Machine & Deep Learning

Pr Rahhal ERRATTAHI
rahhal.errattahi@um6p.ma
errattahi.r@ucd.ac.ma

Lecture 03
Linear Models

Lecture 3 Overview

- Linear models
- Linear models for regression
- Linear models for Classification
- Linear Models for multiclass classification
- Linear models overview
- Summary

Lecture 3 Overview

- Linear models
- Linear models for regression
- Linear models for Classification
- Linear Models for multiclass classification
- Linear models overview
- Summary

ML: the basic recipe

- Abstract your problem to a **standard task.**
  - **Classification**, **Regression**, Clustering, Density estimation, Generative Modeling, Online learning, Reinforcement Learning,
- Choose your **instances** and their **features**.
  - For supervised learning, choose a target.
- Choose your **model class**.
  - **Linear models**
- **Search** for a good model.
  - Choose a **loss function**, choose a **search method** to minimise the loss.

ML: problem setup

Problem Setting:

- Set of possible instances X
- Dataset D, given by $D = \{< \vec{x}_i, y_i >, \ldots, < \vec{x}_n, y_n >\} \subseteq X \times Y$

Where:

- $\vec{x}_i$ is a feature vector ($\mathbb{R}^d$),
- y_i is a label / target variable,
- X is space of all features and
- Y is space of labels.

- Unknown target function $f : X \rightarrow Y$
- Set of function hypotheses $H = \{h | h : X \rightarrow Y\}$

Linear models

- Linear models make a prediction using a linear function of the input features X

$$f_w(X) = \sum_{i=1}^{P} w_i \cdot x_i + w_0$$

- Learn w from$X$, given a loss function $\mathcal{L}$:

$$\underset{w}{\operatorname{argmin}} \mathcal{L}(f_w(X))$$

Linear models

- Many algorithms with different $\mathcal{L}$: Least squares, Ridge, Lasso, Logistic Regression, Linear SVMs,…
- Can be very powerful (and fast), especially for large datasets with many features.
- Can be generalized to learn non-linear patterns: *Generalized Linear Models*
  - Features can be augmented with polynomials of the original features
  - Features can be transformed according to a distribution (Poisson, Tweedie, Gamma,…)
  - Some linear models (e.g. SVMs) can be *kernelized* to learn non-linear functions

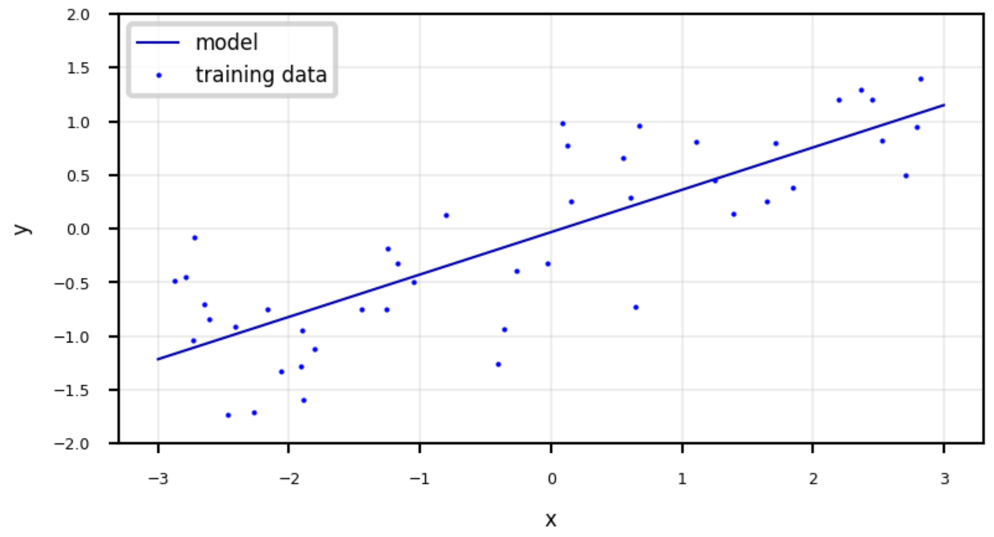
Lecture 3 Overview

- Linear models
- Linear models for regression
- Linear models for Classification
- Linear Models for multiclass classification
- Linear models overview
- Summary

Linear models for regression

- Prediction formula for input features x:
  - $w_1 \dots w_p$ usually called weights or coefficients , w_0 the bias or intercept
  - Assumes that errors are $N(0, \sigma)$

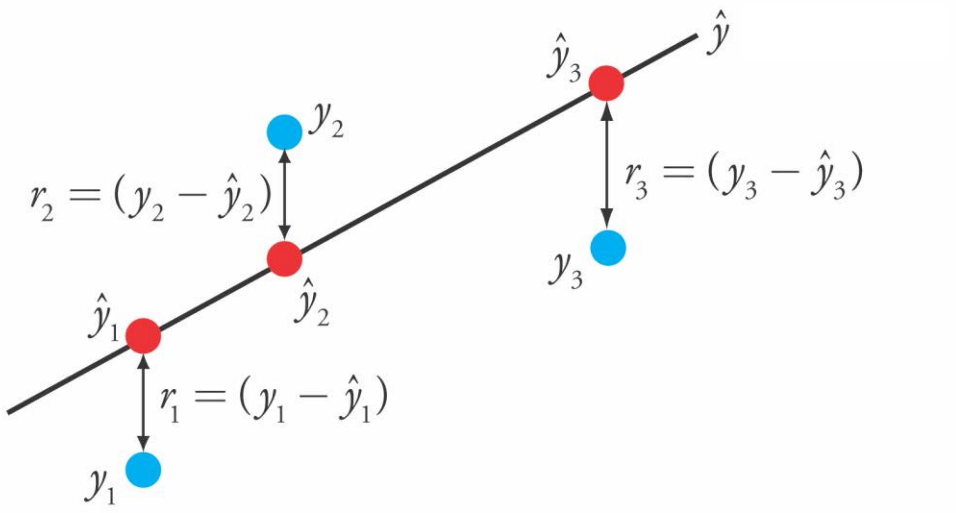
$$\hat{y} = WX + w_0 = \sum_{i=1}^{P} w_i \cdot x_i + b$$



Ordinary Least Squares (OLS)

- Loss function is the *sum of squared errors (SSE)* (or residuals) between predictions $\hat{y}_i$ (red) and the true regression targets y_i (blue) on the training set.

$$\mathcal{L}_{SSE} = \sum_{n=1}^{N} (y_n - \hat{y}_i)^2 = \sum_{n=1}^{N} \left(y_n - (W \cdot X_n + w_0)\right)^2$$

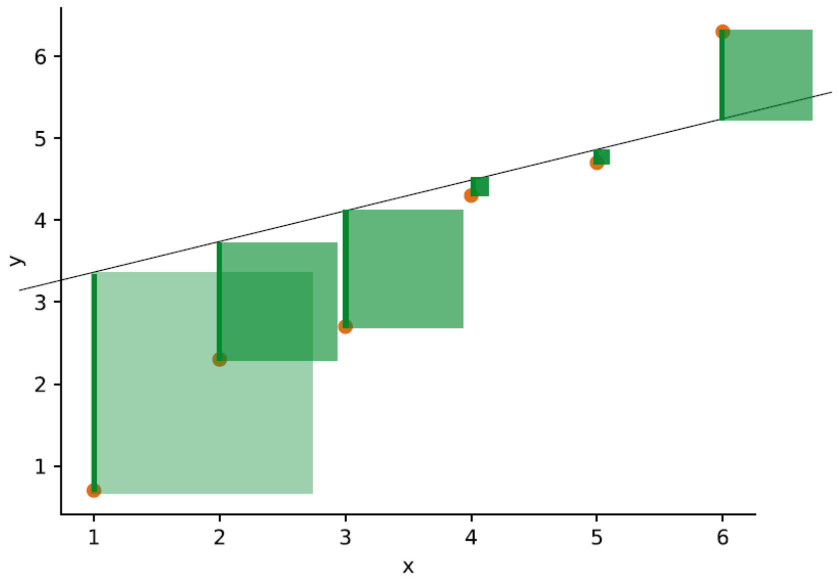


$\hat{y}$
$\hat{y}_3$
y_2
$r_2 = (y_2 - \hat{y}_2)$
$r_3 = (y_3 - \hat{y}_3)$
y_3
$\hat{y}_2$
$\hat{y}_1$
$r_1 = (y_1 - \hat{y}_1)$
y_1

Mean squared error (MSE)

- The MSE loss takes the residual for each instance in our data, squares them, and returns the average.

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_i)^2 = \frac{1}{N}\sum_{n=1}^{N}\left(y_n - (W \cdot X_n + w_0)\right)^2$$

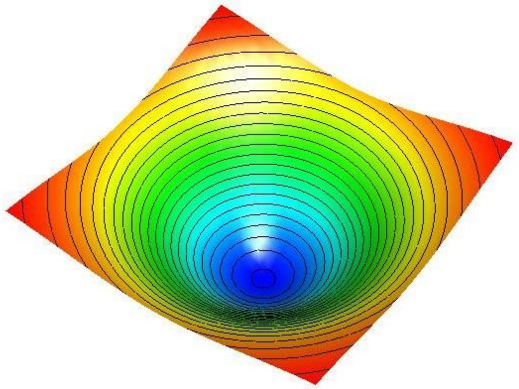
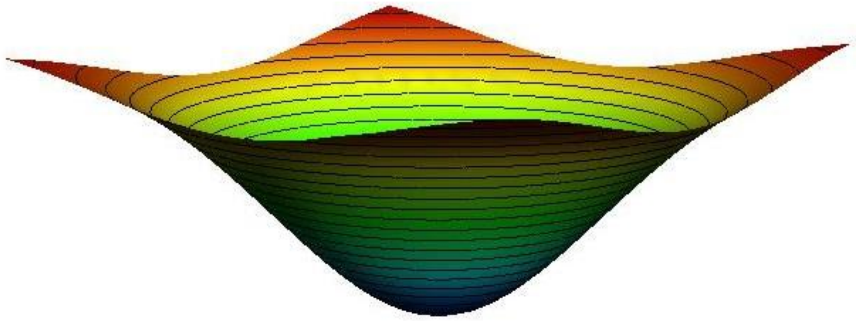


Optimization

$$p^* = \underset{p}{\operatorname{argmin}}\, \mathcal{L}(f_p(X))$$

In our example:

$$p = \{w_1 \ldots w_p, w_0\}$$

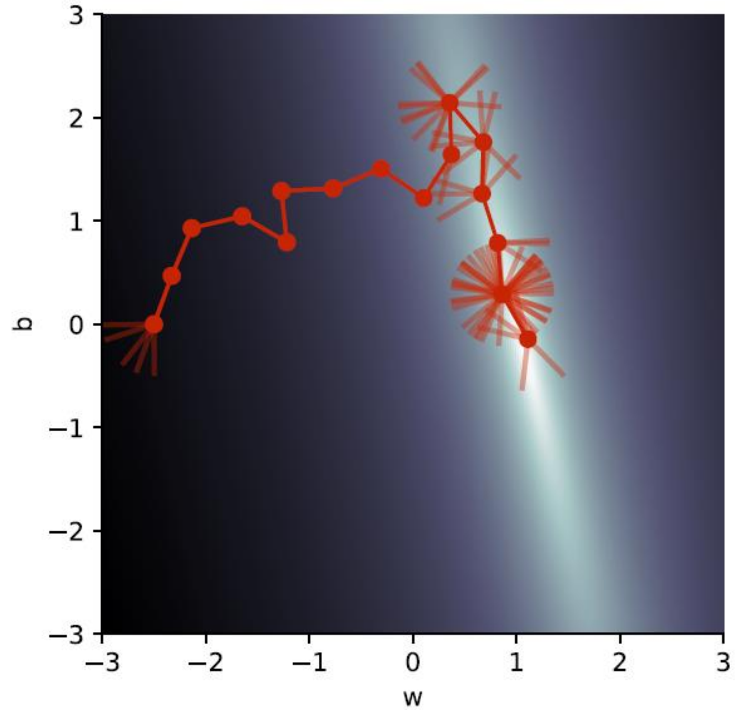



Optimization: Random Search

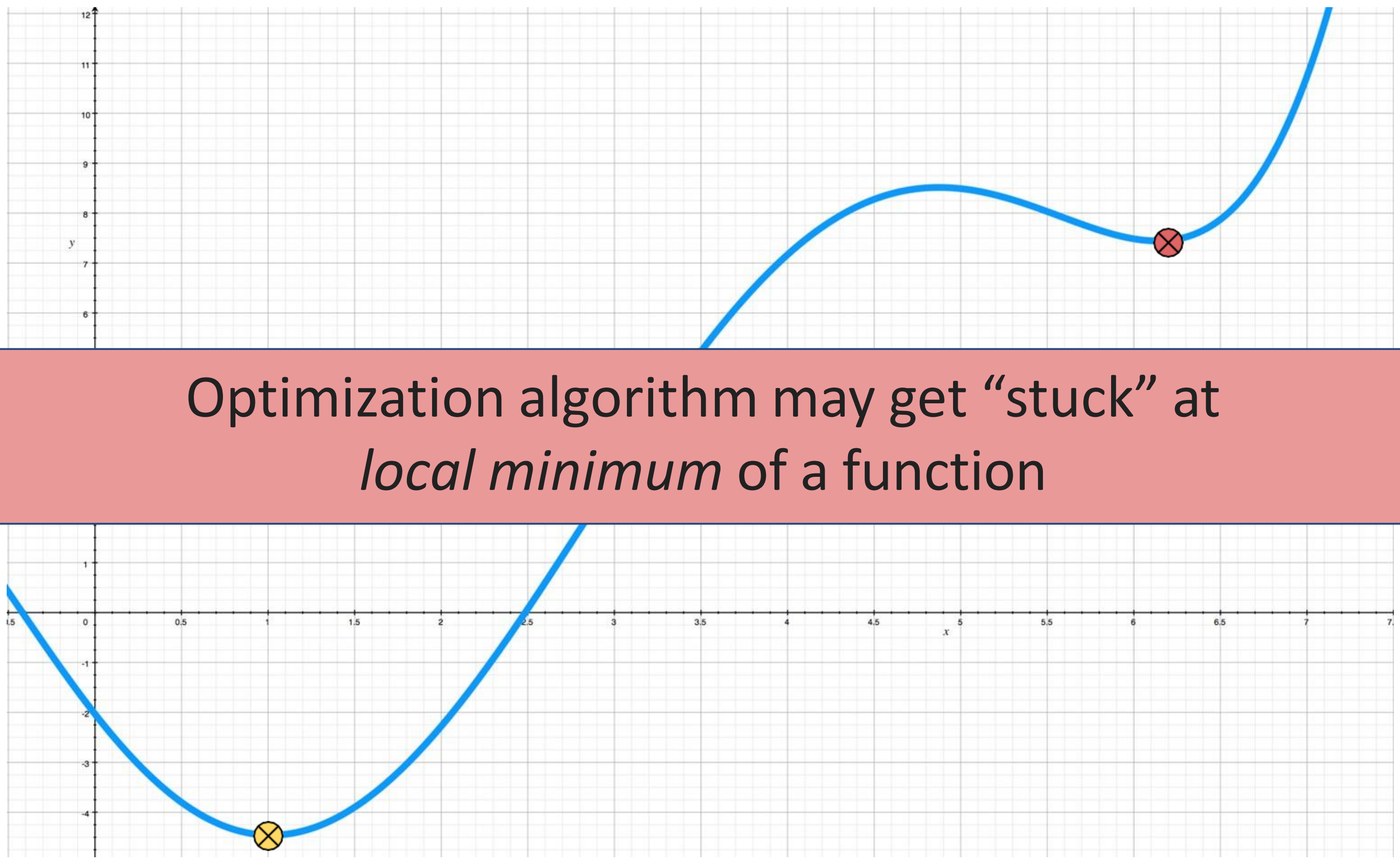
```
start with a random point **p** in the model space
loop:
  pick a random point p' close to p
  if loss(p') < loss(p):
      p <- p'
```

Optimization: Random Search

- To implement the random search we need to define how to pick a point "close to" another in model space.
- One simple option is to choose the next point by sampling uniformly among all points with some pre-chosen distance **r** from the current point.



Optimization: Random Search



Optimization algorithm may get “stuck” at
local minimum of a function

⊗ Minimum value of the function

⊗ Local minimum value of the function

Optimization: Gradient Descent

- **Q:** Imagine you are blindfolded on a mountain, how will you go to the bottom?

- **A:** Sense the slope around you, and move in the direction where the slope points downwards

Optimization: Gradient Descent

Intuition: walking downhill using only the slope you “feel” nearby

Optimization: Gradient Descent

- Concept of gradient == “your sense of slope” for the loss function
- The slope of a linear function is simply **how much it moves up** if we move one step to the right

- The gradient of a function is mathematically defined as the slope of the tangent i.e. slope at any given point on the function

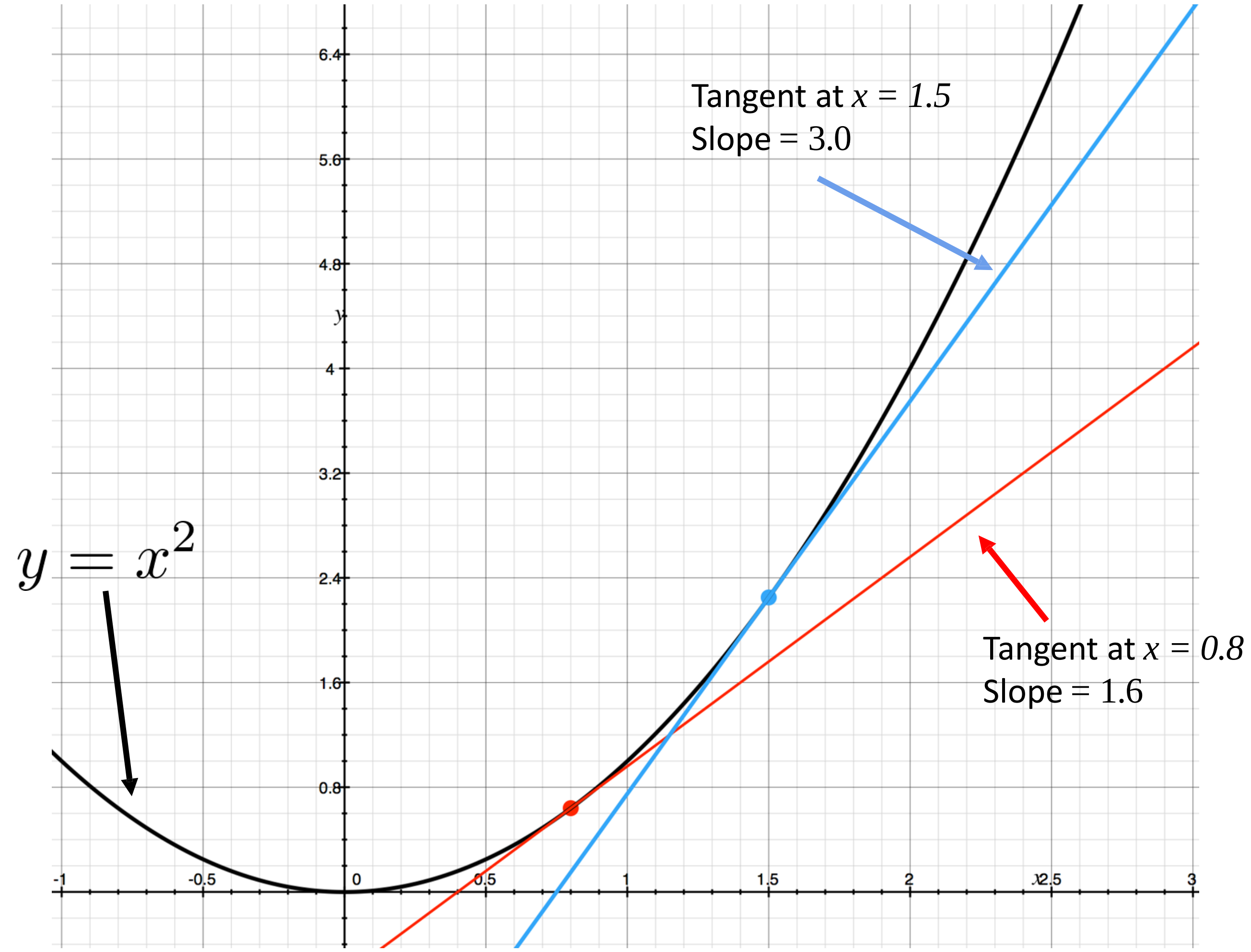
Optimization: Gradient Descent

Gradients:

- A *gradient* ∇f is the derivative of a function in multiple dimensions
  - It is a vector of partial derivatives: $\nabla f = \left[\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}, \ldots\right]$
  - E.g. $f = 2x_0 + 3x_1^2 - \sin(x_2) \rightarrow \nabla f = [2, 6x_1, -cos(x_2)]$

- Example: $f = -(x_0^2 + x_1^2)$
  - $\nabla f = \left[\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}\right] = [-2x_0, -2x_1]$
  - Evaluated at point (-4,1): $\nabla f(-4, 1) = [8, -2]$
    - These are the slopes at point (-4,1) in the direction of x_0 and x_1 respectively

Optimization: Gradient Descent



Optimization: Gradient Descent

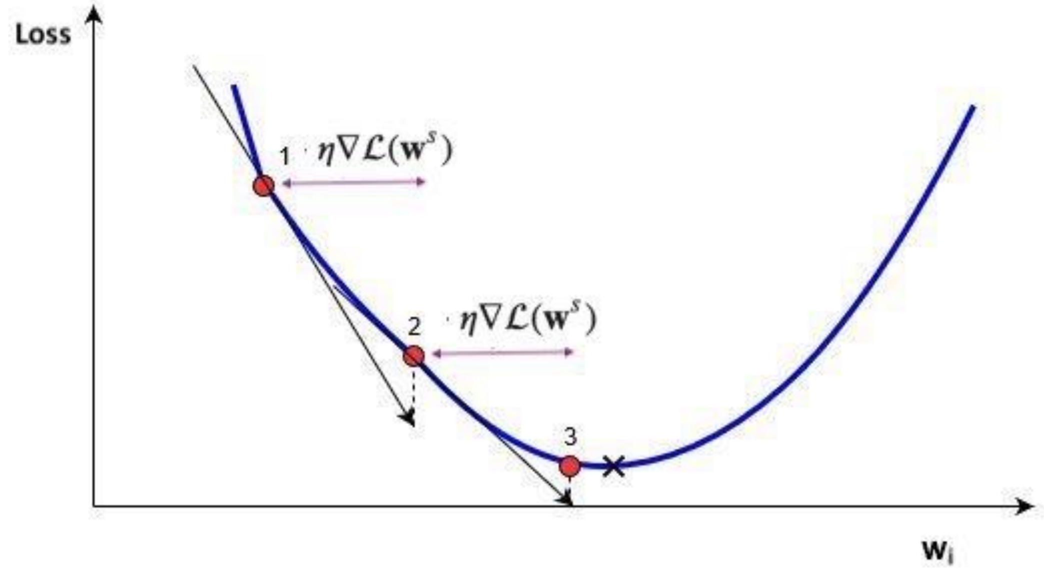
Gradient Descent:

- Start with an initial, random set of weights W^0:
- Given a differentiable loss function $\mathcal{L}$ (e.g. $\mathcal{L}_{\mathrm{SSE}}$), compute $\nabla\mathcal{L}$
- For least squares: $\frac{\partial\mathcal{L}_{SSE}}{\partial w_i}(W) = -2\sum_{n=1}^{N}(y_n - \hat{y}_n)x_{n,i}$
  - If feature $X_{:,i}$ is associated with big errors, the gradient wrt w_i will be large
- Update all weights slightly (by step size or learning rate η) in 'downhill' direction.
- Basic update rule (step s):

$$w^{S+1} = w^S - \eta\nabla\mathcal{L}(w^S)$$

Optimization: Gradient Descent

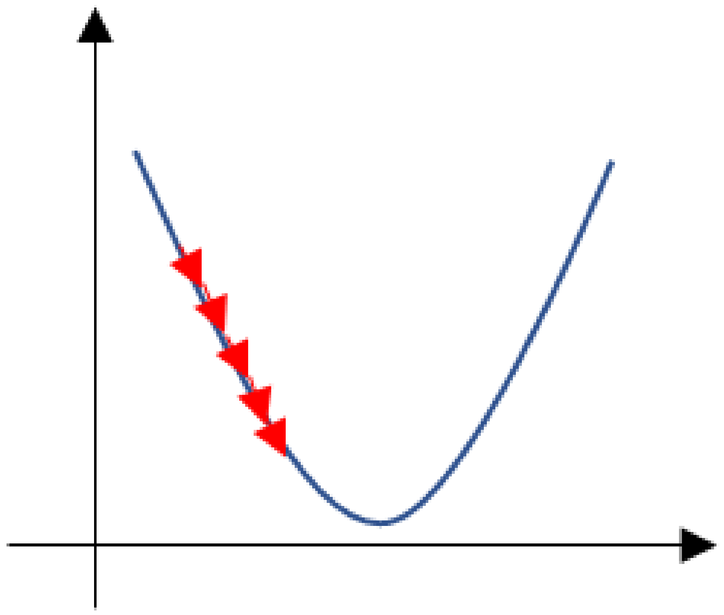
Gradient Descent:


Loss
1
$\eta\nabla\mathcal{L}(\mathbf{w}^s)$
2
$\eta\nabla\mathcal{L}(\mathbf{w}^s)$
3
w_i

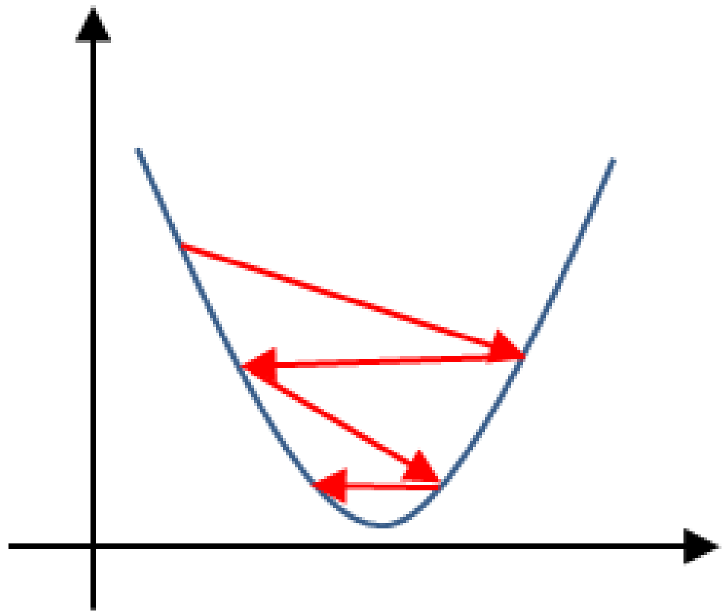
Optimization: Gradient Descent

- Learning rate:

Too small: slow convergence.



Too large: possible divergence



Optimization: Gradient Descent

- Maximum number of iterations
  - Too small: no convergence. Too large: wastes resources
- Learning rate decay with decay rate
  - E.g. exponential ($\eta^{s+1} = \eta^0 e^{-ks}$), inverse-time ($\eta^{s+1} = \frac{\eta^s}{1+ks}$),…
- Many more advanced ways to control learning rate
  - Adaptive techniques: depend on how much loss improved in previous step

Ridge regression

- Adds a penalty term to the least squares loss function:

$$\mathcal{L}_{Ridge} = \sum_{n=1}^{N} \left(y_n - (W \cdot X_n + w_0)\right)^2 + \alpha \sum_{i=1}^{P} w_i^2$$

- Model is penalized if it uses large coefficients (W)
  - Each feature should have as little effect on the outcome as possible
  - We don't want to penalize w_0, so we leave it out
- Regularization: explicitly restrict a model to avoid overfitting.
  - Called L2 regularization because it uses the L2 norm:$\alpha \sum_{i=1}^{P} w_i^2$
- The strength of the regularization can be controlled with the α hyperparameter

Lasso (Least Absolute Shrinkage and Selection Operator)

- Adds a different penalty term to the least squares sum:

$$\mathcal{L}_{Lasso} = \sum_{n=1}^{N} \left(y_n - (W \cdot X_n + w_0)\right)^2 + \alpha \sum_{i=1}^{P} |w_i|$$

- Called L1 regularization because it uses the L1 norm
  - Will cause many weights to be exactly 0
- Same parameter α to control the strength of regularization.
  - Will again have a ‘sweet spot’ depending on the data

Elastic-Net

- Adds both L1 and L2 regularization:

$$\mathcal{L}_{Elastic} = \sum_{n=1}^{N} \left(y_n - (W \cdot X_n + w_0)\right)^2 + \alpha\rho \sum_{i=1}^{P} |w_i| + \alpha(1-\rho) \sum_{i=1}^{P} w_i^2$$

- ρ is the L1 ratio
  - With $\rho = 1$, $\mathcal{L}_{Elastic} = \mathcal{L}_{Lasso}$
  - With $\rho = 0$, $\mathcal{L}_{Elastic} = \mathcal{L}_{Ridge}$
  - $0 < \rho < 1$ sets a trade-off between L1 and L2.
- Allows learning sparse models (like Lasso) while maintaining L2 regularization benefits
  - E.g. if 2 features are correlated, Lasso likely picks one randomly, Elastic-Net keeps both

Lab 4 - Linear Models for Regression

Lecture 3 Overview

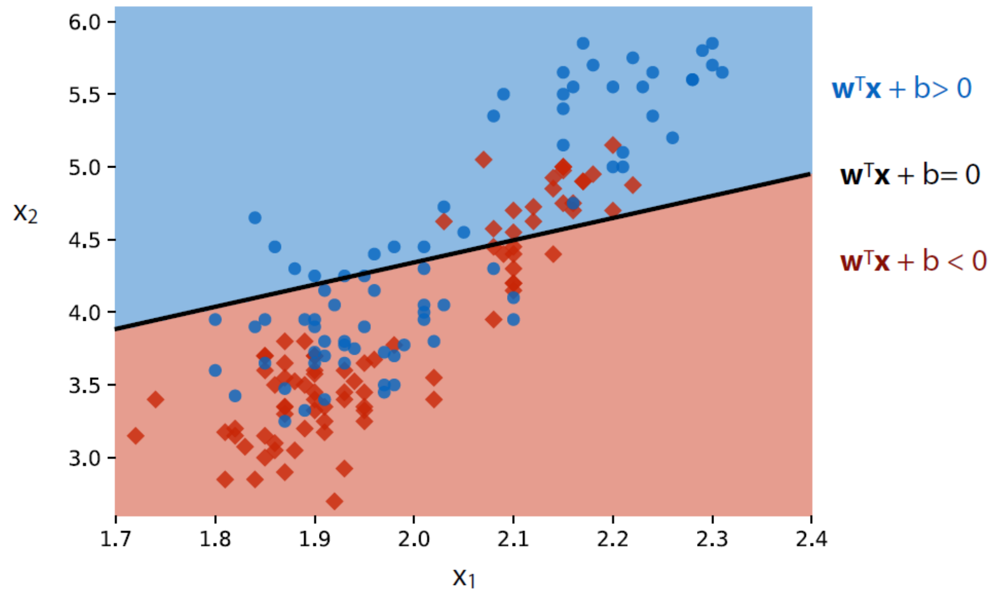
- Linear models
- Linear models for regression
- Linear models for Classification
- Linear Models for multiclass classification
- Linear models overview
- Summary

Linear models for Classification

- Aims to find a hyperplane that separates the examples of each class.
- For binary classification (2 classes), we aim to fit the following function:

$$\hat{y} = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_P \cdot x_P + w_0$$

When $\hat{y} < 0$, predict class -1, otherwise predict class +1



Linear models for Classification

- There are many algorithms for linear classification, differing in **loss function**, **regularization techniques**, and **optimization method**
- Most common techniques:
  - Convert target classes {neg,pos} to {0,1} and treat as a regression task
    - Logistic regression (Log loss)
    - Ridge Classification (Least Squares + L2 loss)
  - Find hyperplane that maximizes the margin between classes
    - Linear Support Vector Machines (Hinge loss)
  - Neural networks without activation functions
    - Perceptron (Perceptron loss)
  - SGDClassifier: can act like any of these by choosing loss function
    - Hinge, Log, Modified_huber, Squared_hinge, Perceptron

Logistic regression

- Aims to predict the *probability* that a point belongs to the positive class
- Converts target values {negative (blue), positive (red)} to {0,1}
- Fits a *logistic* (or *sigmoid* or *S* curve) function through these points
  - Maps (-Inf,Inf) to a probability [0,1]

$$\hat{y} = logistic\big(f_\theta(\mathrm{x})\big) = \frac{1}{1 + e^{-f_\theta(\mathrm{x})}}$$

  - E.g. in 1D:

$$\hat{y} = logistic(w_1 \cdot x_1 + b) = \frac{1}{1 + e^{-w_1 \cdot x_1 - w_0}}$$

Logistic regression

- Models that return class probabilities can use *cross-entropy loss*

$$L_{\log}(w) = \sum_{n=1}^{N} H(p_n, q_n) = -\sum_{n=1}^{N} \sum_{c=1}^{C} p_{n,c} log(q_{n,c})$$

- Also known as log loss, logistic loss, or maximum likelihood
- Based on true probabilities *p* (0 or 1) and predicted probabilities *q* over *N* instances and *C* classes
- Penalty (or surprise) grows exponentially as difference between *p* and *q* increases
- Often used together with L2 (or L1) loss

Ridge Classification

- Instead of log loss, we can also use ridge loss:

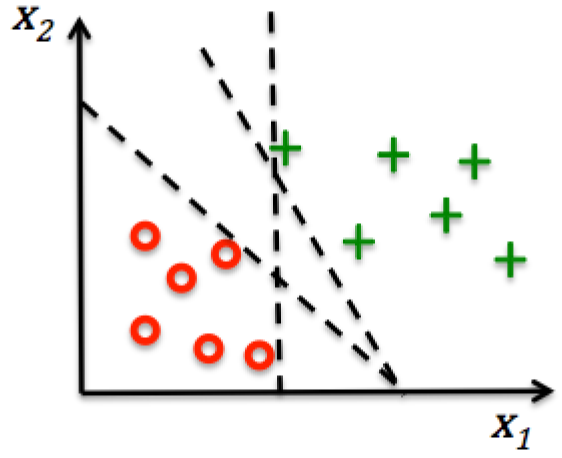
$$\mathcal{L}_{Ridge} = \sum_{n=1}^{N} (y_n - (\mathbf{w}\mathbf{x_n} + w_0))^2 + \alpha \sum_{i=1}^{p} w_i^2$$

- In this case, target values {negative, positive} are converted to {-1,1}
- Can be solved similarly to Ridge regression:
  - Closed form solution (a.k.a. Cholesky)
  - Gradient descent and variants
    - E.g. Conjugate Gradient (CG) or Stochastic Average Gradient (SAG,SAGA)
  - Use Cholesky for smaller datasets, Gradient descent for larger ones

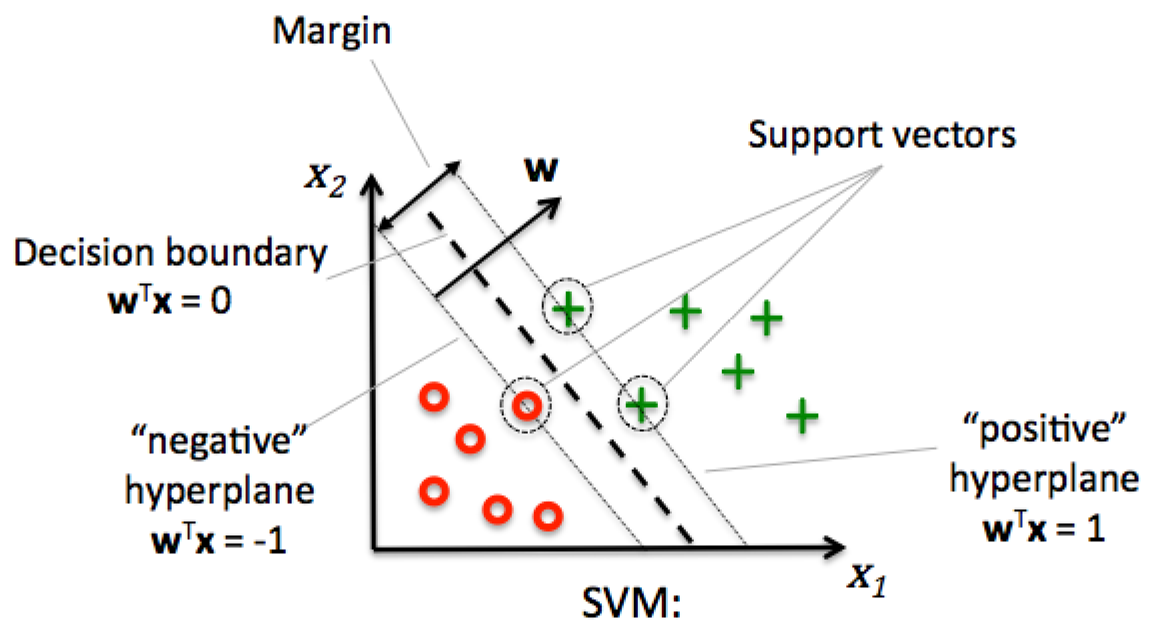
Support vector machines

- Decision boundaries close to training points may generalize badly
    - Very similar (nearby) test point are classified as the other class
- Choose a boundary that is as far away from training points as possible
- The **support vectors** are the training samples closest to the hyperplane
- The **margin** is the distance between the separating hyperplane and the *support vectors*
- Hence, our objective is to *maximize the margin*

Support vector machines



Which hyperplane?

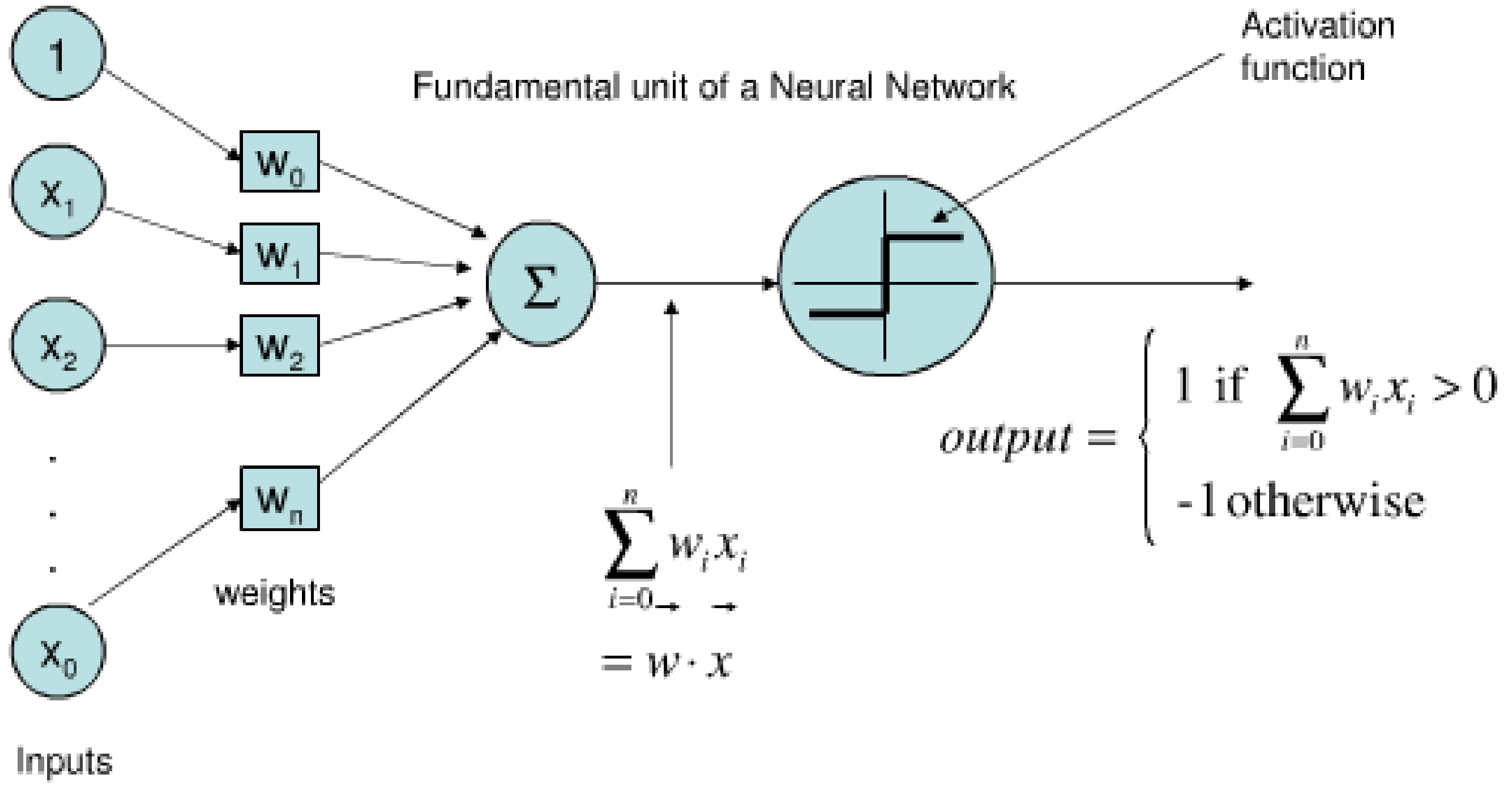


SVM:
Maximize the margin

Perceptron

- Represents a single neuron (node) with inputs x_i, a bias w_0, and output y
- Each connection has a (synaptic) weight w_i.
- The node outputs $\hat{y} = \sum_{i=1}^{n} w_i \cdot x_i + w_0$
- The *activation function* predicts 1 if $\mathrm{xw} + w_0 > 0$, -1 otherwise
- Weights can be learned with (stochastic) gradient descent and Hinge(0) loss
  - Updated *only* on misclassification, corrects output by ±1

Perceptron


1
x_1
x_2
x_0
Inputs
w_0
w_1
w_2
w_n
weights
Fundamental unit of a Neural Network
Σ
Activation function
$\sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

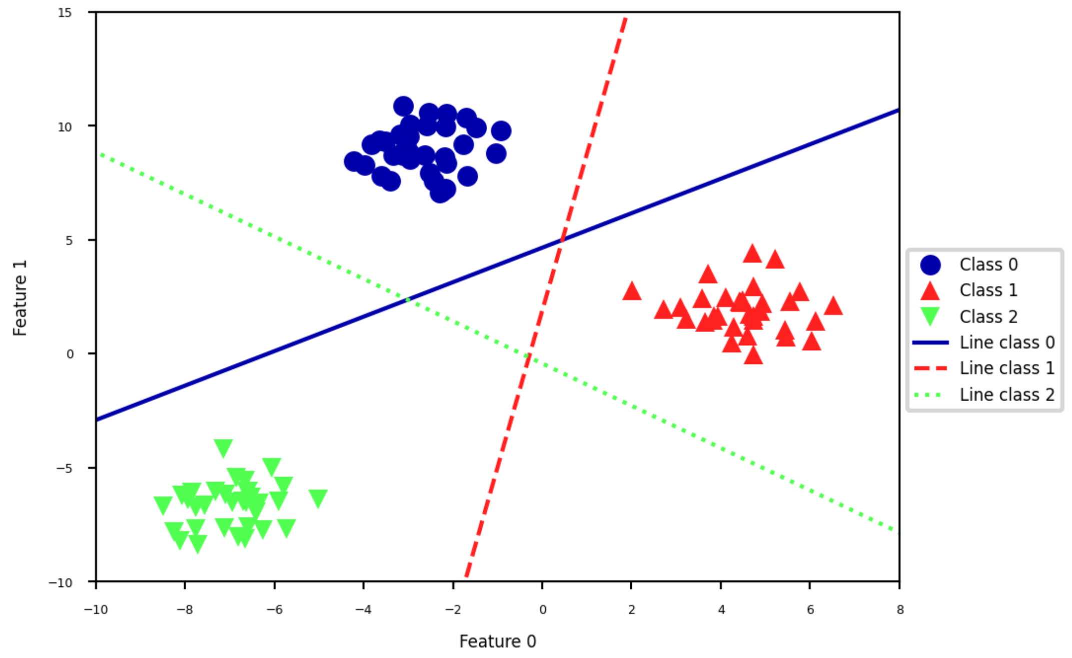
Lecture 3 Overview

- Linear models
- Linear models for regression
- Linear models for Classification
- Linear Models for multiclass classification
- Linear models overview
- Summary

Linear Models for multiclass classification

one-vs-rest (aka one-vs-all):

- Learn a binary model for each class vs. all other classes
- Create as many binary models as there are classes



Every binary classifiers makes a prediction, the one with the highest score (>0) wins

Linear Models for multiclass classification

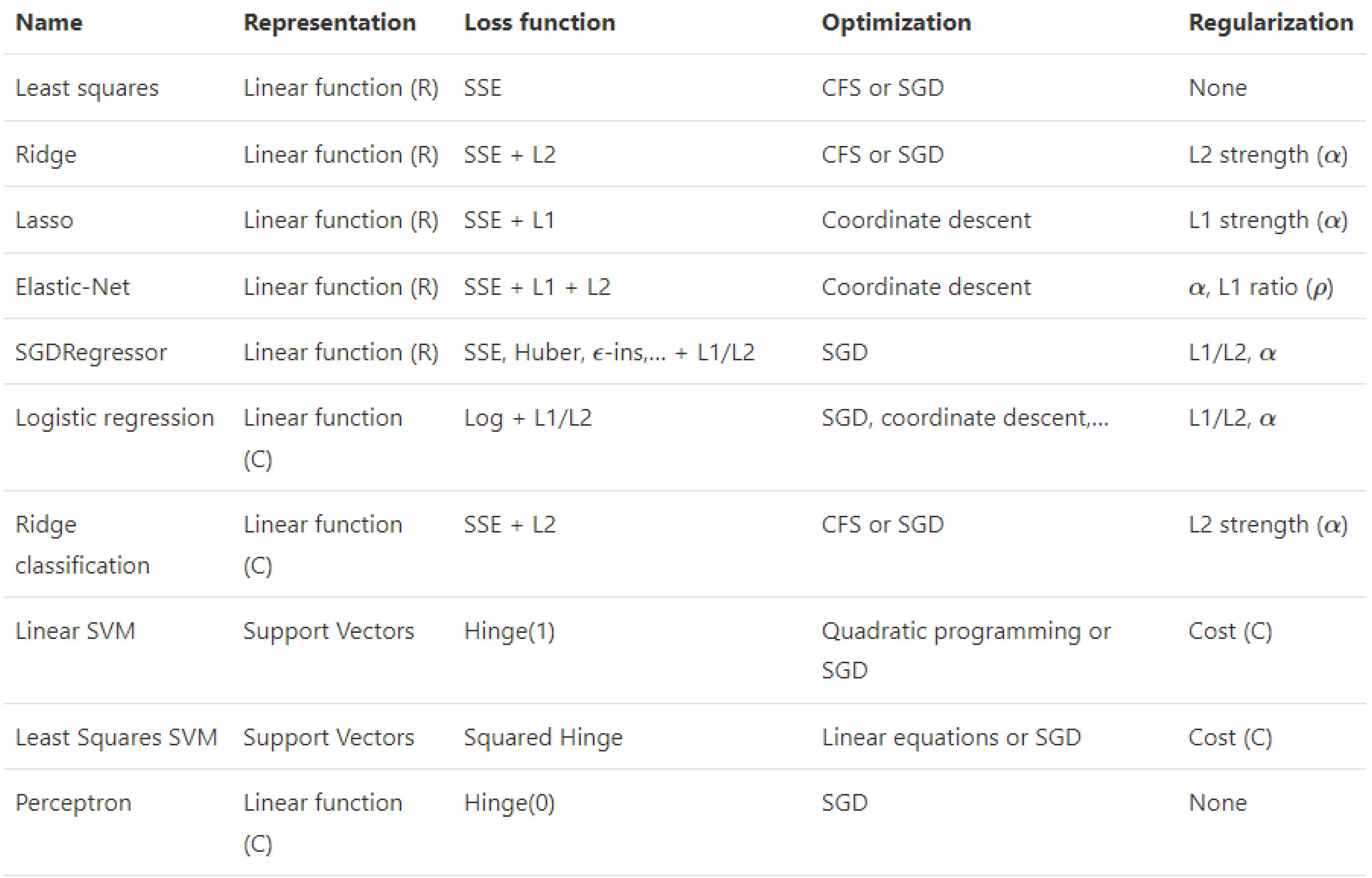
one-vs-one:

- An alternative is to learn a binary model for every *combination* of two classes
  - For C classes, this results in $\frac{C(C-1)}{2}$ binary models
  - Each point is classified according to a majority vote amongst all models
  - Can also be a 'soft vote': sum up the probabilities (or decision values) for all models. The class with the highest sum wins.
- Requires more models than one-vs-rest, but training each one is faster
  - Only the examples of 2 classes are included in the training data
- Recommended for algorithms than learn well on small datasets
  - Especially SVMs and Gaussian Processes

Lecture 3 Overview

- Linear models
- Linear models for regression
- Linear models for Classification
- Linear Models for multiclass classification
- Linear models overview
- Summary

Linear models overview



Name	Representation	Loss function	Optimization	Regularization
Least squares	Linear function (R)	SSE	CFS or SGD	None
Ridge	Linear function (R)	SSE + L2	CFS or SGD	L2 strength (α)
Lasso	Linear function (R)	SSE + L1	Coordinate descent	L1 strength (α)
Elastic-Net	Linear function (R)	SSE + L1 + L2	Coordinate descent	α, L1 ratio (ρ)
SGDRegressor	Linear function (R)	SSE, Huber, ϵ-ins,... + L1/L2	SGD	L1/L2, α
Logistic regression	Linear function (C)	Log + L1/L2	SGD, coordinate descent,...	L1/L2, α
Ridge classification	Linear function (C)	SSE + L2	CFS or SGD	L2 strength (α)
Linear SVM	Support Vectors	Hinge(1)	Quadratic programming or SGD	Cost (C)
Least Squares SVM	Support Vectors	Squared Hinge	Linear equations or SGD	Cost (C)
Perceptron	Linear function (C)	Hinge(0)	SGD	None

- SSE: Sum of Squared Errors
- CFS: Closed-form solution
- SGD: (Stochastic) Gradient Descent and variants
- (R)egression, (C)lassification

https://ml-course.github.io/

Lecture 3 Overview

- Linear models
- Linear models for regression
- Linear models for Classification
- Linear Models for multiclass classification
- Linear models overview
- Summary

Summary

- Linear models
  - Good for very large datasets (scalable)
  - Good for very high-dimensional data (not for low-dimensional data)
- Regularization is important. Tune the regularization strength ($\propto$)
  - Ridge (L2): Good fit, sometimes sensitive to outliers
  - Lasso (L1): Sparse models: fewer features, more interpretable, faster
  - Elastic-Net: Trade-off between both, e.g. for correlated features
- Most can be solved by different optimizers (solvers)
  - Closed form solutions or quadratic/linear solvers for smaller datasets
  - Gradient descent variants (SGD,CD,SAG,CG,…) for larger ones
- Multi-class classification can be done using a one-vs-all approach

Lab 5 - Linear Models for Classification