ESS2222

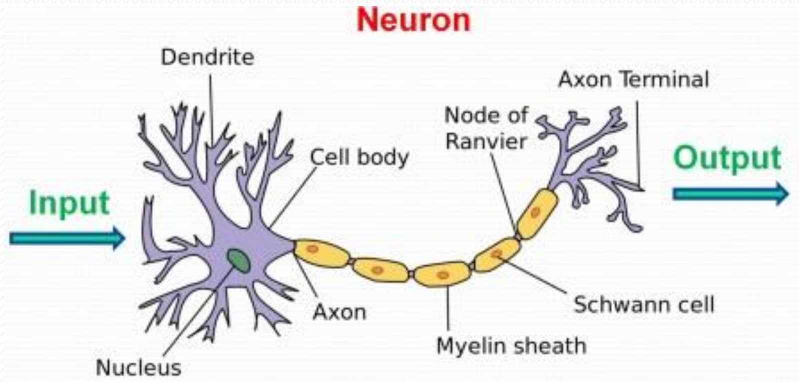
Lecture 4 – Linear model

Hosein Shahnas

University of Toronto, Department of Earth Sciences,

Outline

- ❑ **Logistic Regression**
- ❑ **Predicting Continuous Target Variables**
- ❑ **Support Vector Machine (Some Details)**
- ❑ **Nested Cross-Validation**
- ❑ **KNN - Algorithm**

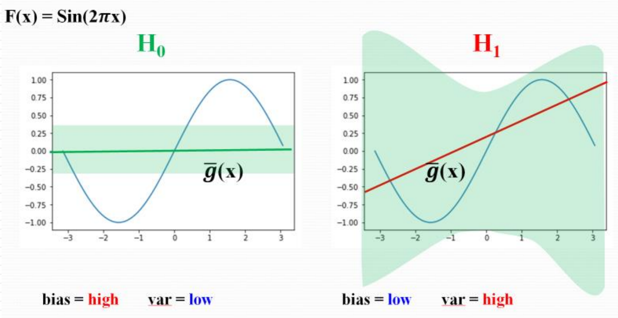


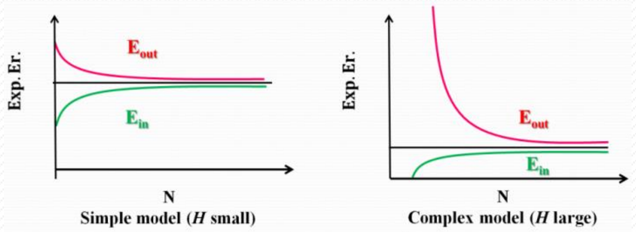
Review of Lecture 3

Bias-Variance Trade-off

$$E_D\left[E_X\left[\left(g^D(x) - f(x)\right)^2\right]\right] = \underbrace{E_X\left[E_D\left[\left[\left(g^D(x) - \bar{g}(x)\right)^2\right]\right]\right]}_{\text{var}} + \underbrace{E_X\left[\left[\left(\bar{g}(x) - f(x)\right)^2\right]\right]}_{\text{bias}}$$

$$\bar{g}(\mathrm{x}) = E_D\left[g^D(x)\right] \approx \frac{1}{k}\sum_k g^{D_k}(\mathrm{x})$$

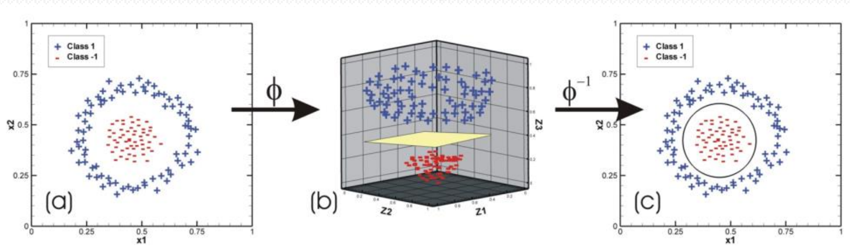




Cross-Validation

16%	16%	16%	16%	16%	20%
Training data	Training data	Training data	Training data	Validation	Testing data
Training data	Training data	Training data	Validation	Training data	Testing data
Training data	Training data	Validation	Training data	Training data	Testing data
Training data	Validation	Training data	Training data	Training data	Testing data
Validation	Training data	Training data	Training data	Training data	Testing data

Nonlinearity



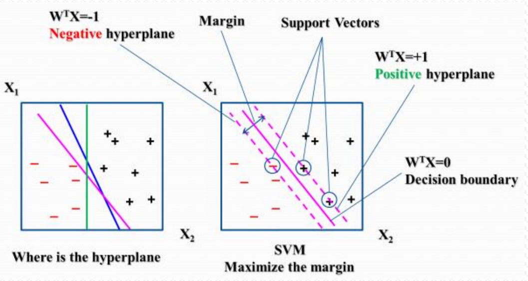
L2-rgression (Ridge)

$$J'_r(w) = \frac{1}{2}\sum_i\left(y^i - \phi(z^i)\right)^2 + \frac{\lambda}{2}\sum_i w^2{}_i$$

L1-rgression (Lasso)

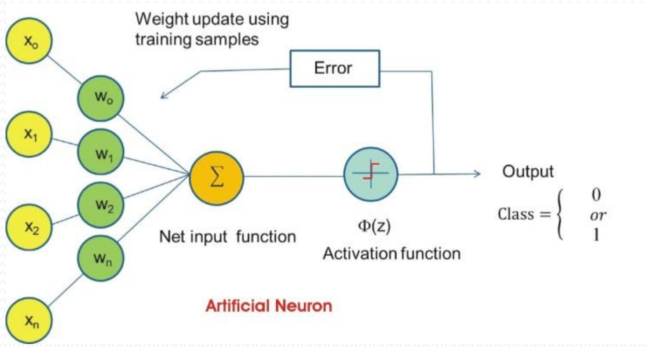
$$J'_l(w) = \frac{1}{2}\sum_i\left(y^i - \phi(z^i)\right)^2 + \frac{\lambda}{2}\sum_i |w_i|$$

$0 \leq \lambda < \infty$

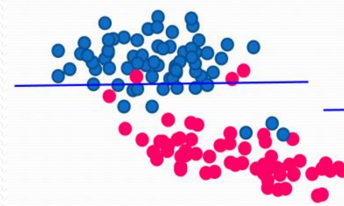


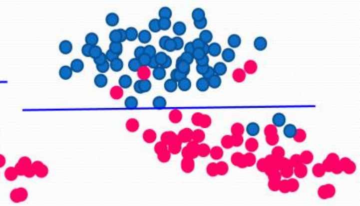
Perceptron Learning Algorithm Versus Pocket Algorithm

Suppose there exists small nonlinearity.



PLA **Pocket**

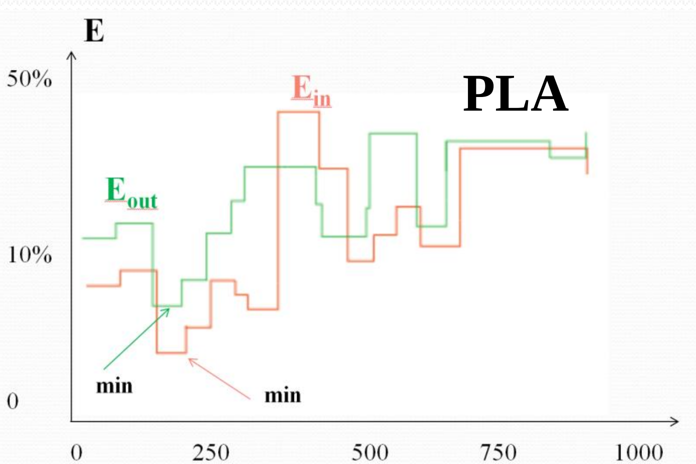


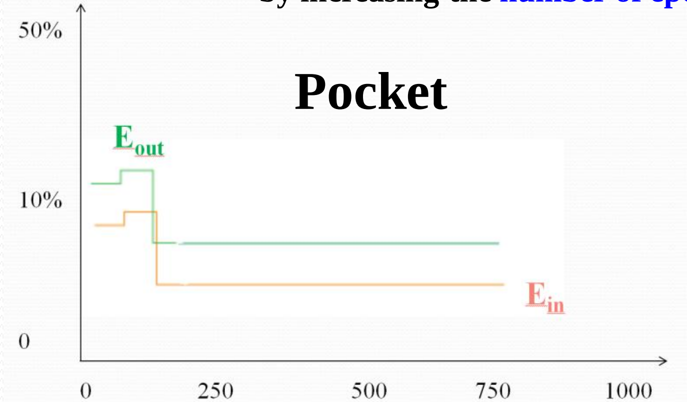


N = 1000 **N = 180**

$$Z = X.W = \sum_0^n w_i\, x_i = w_0\, x_0 + w_1\, x_1 + w_2\, x_2 + \dots + w_n\, x_n$$

In this problem convergence cannot be achieved by increasing the number of epochs





Store the best output and change it only for better outputs.

Logistic Regression

Modeling Class Probabilities via Logistic Regression

In order to overcome the convergence problem in Perceptron algorithm, we consider another simple yet more powerful algorithm for linear and binary classification problems. The method can be extended to multiclass classification via the One-vs.-Rest (OvR).
Note that, this is an algorithm for classification, not regression (despite the name).

Definition:
Odds for (or odds of, or odds in favour): Odds of event reflect the likelihood that the event will take place, while odds against reflect the likelihood that it will not.

In gambling:
The odds are the ratio of payoff to stake, and do not necessarily reflect exactly the probabilities'

$O_f = \frac{W}{L} = \frac{1}{O_a}$, $O_a = \frac{W}{L} = \frac{1}{O_f}$, $O_a \cdot O_a = 1$ W: winning, L: loosing

$P = \frac{W}{(W+L)} = 1 - q$, $q = \frac{L}{(W+L)} = 1 - p$, $P + q = 1$

Logistic Regression

$O_f = \frac{W}{L} = \frac{1}{O_a}$, $O_a = \frac{L}{W} = \frac{1}{O_f}$, $O_f \cdot O_a = 1$ Odds

$p = \frac{W}{(W+L)} = 1 - q$, $q = \frac{L}{(W+L)} = 1 - p$, $p + q = 1$ Probabilities

$O_f = \frac{p}{q} = \frac{p}{(1-p)} = \frac{(1-q)}{q}$, $O_a = \frac{q}{p} = \frac{(1-p)}{p} = \frac{q}{(1-q)}$,

Logistic Regression as a Probabilistic Model

The odds ratio for an event: $O_f = \frac{p}{(1-p)}$

The **logit** function is defend as: $\text{logit}(p) = log\left[\frac{p}{(1-p)}\right]$, $usually\ natural\ log.$

p: (0 – 1) → logit(p): (-∞ – ∞)

Logistic Regression

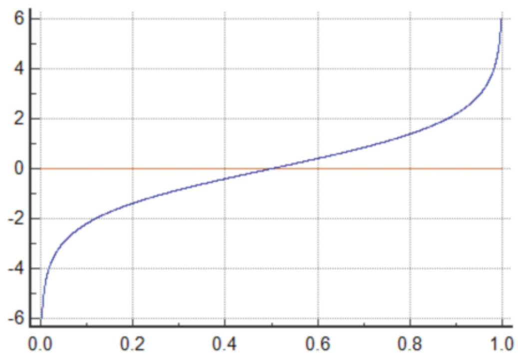
The inverse function of logit function is **logistic** function:

The inverse form of logit function is logistic function: $\phi(z) = \frac{1}{1+e^{-z}}$

z: (-∞ – ∞) → $\phi(z)$: (0 – 1)

$y = log\left(\frac{x}{1-x}\right)$, y→x x→y $x = log\left(\frac{y}{1-y}\right)$

→ $e^x = \frac{y}{1-y}$ → y= $\frac{1}{1+e^{-x}}$



logit(p)

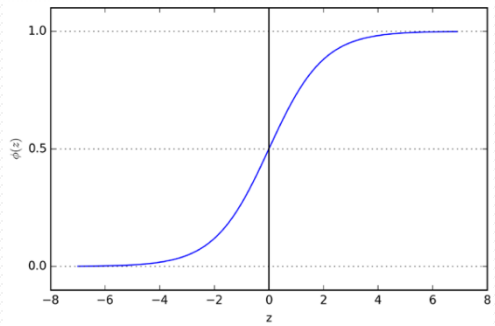
Logistic Regression

Logistic function, sometimes simply abbreviated as sigmoid function due to its characteristic S-shape.

z = x.w = $\sum_0^n w_i x_i = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m$ Net input

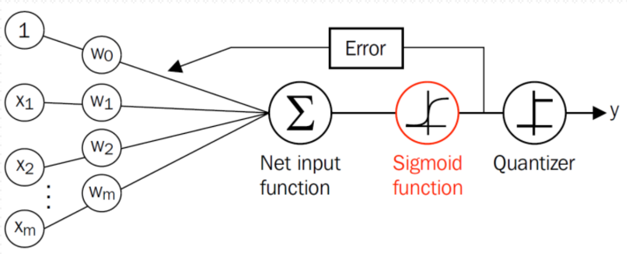
How do we interpret the sigmoid function?

The output of the sigmoid function is interpreted as the probability of particular sample belonging to y, given x (i.e., class 1, given features x parameterized by the weights w):



$\phi(z) = P(y=1|x,w)$, $\phi(z) = \frac{1}{1+e^{-z}}$,

$\hat{y} = \begin{cases} 1 & if\ \Phi(z) \geq \mathbf{0.5} \\ 0 & otherwise \end{cases}$ or $\hat{y} = \begin{cases} 1 & if\ z \geq \mathbf{0} \\ 0 & otherwise \end{cases}$



Logistic Regression – The Cost Function

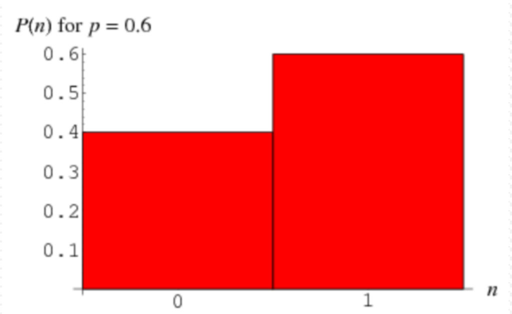
In Adeline algorithm we defined sum-squared-error cost function:

$$\boldsymbol{J(w) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2}$$

In order to learn the weights w we minimized this function.

Bernoulli Distribution

$$P(n) = \begin{cases} 1-p & for\ n = 0 \\ p & for\ n = 1 \end{cases}$$



$P(n) = p^n (1-p)^{(1-n)}$

Logistic Regression – The Cost Function

To derive the cost function for logistic regression, we define the likelihood function L, assuming that the individual samples in our dataset are independent of one another:

$$\boldsymbol{L}(\boldsymbol{w}) = \boldsymbol{P}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}) = \prod_{i=1}^{n} \boldsymbol{P}\left(\boldsymbol{y}^{i}, \boldsymbol{x}^{i}; \boldsymbol{w}\right) = \prod_{i=1}^{n} \left(\boldsymbol{\phi}(\boldsymbol{z}^{i})\right)^{\boldsymbol{y}^{i}} \left(\boldsymbol{1} - \boldsymbol{\phi}(\boldsymbol{z}^{i})\right)^{(\boldsymbol{1}-\boldsymbol{y}^{i})}$$

It is easy to work with (natural) log of this function for two reasons:

a) Applying the log function reduces the potential for numerical underflow, which can occur if the likelihoods are very small,
b) We can convert the product of factors into a summation of factors, which makes it easier to obtain the derivative of this function.

$$\boldsymbol{l}(\boldsymbol{w}) = \boldsymbol{log}\ \boldsymbol{L}(\boldsymbol{w}) = \sum_{i}^{n} \boldsymbol{y}^{i}\ \boldsymbol{log}\left(\boldsymbol{\phi}(\boldsymbol{z}^{i})\right) + \left(\boldsymbol{1} - \boldsymbol{y}^{i}\right)\ \boldsymbol{log}\left(\boldsymbol{1} - \boldsymbol{\phi}(\boldsymbol{z}^{i})\right)$$

Logistic Regression – The Cost Function

We can use an optimization algorithm such as gradient ascent to maximize this log-likelihood function. Alternatively, we can use gradient decent to minimize the following cost function:

$$J(w) = -\, l(w) = \sum_i^n -y^i \ log\left(\phi(z^i)\right) - \left(1-y^i\right) \ log\left(1-\phi(z^i)\right)$$

To get a better grasp on this cost function, let's calculate for one single-sample instance:

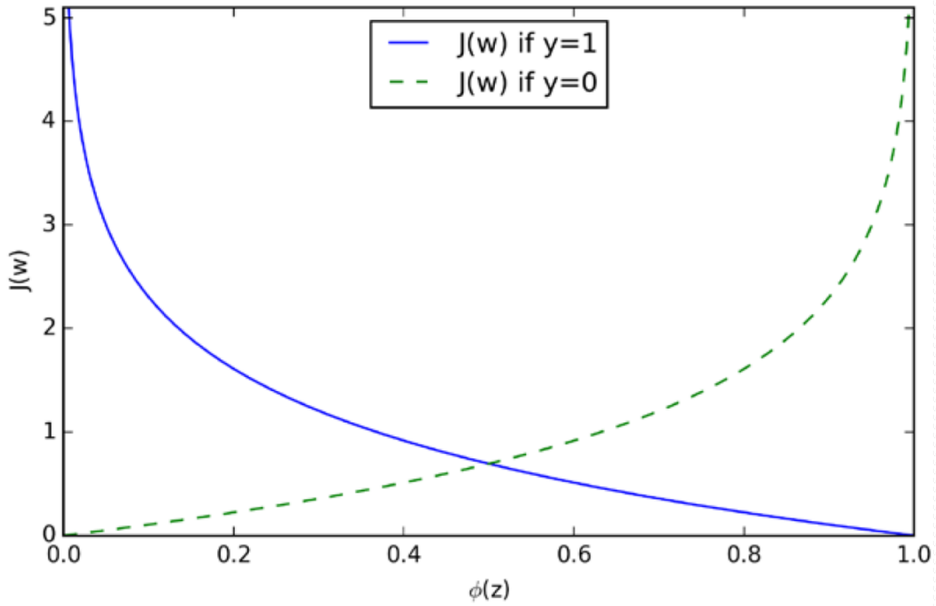
$$J(w) = -\, y \, log\left(\phi(z)\right) - \ (1-y) \ log(1-\phi(z))$$

It can be seen that the first term becomes zero if y = 0 and the second term becomes zero if y = 1, respectively:

$$J(\phi(z), y{:}\, w) = = \begin{cases} -log\left(\phi(z)\right) & if \ \ y = 1 \\ -log(1-\phi(z)) & if \ y = 0 \end{cases}$$

Logistic Regression – The Cost Function

$$J(\phi(z), y; w) = = \begin{cases} -log(\phi(z)) & if \;\; y = 1 \\ -log(1 - \phi(z)) & if \;\; y = 0 \end{cases}$$



Note that J → 0 (blue line) if the sample is correctly predicted as class 1. Similarly, J → 0 (green dashed line) if the sample is correctly predicted as class 0. However, for wrong prediction J → ∞. To minimize J, the wrong predictions should be penalized with an increasingly larger cost.

Logistic Regression – The Cost Function

We can show that the weight update in logistic regression via gradient descent is the same we obtained in Adaline:

$$J(w) = -y\, log\big(\phi(z)\big) - (1-y)\ log(1-\phi(z))$$

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial \phi(z)}\frac{\partial \phi(z)}{\partial w_j} = -\left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{(1-\phi(z))}\right)\frac{\partial \phi(z)}{\partial w_j}$$

$$\frac{\partial \phi(z)}{\partial w_j} = \frac{\partial \phi(z)}{\partial z}\frac{\partial z}{\partial w_j}, \qquad \textit{for} \quad \phi(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial \phi(z)}{\partial w_j} = \frac{e^{-z}}{(1-e^{-z})^2}\ \frac{\partial z}{\partial w_j} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)\frac{\partial z}{\partial w_j} = \phi(z)(1-\phi(z))\frac{\partial z}{\partial w_j}$$

$$\frac{\partial z}{\partial w_j} = x_j$$

$$\frac{\partial J}{\partial w_j} = -\left(y\frac{1}{\phi(z)} - (1-y)\frac{1}{(1-\phi(z))}\right)\phi(z)\,(1-\phi(z))\ x_j$$

Logistic Regression – The Cost Function

$$\frac{\partial J}{\partial w_j} = -\left(y \frac{1}{\phi(z)} - (1 - y) \frac{1}{(1-\phi(z))} \right) \phi(z)\,(1\ - \phi(z))\ x_j$$

$$= -\ \mathrm{y}\,(1\ - \phi(z))\ x_j + ((1 - y)\,\phi(z)\ x_j = -\ \ (y\ - \phi(z))\ x_j$$

$$\Delta wj = -\,\eta\,\frac{\partial J}{\partial w_j} = \eta \sum_i \left(y^i - \phi(z^i)\right) x^i{}_{\mathrm{j}}$$

Using sklearn:

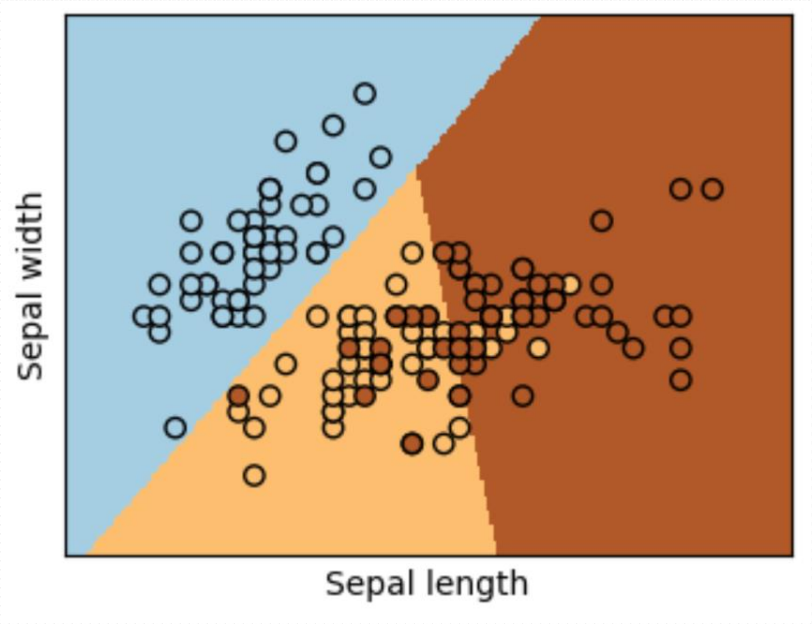
Example -

```
from sklearn.linear_model import LogisticRegression
Log_reg = LogisticRegression(C, random_state)
Log_reg.fit(X_train, y_train)
Log_reg.predict_proba(X_test[i])
```

>> 0.000, 0.063, 0.937 The probability that the test sample belongs to class **0**, **1,** and **2** are respectively **0.000, 0.063,** and **0.937**.

Log_reg.predict(X_test[i]) will return class **2** for this example,

Logistic Regression
Iris Example



https://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

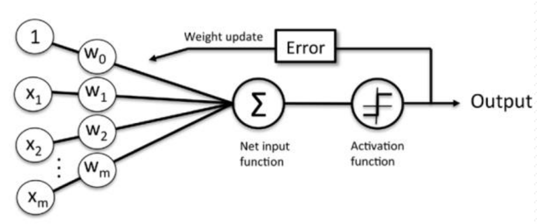
Predicting Continuous Target Variables

$$\mathbf{z(x)} = \sum_{i=0}^{m} w_i x_i = \mathbf{w}^{\mathrm{T}}\,\mathbf{x}$$

$$\boldsymbol{\phi}(\boldsymbol{z}) \equiv \mathbf{h(x)}$$

$\phi(z) = sign(z)$

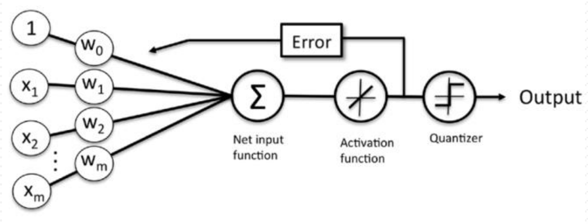
Perceptron



Discrete labels

$\phi(z) = z$ + **Quantizer**

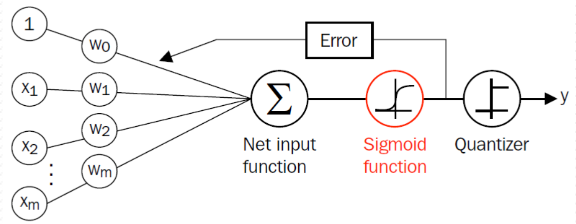
Linear regression (Adeline)



Discrete labels

$\phi(z) = \frac{1}{1+e^{-z}}$, + **Quantizer**

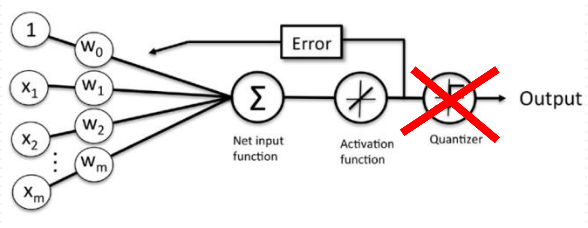
Logistic regression



Discrete labels

$\phi(z) = z$ + ~~**Quantizer**~~

Linear regression



Continuous outputs

Linear Regression

Examples:

Classification: **Credit approval (good/bad)**

(x_i, y_i) **$y_i \in$ [0, 1]**

Regression: **Credit line (dollar amount)**

(x_i, y_i) **$y_i \in \mathbb{R}$**

The error:

- **In classification problem** **we just count the number of wrong prediction (the frequency) compared to the total number of estimations.**

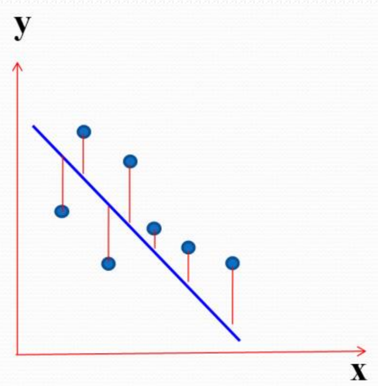
$$E_{in}(h) = \frac{1}{N}\sum_{1}^{N} [\![h(x_n) \neq f(x_n)]\!]$$

- **In regression problem:**

$$E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}\left(h(x_n) - y_n\right)^2$$

$$E_{in}(w) = \equiv \frac{1}{N}\ \sum_{n=1}^{N}\left(w^T x_n - y_n\right)^2 \ \equiv \frac{1}{N}\ \|Xw - y\|^2$$ **in-sample error**

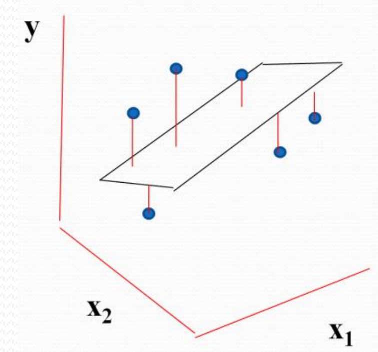
Linear Regression
Minimizing $E(h)_{in}$



$$E_{in}(w) = \equiv \frac{1}{N} \sum_{n=1}^{N} \left(w^T x_n - y_n\right)^2 \equiv \frac{1}{N} \|Xw - y\|^2$$

$$\nabla_w E_{in}(w) = \frac{2}{N} X^T(Xw - y) = 0 \quad \Rightarrow \quad X^T X w = X^T y$$

$$(X^T X)^{-1} X^T X w = (X^T X)^{-1} X^T y \quad \Rightarrow \quad w = (X^T X)^{-1} X^T y$$



$$w = (X^T X)^{-1} X^T y$$

$w = X^{\dagger} y$ where $X^{\dagger} = (X^T X)^{-1} X^T$ pseudo-inverse

Linear Regression
Minimizing $E(h)_{in}$

$$\boldsymbol{X}^{\dagger} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$$

$x_i = [x_{i0}, x_{i1}, \quad .. \quad x_{im}]$
w= $[w_0, w_1, \quad .. \quad w_m]$

$$X = \begin{bmatrix} x_{10}, x_{11}, .. & x_{1m} \\ x_{20}, x_{21}, .. & x_{2m} \\ \\ x_{i0}, x_{i1}, .. & x_{im} \\ \\ \\ x_{N0}, x_{N1}, .. & x_{Nm} \end{bmatrix} \qquad X^T = \begin{bmatrix} x_{10}, x_{20}, .. & x_{N0} \\ x_{11}, x_{21}, .. & x_{N1} \\ \\ x_{1k}, x_{2k}, .. & x_{Nk} \\ \\ \\ x_{1m}, x_{2m}, .. & x_{Nm} \end{bmatrix}$$

$$wX^T = [y_1, y_N, \quad .. \quad y_N]$$

$$Xw^T = \begin{bmatrix} y_1 \\ y_2 \\ \\ \\ y_N \end{bmatrix}$$

Linear Regression
Minimizing E(h)$_{in}$

$$X^\dagger = (X^T X)^{-1} X^T$$

$$v_1 \underset{h}{\begin{bmatrix} \quad \end{bmatrix}} \; h \underset{h_2}{\begin{bmatrix} \quad \end{bmatrix}} = \underset{h_2}{\begin{bmatrix} \quad \end{bmatrix}} v_1$$

$$X^\dagger = \left(\underbrace{\begin{bmatrix} \qquad \end{bmatrix}}_{(m+1)\times N} \underbrace{\begin{bmatrix} \qquad \end{bmatrix}}_{N \times (m+1)} \right)^{-1} \underbrace{\begin{bmatrix} \qquad \end{bmatrix}}_{(m+1)\times N}$$

N: Number of samples
m: Number of features

Linear Regression
Minimizing $E(h)_{in}$

$$X^{\dagger} = (X^{T}X)^{-1}X^{T}$$

$$X^{\dagger} = \underbrace{\left(\underbrace{\begin{bmatrix} \quad & \quad \end{bmatrix}}_{(m+1)\times(m+1)} \right)^{-1} \underbrace{\begin{bmatrix} \quad & \quad \end{bmatrix}}_{(m+1)\times N}}_{(m+1)\times N}$$

N: Number of samples
m: Number of features

Linear Regression
Minimizing $E(h)_{in}$

Method:

- Construct matrix **X** and vector **y** sing data $(X_1, y_1), (X_2, y_2), \ldots\ldots, (X_m, y_m)$

$$X^{\dagger} = \begin{array}{cc} \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} & \text{m+1} \\ \text{N} & \end{array} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ \\ y_N \end{bmatrix} \qquad \text{w} = \begin{bmatrix} w_0 \\ w_1 \\ \\ \\ w_m \end{bmatrix}$$

- Compute $(X^T X)^{-1} X^T$

- Return $w = X^{\dagger} y$

Linear Regression For Classification

- **In linear regression learning the model learn a real valued function $y = f(x) \in R$**

- **In classification learning the binary-valued functions are also real valued $y = \pm 1 \in R$**

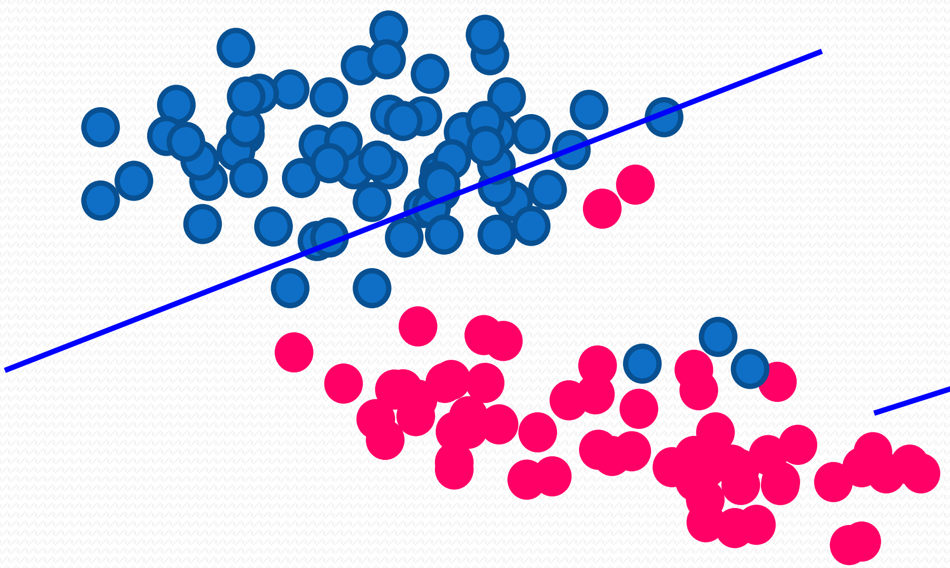
- **We can use linear regression to obtain w ($w^T x = y$)**

- **Sign($w^T x_j$) likely agrees with $y_j = \pm 1$**

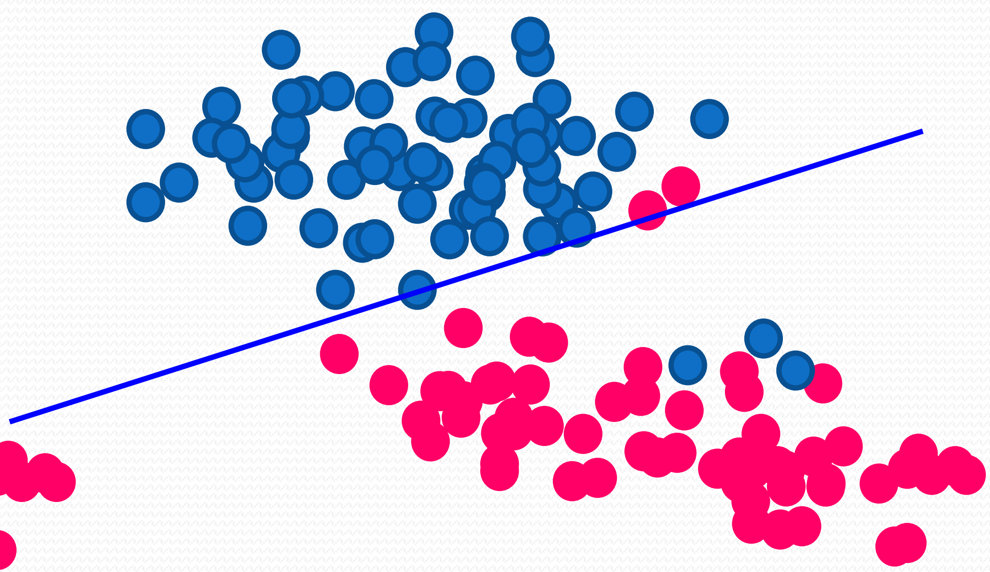
- **The initial values for w obtained by the linear regression method can be good starting point for classification which minimizes the computation time (compared to the case we start from random values for w).**

Linear Regression For Classification

Start with linear regression



Continue with classification



Support Vector Machine

Maximum margin classification with support vector machines

The objective in SVM-algorithm is to maximize the margin.

Margin: The distance between the separating hyperplane (decision boundary) and the training samples that are closest to this hyperplane, which are the so-called **support vectors**.

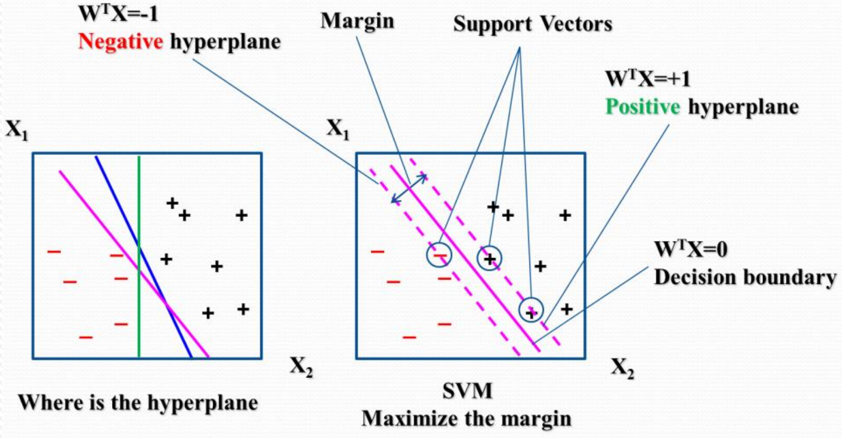
Models with decision boundaries with **large margins** tend to have a **lower generalization error** whereas models with small margins are **more prone to overfitting**.

Example:

$w_0 + w^T x_{pos} = 1$
$w_0 + w^T x_{neg} = -1$
$w^T (x_{pos} - x_{neg}) = 2$

$$\frac{w^T (x_{pos} - x_{neg})}{\|w\|} = \frac{2}{\|w\|}$$

$$\|w\| = \sqrt{\sum_{j=1}^{m} w_j^{\,2}}$$



Support Vector Machine

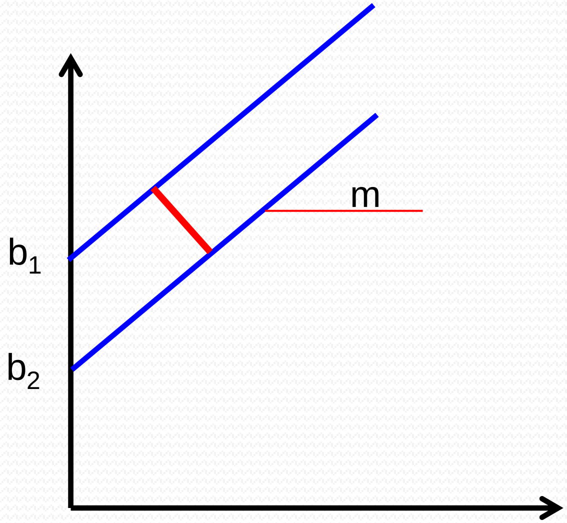
$\frac{w^T (x_{pos} - x_{neg})}{\|w\|}$ margin: distance between positive and negative hyperplanes

The objective of SVM algorithm is maximization of this margin by maximizing of $\frac{2}{\|w\|}$ under the constraint that the samples are classified correctly:

$w_0 + w^T x^i \geq 1$ if $y^i = 1$ (1)

$w_0 + w^T x^i \leq -1$ if $y^i = -1$ (2)

$y^i (w_0 + w^T x^i) \geq 1 \quad \forall i$ (3) compact form



Maximizing $\frac{2}{\|w\|}$ → Minimizing $\frac{1}{2}\|w\|$

So $\frac{1}{2}\|w\|$ is minimized with the condition of $y^i (w_0 + w^T x^i) \geq 1 \quad \forall i$ using quadratic Programming.

$$d = \frac{|b_1 - b_2|}{\sqrt{1+m^2}}$$

Soft-margin Classification

Dealing with the Nonlinearly Separable Case Using Slack Variables
Introduced by Vladimir Vapnik in 1995

By introducing the positive slack variable, the linear constraints are relaxed for nonlinearly separable data to allow convergence of the optimization in the presence of misclassifications under the appropriate cost penalization.

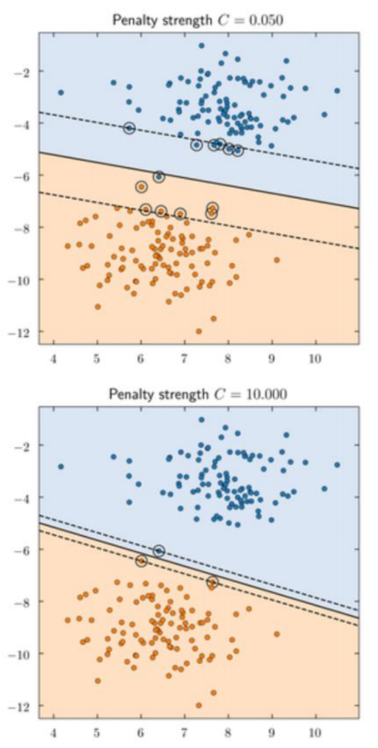
$w^T x^i \geq 1 \quad \text{if } y^i = 1 - \xi^i$
$w^T x^i \leq -1 \quad \text{if } y^i = 1 + \xi^i$

The new objective to be minimized is:

$$\frac{1}{2} ||w|| + \mathbf{C} \sum_i \xi^i$$

C(penalty strength): Controls the penalty for misclassification.

Increasing the value of C increases the bias and lowers the variance of the model. If the penalty is small the number of training points that define the separating hyperplane is large.



Nested Cross-Validation

Cross-Validation (CV):

1- **Train** a model on a subset of data, and **validate** the trained model on the remaining data (validation data).

2- Repeat this for all splits and average the validation error. This gives an estimate of the generalization performance of the model. Chose the parameter leading to the best score.

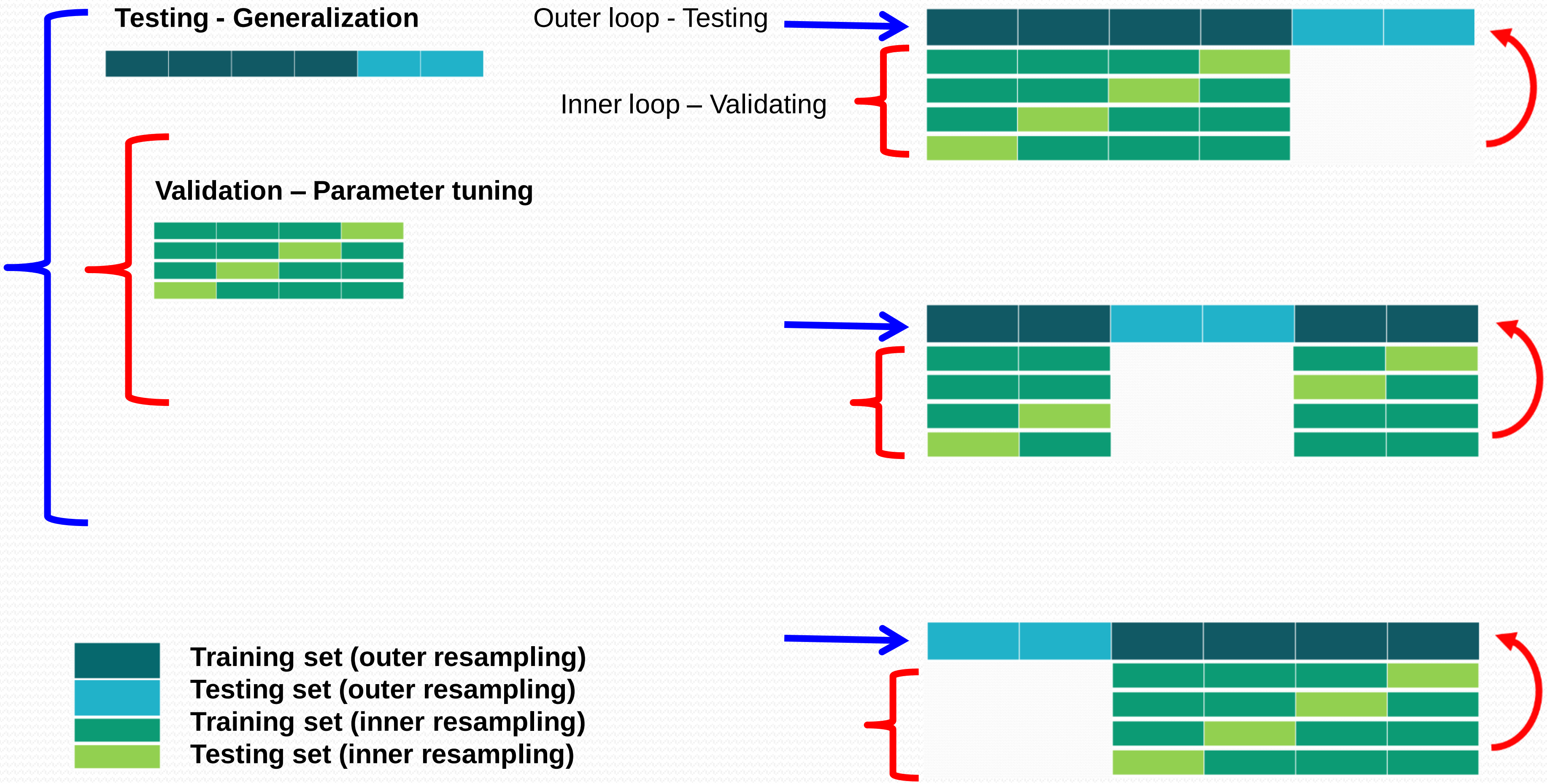
Model selection in CV uses the same data to tune model parameters and evaluate model performance. Information may thus "leak" into the model and overfit the data.

Choosing the parameters that maximize the scores in non-nested CV biases the model to the dataset, yielding an overly-optimistic score.

20%	20%	20%	20%	20%
Training data	Training data	Training data	Training data	Validation
Training data	Training data	Training data	Validation	Training data
Training data	Training data	Validation	Training data	Training data
Training data	Validation	Training data	Training data	Training data
Validation	Training data	Training data	Training data	Training data

5-fold validation

Nested Cross-Validation



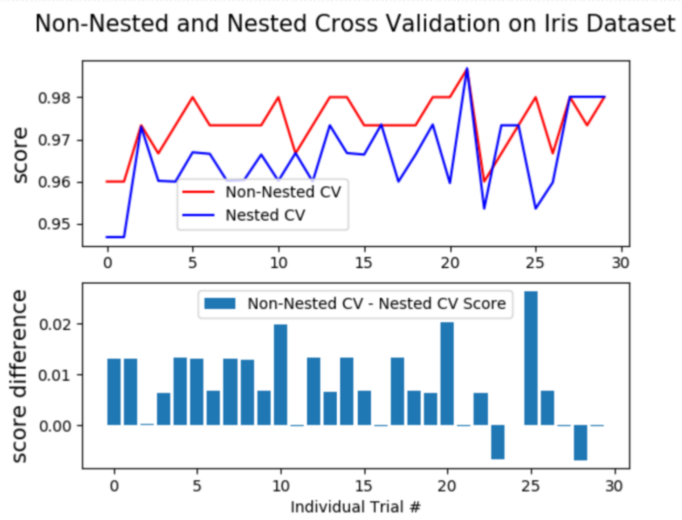
Nested Cross-Validation

Nested Cross-Validation:

1- The **hyper-parameter tuning** is carried out in the inner loop.

2- In the outer loop the **generalization** error of the underlying layer is estimated. The inner loop is responsible for model selection / hyper-parameter tuning., while the outer loop is for error estimation (test set).

In **sklearn** the hyper-parameters are turned by **GridSearchCV** in the inner loop. In the outer loop generalization error is estimated by averaging test set scores (using **cross_val_score**) over several dataset splits



Non-nested and nested cross-validation strategies on a classifier of the iris data set.
https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

K-Nearest Neighbours (KNN) Algorithm

K-Nearest Neighbours – A Lazy Learning Algorithm

It is called lazy because it doesn't learn a discriminative function from the training data but memorizes the training dataset instead.

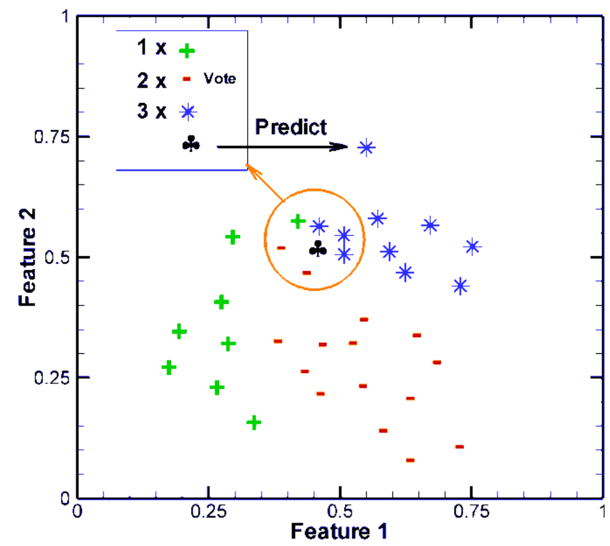
Steps:

1. Choose the number of k and a distance metric.
2. Find the k nearest neighbours of the sample to be classified.
3. Assign the class label by majority vote.

d

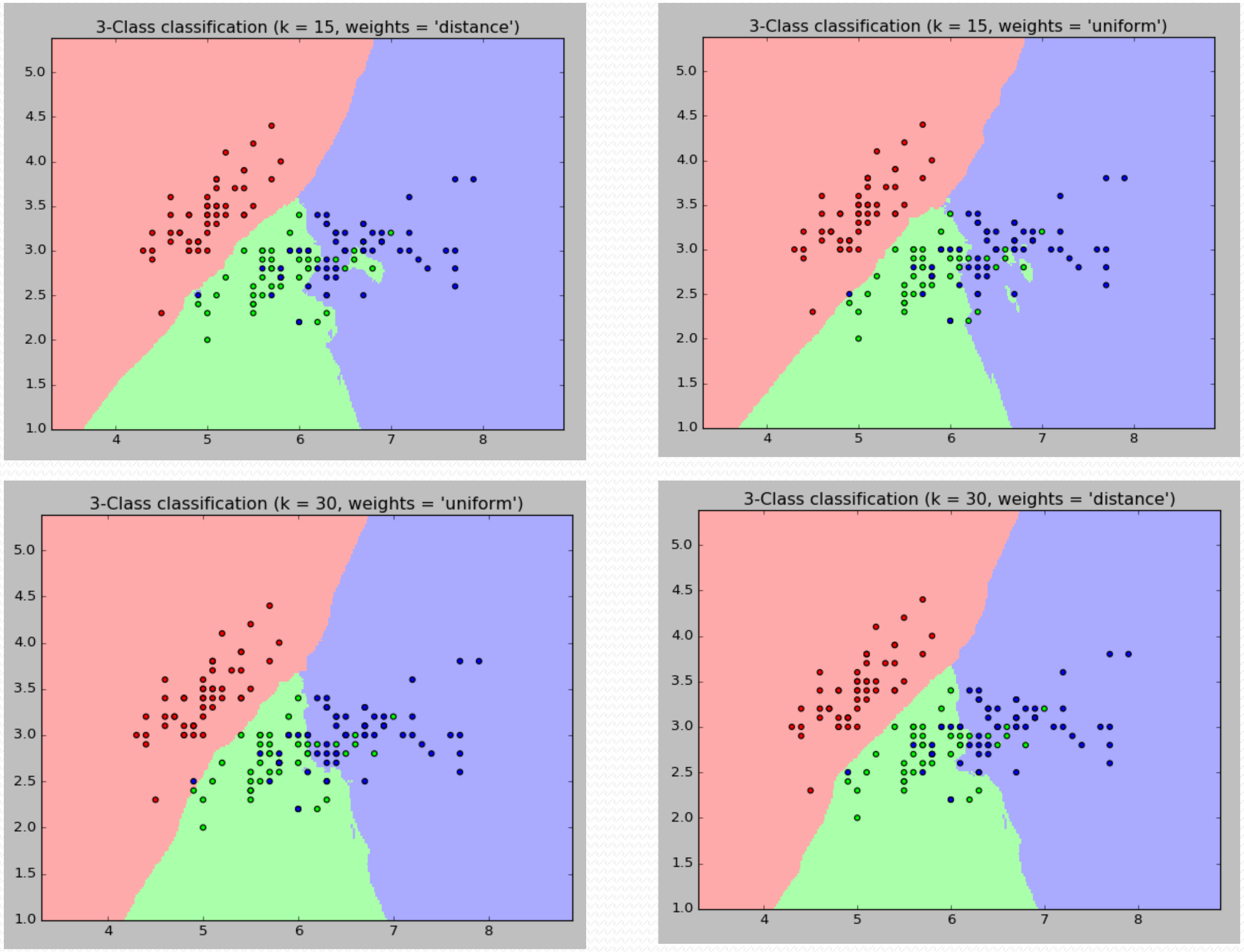
The right choice of k is crucial to find a good balance between over- and underfitting.

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$ Minkowski distance



Metrics intended for real-valued vector spaces:			
identifier	class name	args	distance function
"euclidean"	EuclideanDistance		sqrt(sum((x - y)^2))
"manhattan"	ManhattanDistance		sum(|x - y|)
"minkowski"	MinkowskiDistance	p	sum(|x - y|^p)^(1/p)

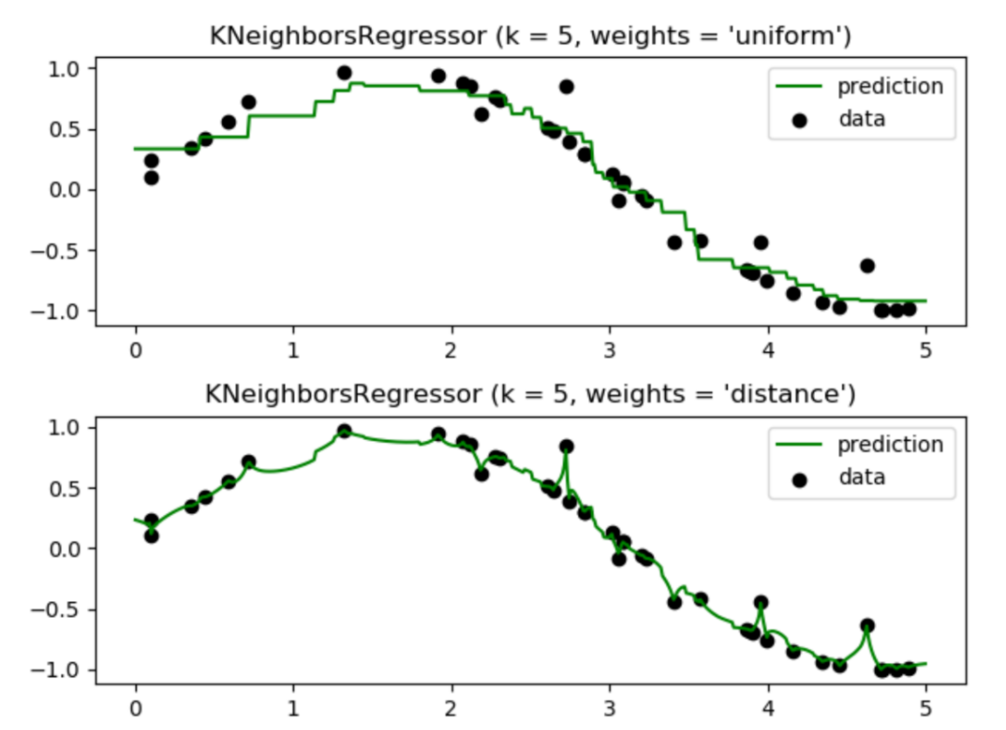
K-Nearest Neighbours (KNN) Algorithm
Iris Example



https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

K-Nearest Neighbours (KNN) Algorithm
Iris Example



Sklearn:

clf = neighbors.KNeighborsClassifier(n_neighbors, weights=distance, p=2, metric='minkowski')

clf.fit(X, y)