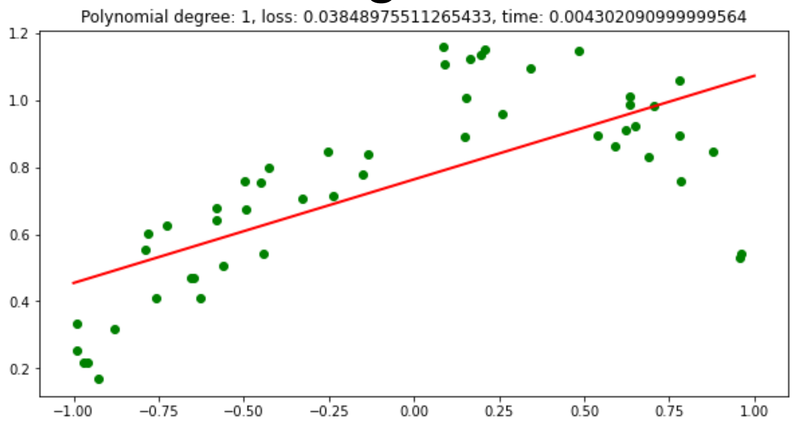
Linear Regression on Polynomial Features

Polynomial features map raw data to monomials of varying degrees.

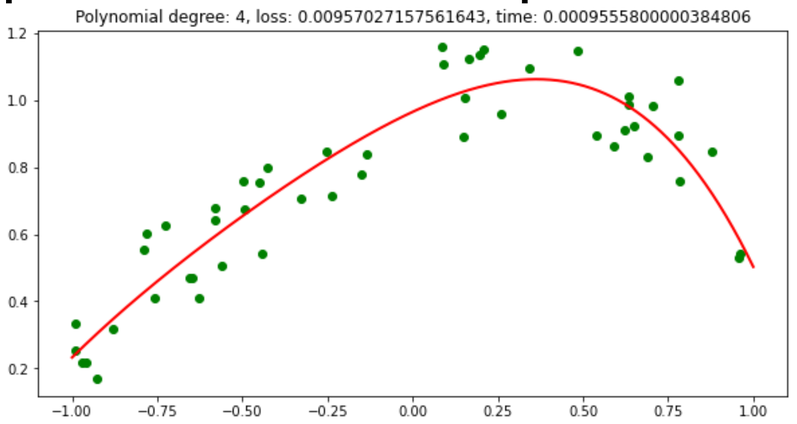
E.g.: Polynomial features up to degree 2

$$\phi(\vec{x}) = (\underbrace{1}_{\text{degree 0}} \quad \underbrace{x_1 \quad \cdots \quad x_n}_{\text{degree 1}} \quad \underbrace{x_1^2 \quad x_1x_2 \quad \cdots \quad x_1x_n \quad x_2^2 \quad x_2x_3 \quad \cdots \quad x_n^2}_{\text{degree 2 (don't forget the cross terms!)}})^\top$$

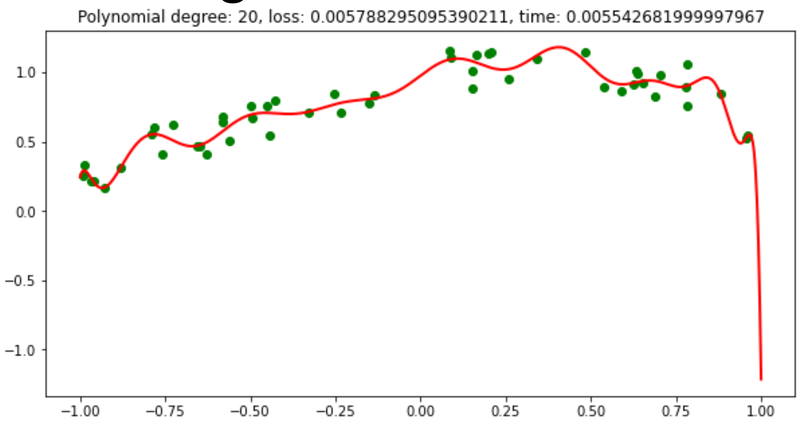
Higher degree => more expressive and more prone to overfitting
Lower degree => less expressive and less prone to overfitting



Degree 1



Degree 4



Degree 20

Non-Linear Regression

Two approaches:

- Linear regression on features: replace the raw data points $\vec{x}_i$ with features $\phi(\vec{x}_i)$ in linear regression.
  - Pro: Can use the same optimization method as in linear regression, i.e.: computing optimal parameters with closed-form formula.
  - Con: Features are fixed, so model is less expressive.
  - Con: Features can be high-dimensional and are computationally expensive to compute.
- **Non-linear least squares**: use a model that is non-linear in the parameters.
  - Pro: Can learn features (in the case of neural networks), so model can be more expressive.
  - Pro: Don’t have to use high-dimensional features for the model to be expressive.
  - Con: Non-convex loss, so may not be able to find global minimum.

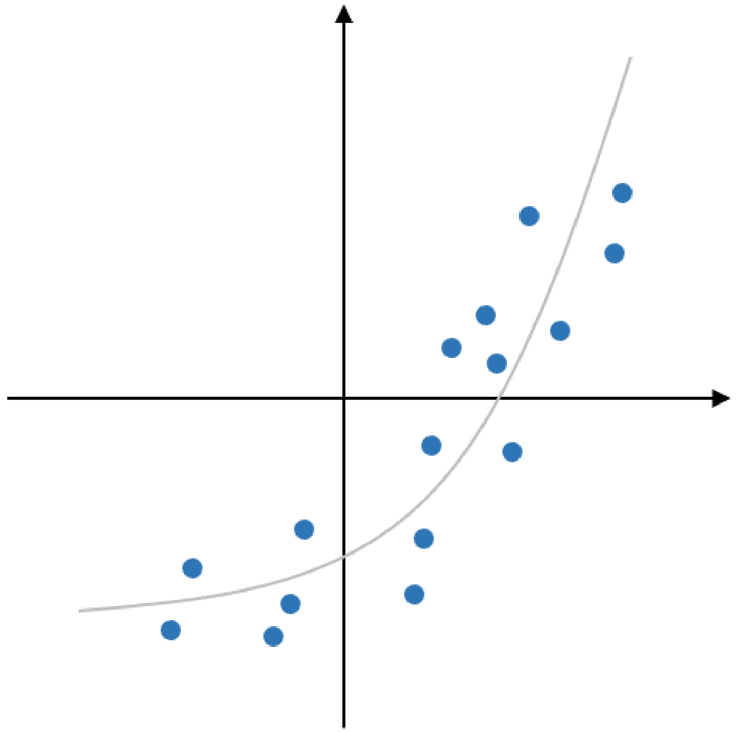
Non-Linear Least Squares

Model: $\hat{y} = f(\vec{x}; \vec{w})$

Parameters: $\vec{w}$

Loss:

$$L(\vec{w}) = \sum_{i=1}^{N} \left(y_i - f(\vec{x}_i; \vec{w})\right)^2 + \lambda \|\vec{w}\|_2^2$$



Training: Iterative gradient-based optimization algorithm that minimizes $L(\cdot)$ w.r.t. $\vec{w}$

Testing: $\hat{y} = f(\vec{x}_i; \vec{w}^*)$, where $\vec{w}^*$ is the solution found by the optimization algorithm

Neural Networks

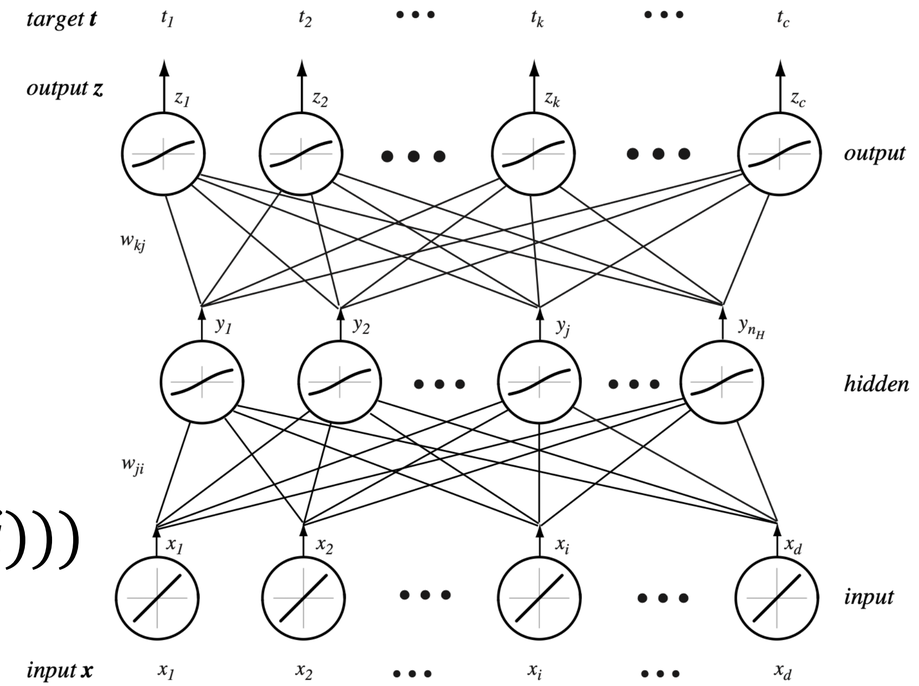
A particular choice of the model f is a neural network.

Key Idea: Learn the features based on the data rather than defining them arbitrarily.

Model:

$$\hat{\vec{y}} := f(\vec{x}; \{W_l\}_{l=0}^{L}) = W_L \psi(W_{L-1} \psi(W_{L-2} \cdots \psi(W_0 \vec{x})))$$

Parameters: W_L ,W_{L-1} ,…. ,and W_0



Training Neural Networks

No closed form solution for the optimal parameters.

Must use iterative gradient-based optimization methods.

- Need to compute gradients efficiently, which is achieved using backpropagation.
  - Forward pass: compute activations from the input layer to the output layer
  - Backward pass: compute gradient w.r.t. activations from the output layer to the input layer

Loss function is non-convex, so may not be able to find the global minimum.

- Gradient descent may get stuck at a local minimum.
  - Momentum can sometimes avoid this, but is not guaranteed to do so.
- Gradient may vanish and explode.
  - To avoid vanishing gradients: Non-saturating activation functions (e.g.: ReLU and softplus), Xavier/He initialization, layer normalization, reparameterization, residual connections
  - To avoid exploding gradients: Xavier/He initialization, layer normalization, weight decay,
  - To get around exploding gradients: gradient clipping

Expressiveness of Neural Networks

Universal function approximator: with sufficiently many hidden units, can approximate any continuous function to arbitrary precision

- More hidden units => more expressive
- More hidden layers => more expressive (since a shallower neural network is a special case of a deeper neural network)

To alleviate overfitting:

- Reduce expressiveness
- Weight decay
- Early stopping

Regression vs. Classification

When you are given a machine learning problem, what should you look for?

- The first thing is to decide which kind of machine learning problem it is.

There are two major types of problems:

- Regression problems
- Classification problems

How do you tell?

- If the goal is to predict a **value within a continuous interval**, the problem is a regression problem.
- If the goal is to predict **one of a discrete set of labels**, the problem is a classification problem.

Examples:

- Predicting tomorrow's temperature: regression
- Predicting whether it rains tomorrow: classification

Regression vs. Classification

When you are given a machine learning problem, what should you look for?

- The first thing is to decide which kind of machine learning problem it is.

There are two major types of problems:

- Regression problems
- **Classification problems**

How do you tell?

- If the goal is to predict a **value within a continuous interval**, the problem is a regression problem.
- If the goal is to predict **one of a discrete set of labels**, the problem is a classification problem.

Examples:

- Predicting tomorrow's temperature: regression
- Predicting whether it rains tomorrow: classification

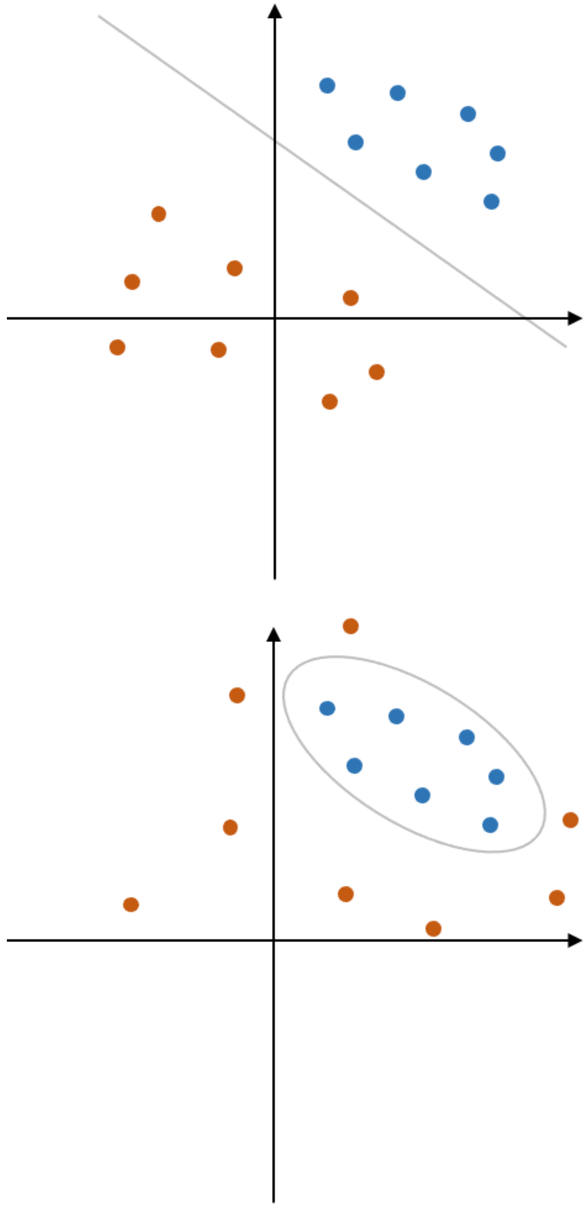
Classification

The next step is to decide which model to use. There are two kinds of models:

- Linear classifiers
- Non-linear classifiers

How do you decide?

- If the data points from different classes **can be mostly separated by hyperplanes**, use a linear classifier
  - Typically the case when the number of data points is not much larger than the number of dimensions
- Otherwise, use a non-linear classifier
- If you don’t know, try both and see which achieves better validation loss. This process is known as **model selection**.



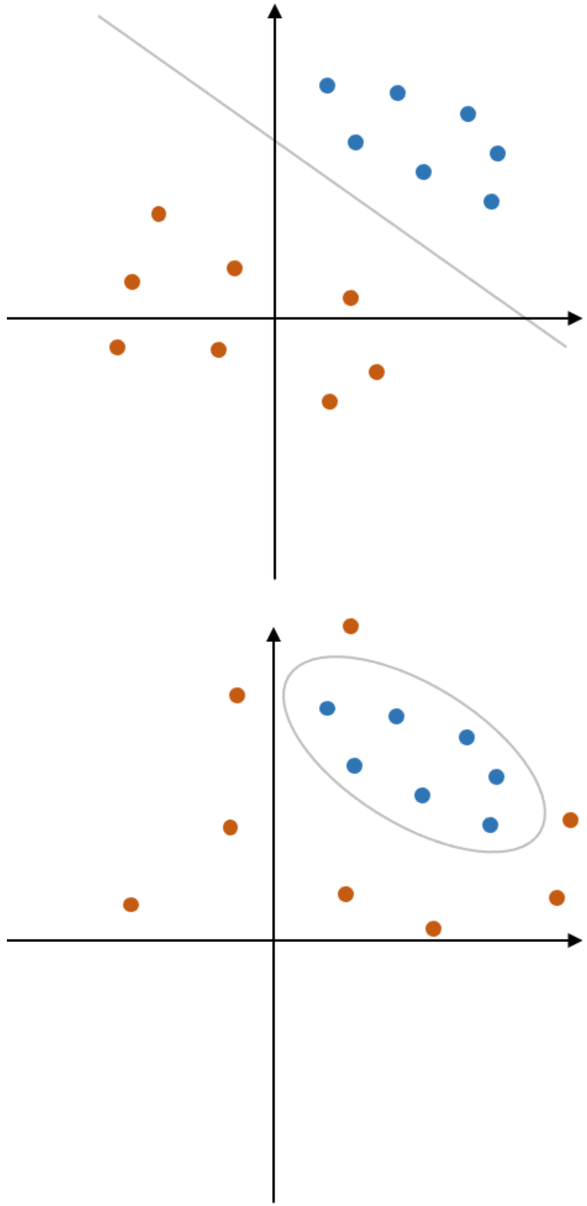
Classification

The next step is to decide which model to use. There are two kinds of models:

- **Linear classifiers**
- Non-linear classifiers

How do you decide?

- If the data points from different classes **can be mostly separated by hyperplanes**, use a linear classifier
  - Typically the case when the number of data points is not much larger than the number of dimensions
- Otherwise, use a non-linear classifier
- If you don’t know, try both and see which achieves better validation loss. This process is known as **model selection**.



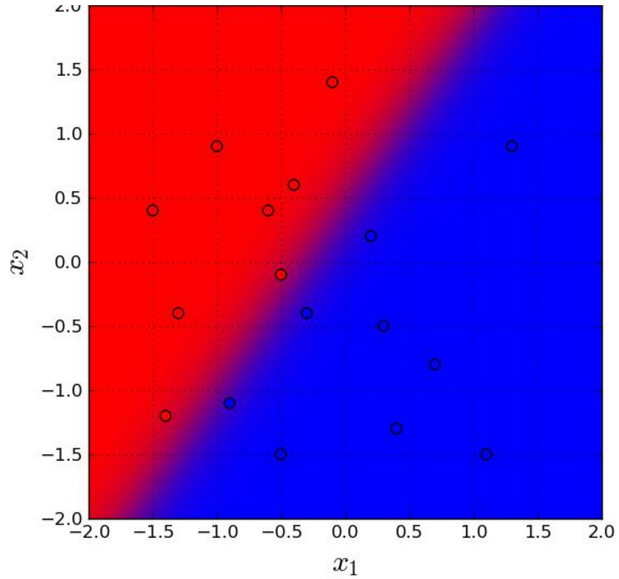
Linear Classifiers

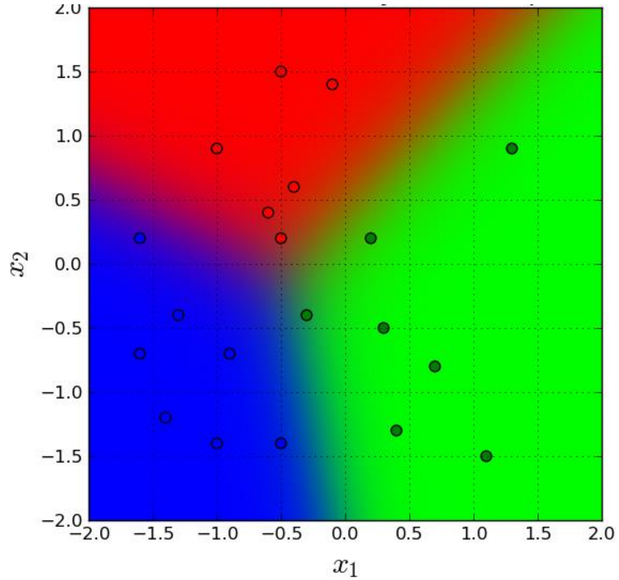
The next step is to decide whether to use:

- Binary linear classifier
- Multinomial linear classifier

How do you decide?

- If you have two classes, use a binary classifier
- If you have more than two classes, use a multinomial classier





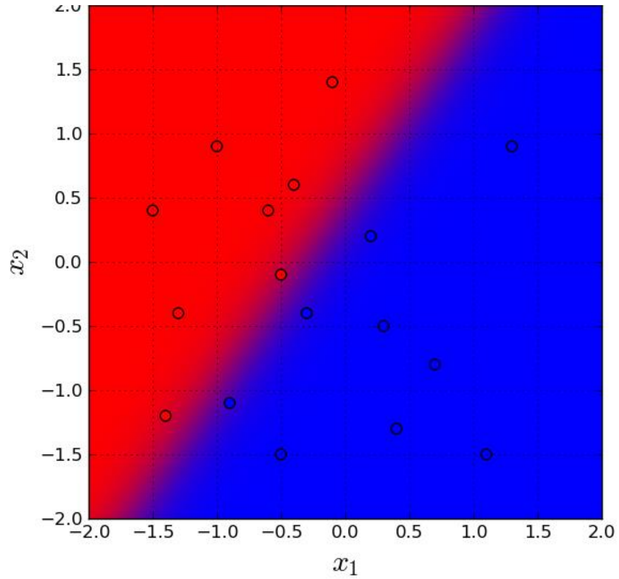
Linear Classifiers

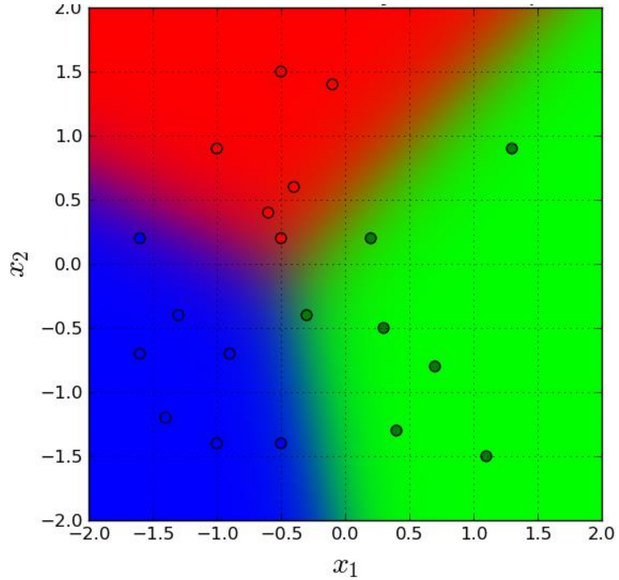
The next step is to decide whether to use:

- **Binary linear classifier**
- Multinomial linear classifier

How do you decide?

- If you have two classes, use a binary classifier
- If you have more than two classes, use a multinomial classier

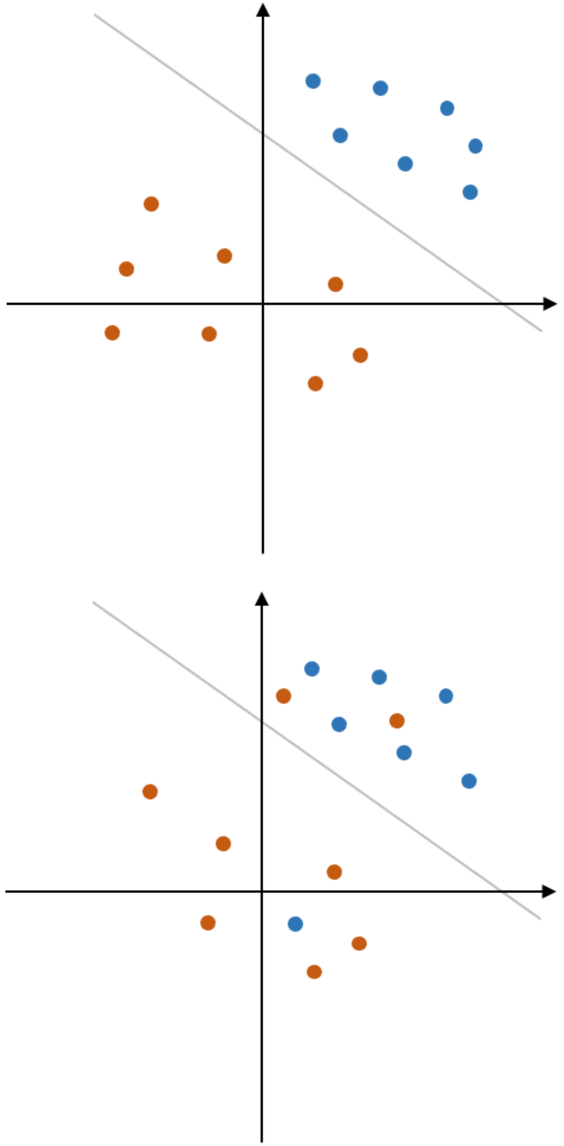




Binary Linear Classifiers

The next step is to see if the two classes can be perfectly separated:

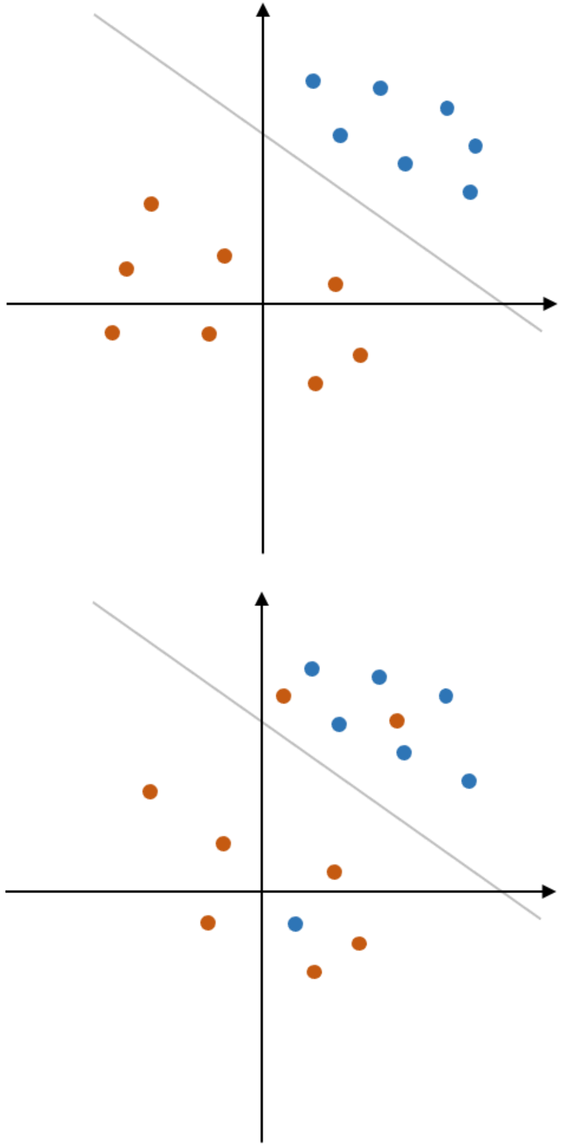
- If so, you can use a hard-margin SVM.
- If not, you cannot use a hard-margin SVM. Instead, you can use:
  - Soft-margin SVMs
  - Logistic Regression
- If you don’t know, try training a hard-margin SVM on the dataset and see if it’s infeasible.



Binary Linear Classifiers

The next step is to see if the two classes can be perfectly separated:

- **If so, you can use a hard-margin SVM.**
- If not, you cannot use a hard-margin SVM. Instead, you can use:
  - Soft-margin SVMs
  - Logistic Regression
- If you don’t know, try training a hard-margin SVM on the dataset and see if it’s infeasible.

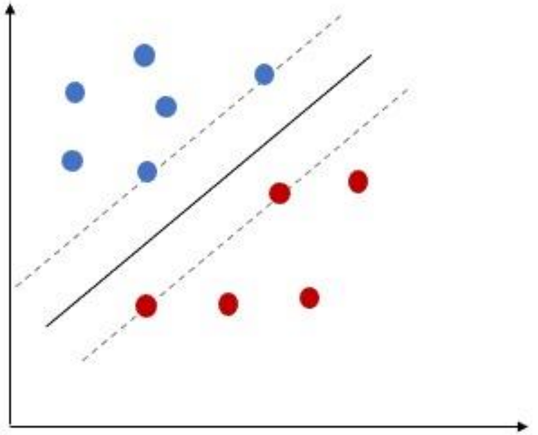


Hard-Margin SVM

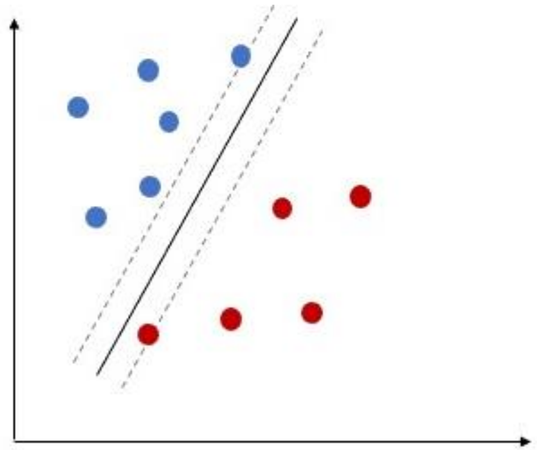
The hard-margin SVM aims to find a hyperplane that perfectly separates the positive data points from the negative data points.

Among all such hyperplanes, it picks the one that maximizes the **margin**, which is the buffer zone on either side of the hyperplane that is empty.

- Such a hyperplane is good because it provides maximum robustness to perturbations of the data points.
- Intuitively, this leads to better generalization.



Preferred



Less Preferred

Hard-Margin SVM

Because of strong duality, we can either solve the primal form or the dual form.

Primal form:

$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|_2^2$$
$$\text{s.t. } y_i(\vec{w}^\top \vec{x}_i - b) \geq 1 \; \forall i$$

Dual form:

$$\max_{\vec{\lambda}} \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \vec{x}_j^\top \vec{x}_i$$
$$\text{s.t. } \sum_{i=1}^{N} \lambda_i y_i = 0 \text{ and } \lambda_i \geq 0 \; \forall i$$

Once the optimal solution $\vec{\lambda}^*$ is found:

$$\vec{w}^* = \sum_{i=1}^{N} \lambda_i^* y_i \vec{x}_i$$
$$b^* = \frac{\max_{i:y_i=-1}\{\vec{w}^{*\top}\vec{x}_i\} + \min_{i:y_i=1}\{\vec{w}^{*\top}\vec{x}_i\}}{2}$$

Testing:

$$\hat{y} = \begin{cases} 1 & \vec{w}^{*\top}\vec{x}_i - b > 0 \\ -1 & \vec{w}^{*\top}\vec{x}_i - b < 0 \end{cases}$$

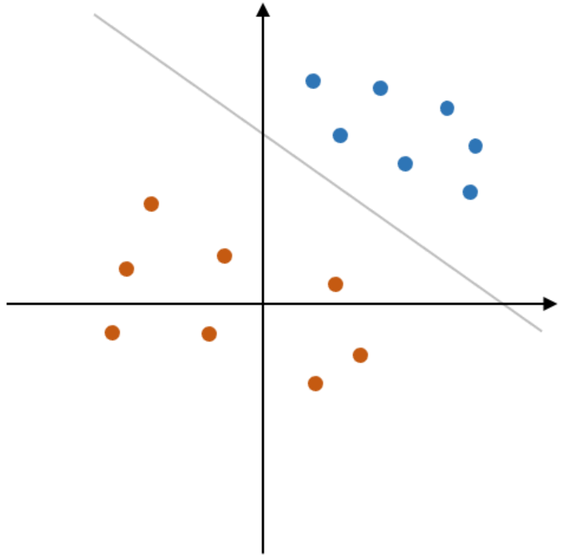
How to decide which one to use?

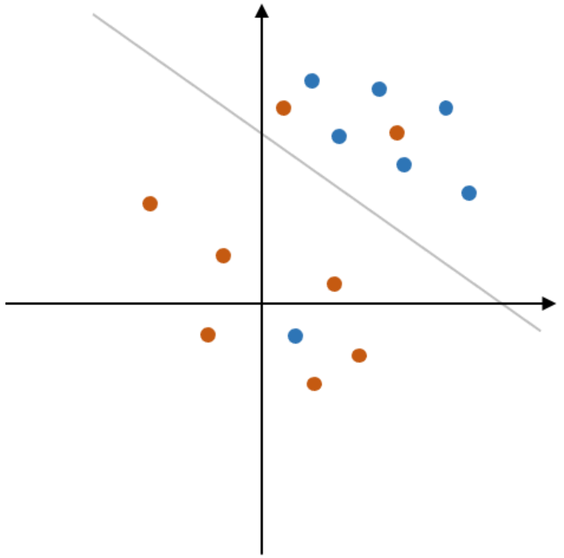
- When $n \ll N$ (dimensionality much less than the number of data points), more efficient to solve the primal.
- Otherwise, more efficient to solve the dual.

Binary Linear Classifiers

The next step is to see if the two classes can be perfectly separated:

- If so, you can use a hard-margin SVM.
- If not, you cannot use a hard-margin SVM. Instead, you can use:
  - Soft-margin SVMs
  - Logistic Regression
- If you don’t know, try training a hard-margin SVM on the dataset and see if it’s infeasible.

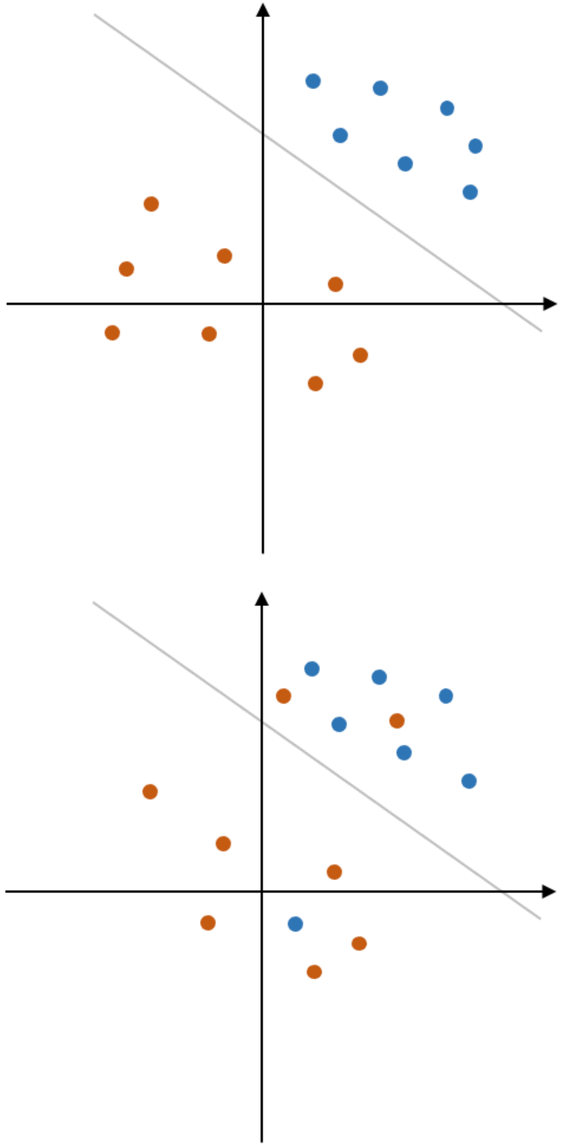




Binary Linear Classifiers

The next step is to see if the two classes can be perfectly separated:

- If so, you can use a hard-margin SVM.
- If not, you cannot use a hard-margin SVM**. Instead, you can use:**
  - **Soft-margin SVMs**
  - **Logistic Regression**
- If you don’t know, try training a hard-margin SVM on the dataset and see if it’s infeasible.



Soft-Margin SVM vs. Logistic Regression

The difference between soft-margin SVM and logistic regression amounts to choosing a different loss:

$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\left(y_i, f(\vec{x}_i; \vec{w})\right) + \lambda \|\vec{w}\|_2^2 \text{, where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Soft-margin SVM (hinge loss):

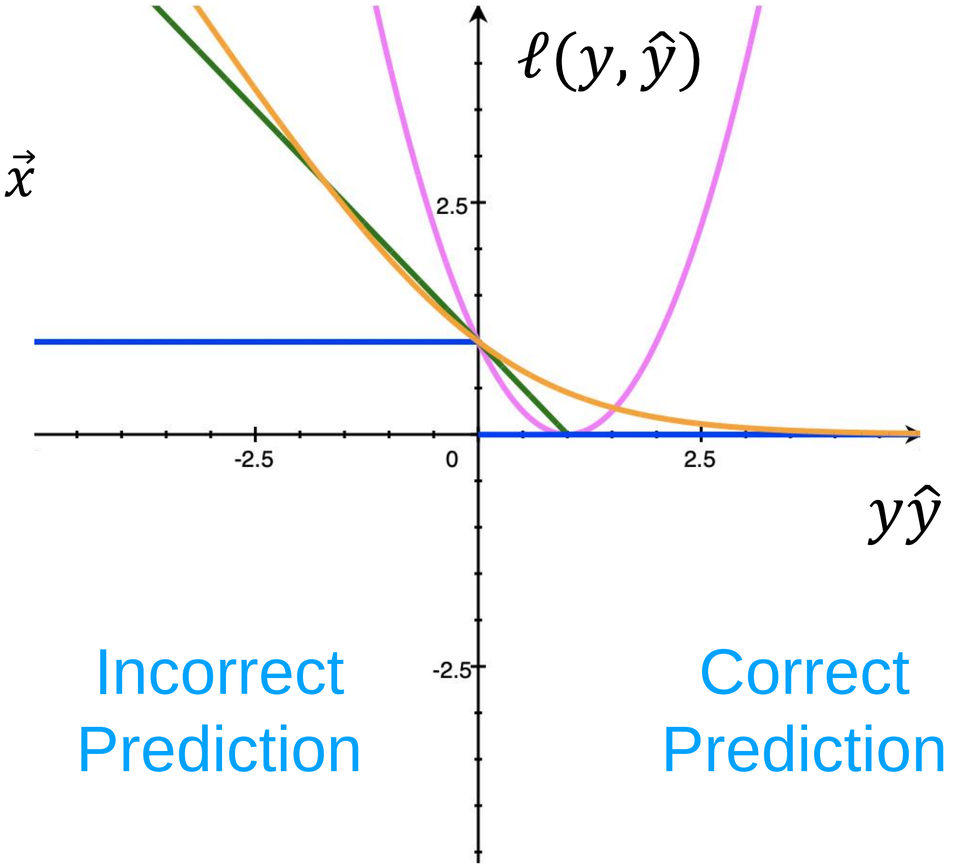
$$\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$$

Logistic loss (logistic loss):

$$\ell(y, \hat{y}) = \log\left(1 + e^{-y\hat{y}}\right) / \log 2$$

Both are **surrogate losses** to the 0-1 loss:

$$\ell(y, \hat{y}) = \begin{cases} 1 & y \neq \operatorname{sign} \hat{y} \\ 0 & y = \operatorname{sign} \hat{y} \end{cases} = \begin{cases} 1 & y\hat{y} < 0 \\ 0 & y\hat{y} > 0 \end{cases}$$



Soft-Margin SVM vs. Logistic Regression

The difference between soft-margin SVM and logistic regression amounts to choosing a different loss:

$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\left(y_i, f(\vec{x}_i; \vec{w})\right) + \lambda \|\vec{w}\|_2^2 \text{, where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Soft-margin SVM (hinge loss):

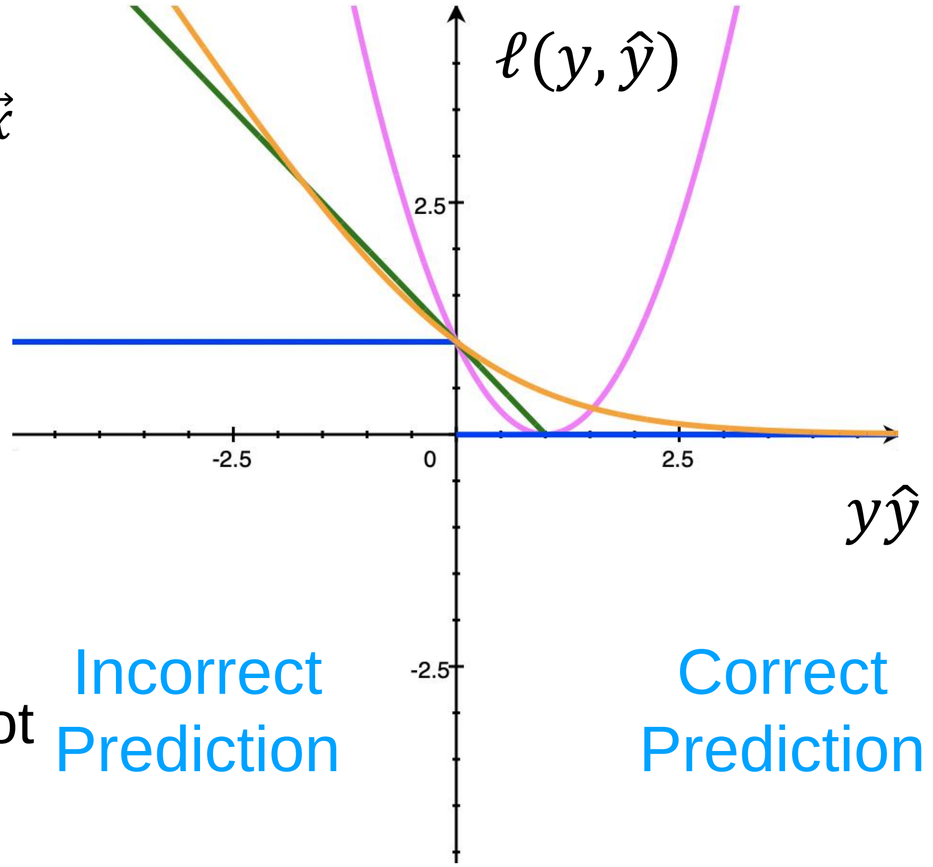
$$\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$$

Logistic loss (logistic loss):

$$\ell(y, \hat{y}) = \log\left(1 + e^{-y\hat{y}}\right)/\log 2$$

Not advisable to use ridge regression because it is not robust to outliers and intra-class variability:

Ridge regression (square loss): $\ell(y, \hat{y}) = (1 - y\hat{y})^2$



Soft-Margin SVM vs. Logistic Regression

The difference between soft-margin SVM and logistic regression amounts to choosing a different loss:

$$\min_{\vec{w}} \sum_{i=1}^{N} \ell\big(y_i, f(\vec{x}_i; \vec{w})\big) + \lambda \|\vec{w}\|_2^2 \text{, where } f(\vec{x}; \vec{w}) = \vec{w}^\top \vec{x}$$

Soft-margin SVM (hinge loss):

$$\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$$

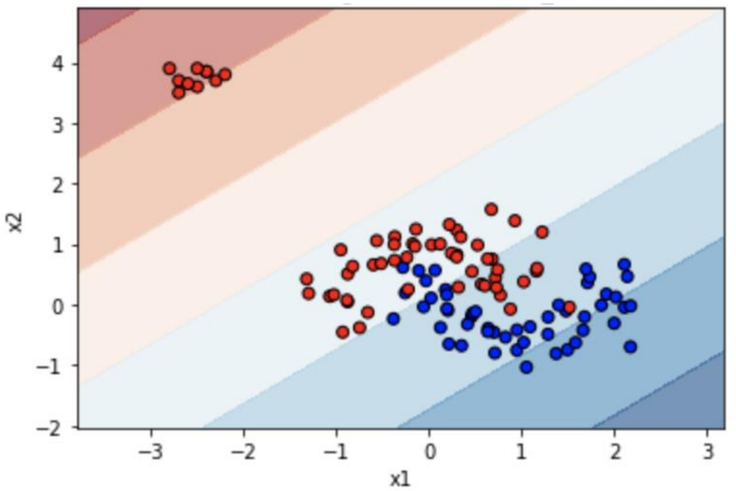
Logistic loss (logistic loss):

$$\ell(y, \hat{y}) = \log\big(1 + e^{-y\hat{y}}\big)/\log 2$$

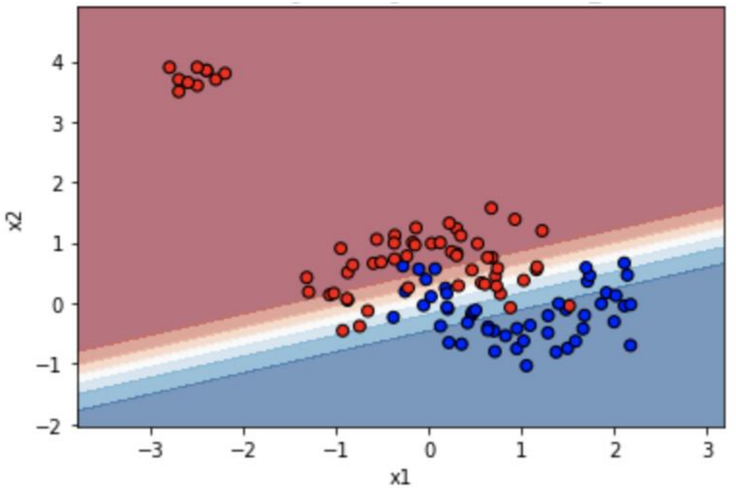
Not advisable to use ridge regression because it is not robust to outliers and intra-class variability:

Ridge regression (square loss): $\ell(y, \hat{y}) = (1 - y\hat{y})^2$

Square loss



Logistic loss



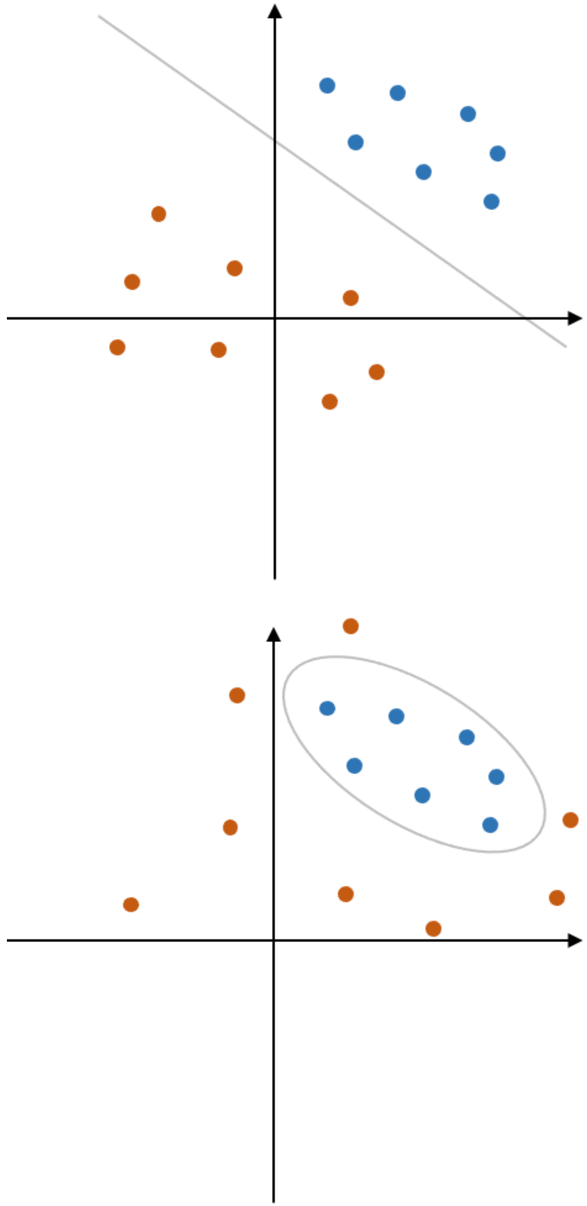
Classification

The next step is to decide which model to use. There are two kinds of models:

- Linear classifiers
- Non-linear classifiers

How do you decide?

- If the data points from different classes **can be mostly separated by hyperplanes**, use a linear classifier
  - Typically the case when the number of data points is not much larger than the number of dimensions
- Otherwise, use a non-linear classifier
- If you don’t know, try both and see which achieves better validation loss. This process is known as **model selection**.



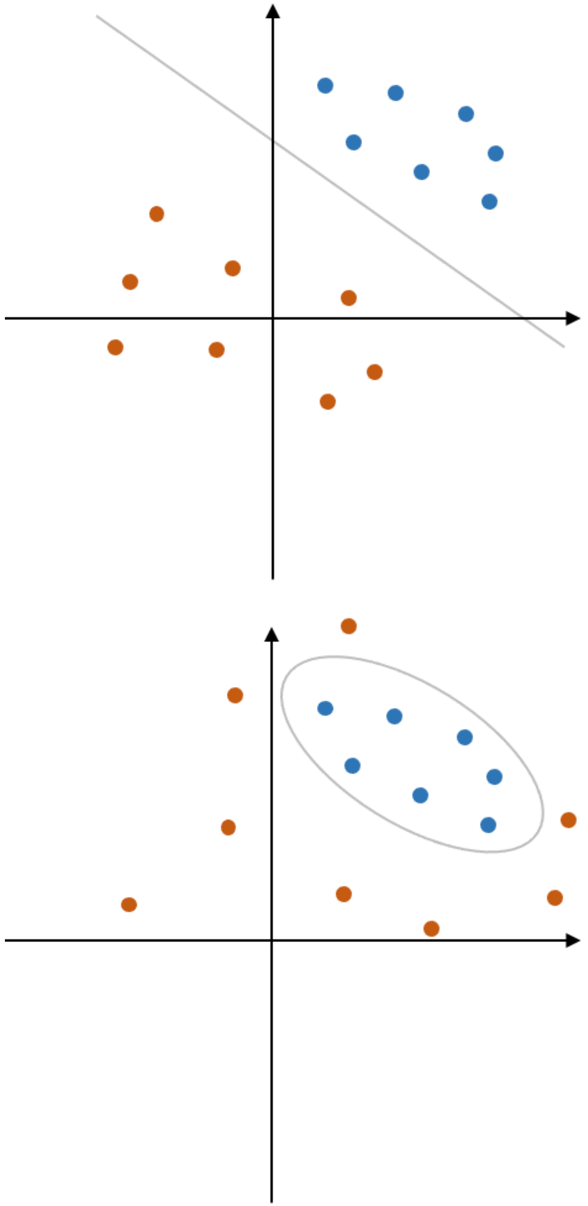
Classification

The next step is to decide which model to use. There are two kinds of models:

- Linear classifiers
- **Non-linear classifiers**

How do you decide?

- If the data points from different classes **can be mostly separated by hyperplanes**, use a linear classifier
  - Typically the case when the number of data points is not much larger than the number of dimensions
- Otherwise, use a non-linear classifier
- If you don’t know, try both and see which achieves better validation loss. This process is known as **model selection**.



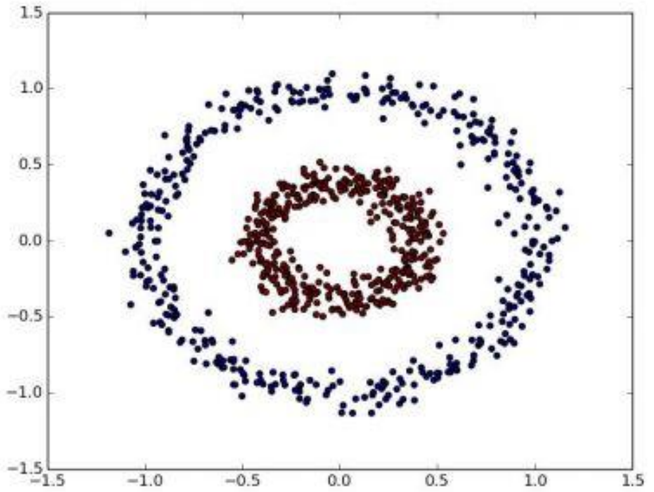
Non-Linear Classifiers

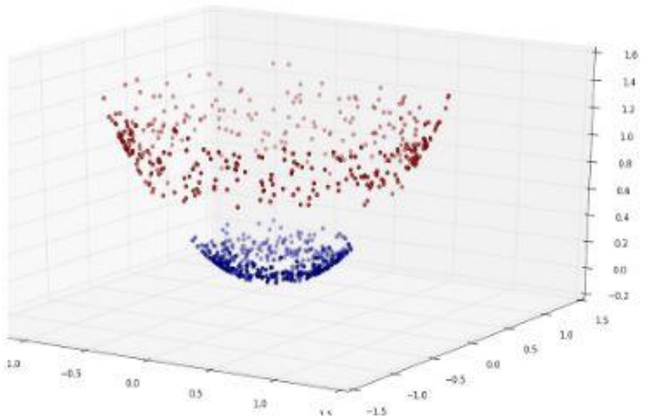
Key idea: Take a linear classifier, but replace the raw data points $\vec{x}_i$ with features $\phi(\vec{x}_i)$.

Two approaches:

- Kernelization: High-dimensional fixed features; expensive computation avoided using kernel trick.
  - Pro: Can use the same training procedure as in linear classifiers. Loss often convex and Lipschitz, so can find global minimum reliably.
  - Con: Features are fixed, so model is less expressive.

- Neural networks: Low-dimensional learned features.
  - Pro: Features are learned, so model is more expressive.
  - Con: Non-convex loss, so may not be able to find global minimum.





Credit: Suriya Narayanan

Choice of Base Linear Classifier

Which linear classifier to use as a basis for the non-linear classifier?

Hard-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
- Con: Constrained problem, so cannot use iterative gradient-based optimization.

Soft-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
- Con: Iterative gradient-based optimization does not work as well because the derivative is discontinuous.

Logistic regression:

- Pro: Can use iterative gradient-based optimization.
- Con: Cannot apply the kernel trick.

Choice of Base Linear Classifier

Which linear classifier to use as a basis for the non-linear classifier?

Hard-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Constrained problem, so cannot use iterative gradient-based optimization.

Soft-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
- Con: Iterative gradient-based optimization does not work as well because the derivative is discontinuous.

Logistic regression:

- Pro: Can use iterative gradient-based optimization.
- Con: Cannot apply the kernel trick.

Choice of Base Linear Classifier

Which linear classifier to use as a basis for the non-linear classifier?

Hard-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Constrained problem, so cannot use iterative gradient-based optimization.
  Does not work with neural networks

Soft-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
- Con: Iterative gradient-based optimization does not work as well because the derivative is discontinuous.

Logistic regression:

- Pro: Can use iterative gradient-based optimization.
- Con: Cannot apply the kernel trick.

Choice of Base Linear Classifier

Which linear classifier to use as a basis for the non-linear classifier?

Hard-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Constrained problem, so cannot use iterative gradient-based optimization.
  Does not work with neural networks

Soft-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Iterative gradient-based optimization does not work as well because the derivative is discontinuous.

Logistic regression:

- Pro: Can use iterative gradient-based optimization.
- Con: Cannot apply the kernel trick.

Choice of Base Linear Classifier

Which linear classifier to use as a basis for the non-linear classifier?

Hard-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Constrained problem, so cannot use iterative gradient-based optimization.
  Does not work with neural networks

Soft-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Iterative gradient-based optimization does not work as well because the derivative is discontinuous.
  Works less well with neural networks

Logistic regression:

- Pro: Can use iterative gradient-based optimization.
- Con: Cannot apply the kernel trick.

Choice of Base Linear Classifier

Which linear classifier to use as a basis for the non-linear classifier?

Hard-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Constrained problem, so cannot use iterative gradient-based optimization.
  Does not work with neural networks

Soft-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Iterative gradient-based optimization does not work as well because the derivative is discontinuous.
  Works less well with neural networks

Logistic regression:

- Pro: Can use iterative gradient-based optimization.
  Works with neural networks
- Con: Cannot apply the kernel trick.

Choice of Base Linear Classifier

Which linear classifier to use as a basis for the non-linear classifier?

Hard-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Constrained problem, so cannot use iterative gradient-based optimization.
  Does not work with neural networks

Soft-margin SVM:

- Pro: Can derive the dual, which only involves inner products between data points. Hence can apply the kernel trick.
  Works with kernelization
- Con: Iterative gradient-based optimization does not work as well because the derivative is discontinuous.
  Works less well with neural networks

Logistic regression:

- Pro: Can use iterative gradient-based optimization.
  Works with neural networks
- Con: Cannot apply the kernel trick.
  Does not work with kernelization

Hard-Margin SVM with Kernels

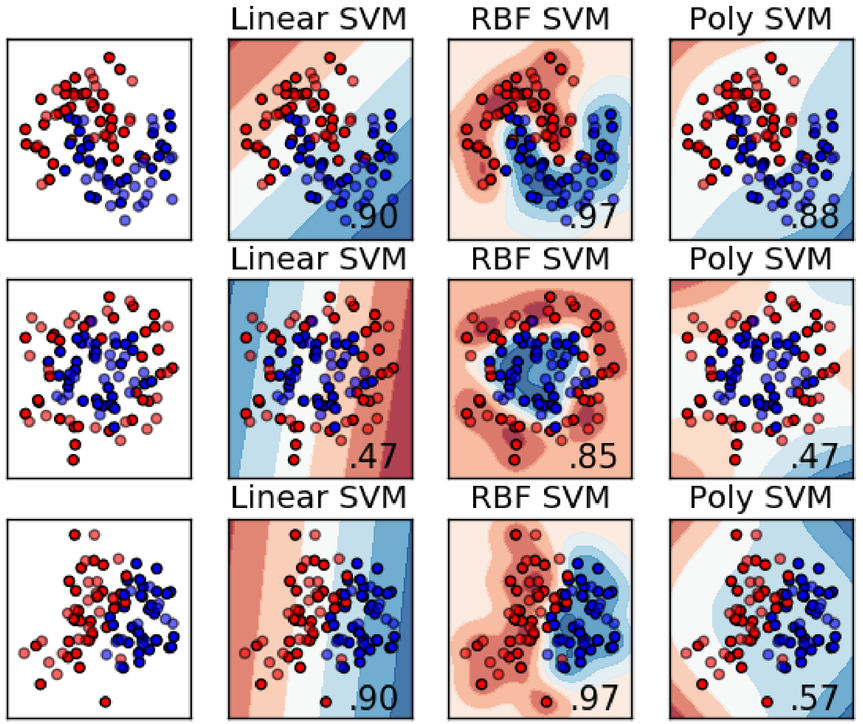
$$\max_{\vec{\lambda}} \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j \underbrace{\phi(\vec{x}_j)^\top \phi(\vec{x}_i)}_{k(\vec{x}_j, \vec{x}_i)} \text{ s.t. } \sum_{i=1}^{N} \lambda_i y_i = 0 \text{ and } \lambda_i \geq 0 \; \forall i$$

Linear kernel: $k(\vec{x}, \vec{y}) = \vec{x}^\top \vec{y} \Leftrightarrow \phi(\vec{x}) = \vec{x}$

Quadratic kernel: $k(\vec{x}, \vec{y}) = (1 + \vec{x}^\top \vec{y})^2$

Polynomial kernel: $k(\vec{x}, \vec{y}) = (1 + \vec{x}^\top \vec{y})^d$

Radial basis function (RBF) kernel: $k(\vec{x}, \vec{y}) = \exp\left(-\frac{1}{2} \|\vec{x} - \vec{y}\|_2^2\right)$



Credit: Fernando Wittmann

Logistic Regression with Neural Networks

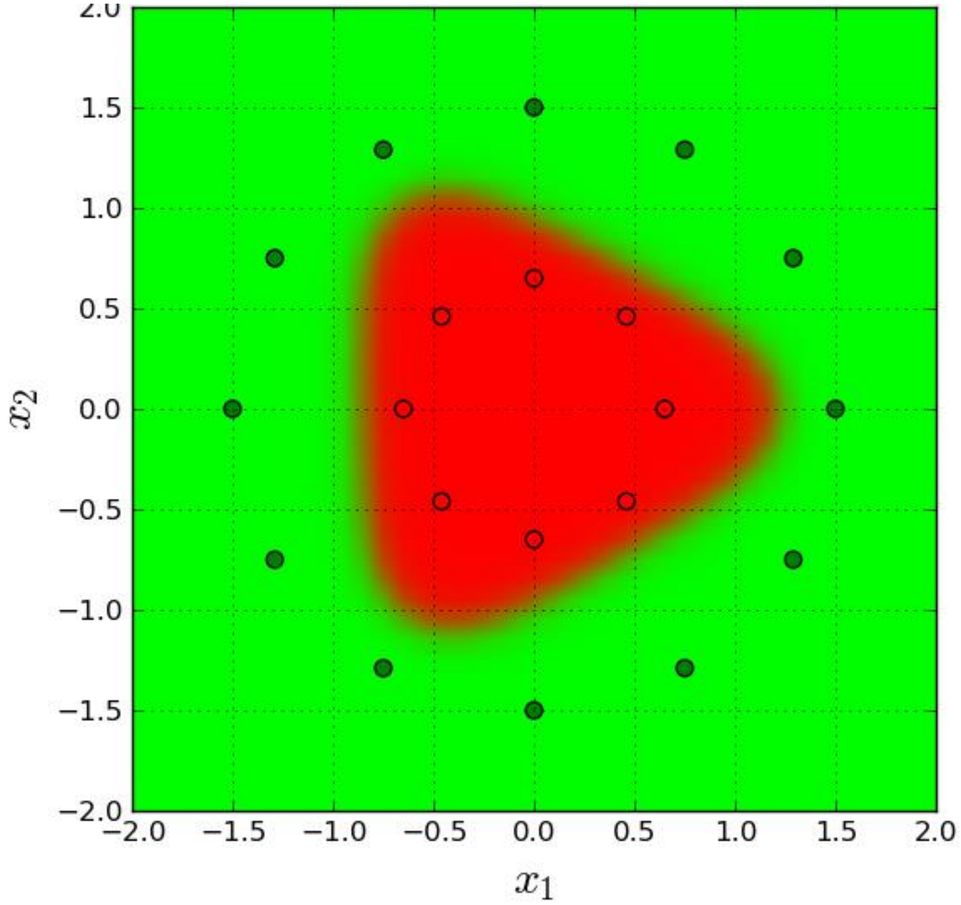
Model: a neural network whose output layer has a sigmoid activation function

Parameters: weight matrices in all layers

Loss: cross entropy loss

Training: Gradient descent

Testing: $\hat{y} = \begin{cases} 1 & \sigma(\vec{w}^{*\top}\vec{x}) > \frac{1}{2} \\ -1 & \sigma(\vec{w}^{*\top}\vec{x}) < \frac{1}{2} \end{cases}$



Credit: Sait Celebi

Review:
Mathematical Foundations

Linear Algebra

Singular value decomposition: $A = U\Sigma V^\top$, where Σ is diagonal, and U and V are orthogonal.

Eigendecomposition: if A is symmetric, $A = U\Lambda U^\top$, where Λ is diagonal and U is orthogonal.

The right-singular vectors are eigenvectors of $A^\top A$.

The left-singular vectors are eigenvectors of $AA^\top$.

The non-zero singular values are the square roots of non-zero eigenvalues of $A^\top A$ (or equivalently the square roots of non-zero eigenvalues of $AA^\top$)

Linear Algebra

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}$$

$$\|A\|_2 = \sup_{\|\vec{x}\|_2=1} \{\|A\vec{x}\|_2\} = \sigma_{1,1}(A)$$

(largest singular value)

$$\inf_{\|\vec{x}\|_2=1} \{\|A\vec{x}\|_2\} = \sigma_{n,n}(A)$$ (smallest singular value)

Positive definiteness:

- Two equivalent definitions:
  - (1) all eigenvalues non-negative,
  - (2) associated quadratic form is always non-negative.
- Special case:
  - $A^\top A \succcurlyeq 0$

Calculus

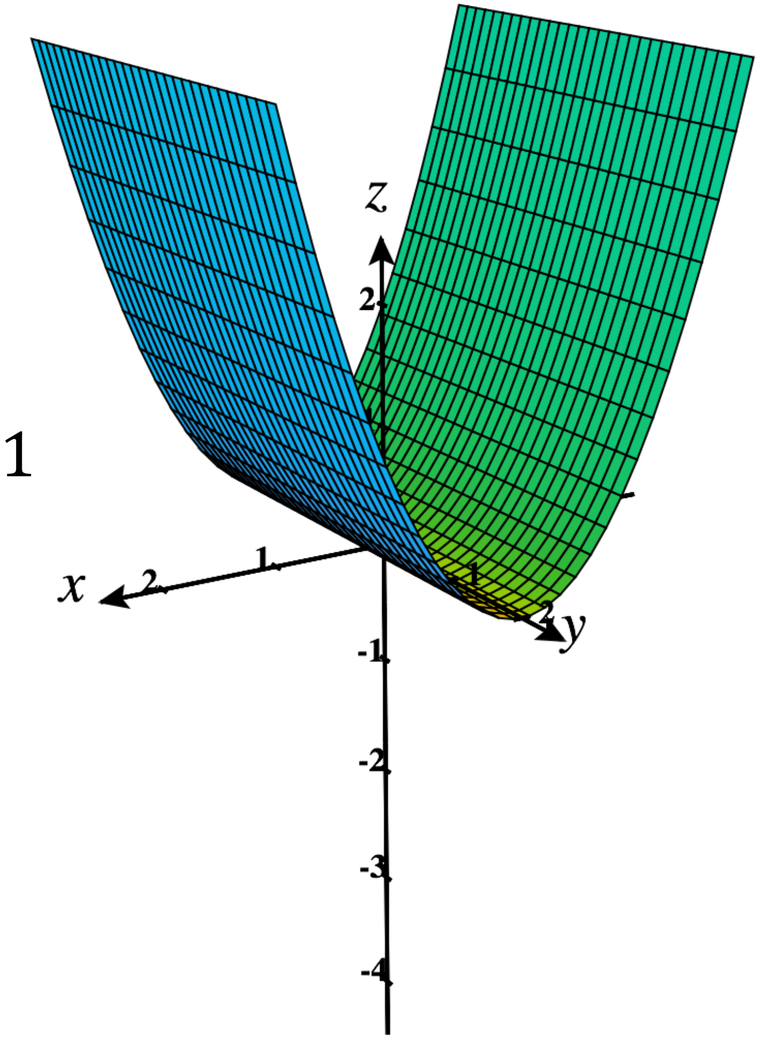
Convex function:

Two equivalent definitions:

- A line segment between any two points on the surface lies on or above the surface:
  $\alpha f(\vec{x}) + (1-\alpha) f(\vec{y}) \geq f(\alpha\vec{x} + (1-\alpha)\vec{y})$ where $0 \leq \alpha \leq 1$
- Hessian of function is positive semi-definite everywhere.

Properties:

- Every critical point is a local minimum.
- Every local minimum is a global minimum.



Calculus

Matrix derivatives:

$$\frac{\partial(\vec{a}^\top \vec{x})}{\partial \vec{x}} = \vec{a}$$

$$\frac{\partial(A\vec{x})}{\partial \vec{x}} = A^\top$$

Sum Rule:

$$\frac{\partial\left(\vec{f}(\vec{x}) + \vec{g}(\vec{x})\right)}{\partial \vec{x}} = \frac{\partial \vec{f}}{\partial \vec{x}} + \frac{\partial \vec{g}}{\partial \vec{x}}$$

Product Rule (for inner products):

$$\frac{\partial\left(\vec{f}(\vec{x})^\top \vec{g}(\vec{x})\right)}{\partial \vec{x}} = \frac{\partial \vec{f}}{\partial \vec{x}}\vec{g}(\vec{x}) + \frac{\partial \vec{g}}{\partial \vec{x}}\vec{f}(\vec{x})$$

Hence,

$$\frac{\partial(\vec{x}A\vec{x})}{\partial \vec{x}} = (A + A^\top)\vec{x}$$

Advanced Courses

Advanced Courses

Core Machine Learning:

CMPT 727: Statistical Machine Learning (More probabilistic models)

CMPT 728: Deep Learning (More advanced neural networks)

CMPT 983: Special Topics on Deep Generative Models (Models for generating complex data, such as images, text, etc.) – Ke Li

Applications of Machine Learning:

CMPT 762: Computer Vision (Advanced neural networks for extracting information from images) – Yasu Furukawa

CMPT 720: Robotics (Classical and data-driven methods in robotic systems) – Mo Chen

CMPT 825: Natural Language Processing (Machine learning methods for extracting information from text) – Angel Chang