UNIVERSIDAD DE CHILE

Minería de Datos

Welcome to the Machine Learning class

Valentin Barriere

Universidad de Chile – DCC

CC5205, Fall 2025

Supervised Learning – Linear Models

Reminders

Outline : Reminders

Outline : Summary

In Summary

I **Have labeled data**

II **Extract the features** = transform documents into vectors

III **Create a mathematical model** f_θ

IV **Implement a cost (error) function** ℓ **to minimize**

V **Find the parameters** θ^* **such that** $\ell(f_{\theta^*}(\mathbf{X}_i), Y_i)$ **is small**

VI **Test** f_{θ^*} **on new data using an appropriate evaluation metric**

In Summary

I **Have labeled data**
- Dataset of size n, $\mathcal{D}_n = \{(\text{Doc}_i, Y_i), i = 1..n\}$
- Doc is a sample (for example: a person)
- Y are the labels (for example: the granted loan amount)

II **Extract the features** = transform documents into vectors

III **Create a mathematical model** f_θ

IV **Implement a cost (error) function** ℓ **to minimize**

V **Find the parameters** θ^* **such that** $\ell(f_{\theta^*}(\mathbf{X}_i), Y_i)$ **is small**

VI **Test** f_{θ^*} **on new data using an appropriate evaluation metric**

In Summary

I **Have labeled data**
- Dataset of size n, $\mathcal{D}_n = \{(\text{Doc}_i, Y_i), i = 1..n\}$
- Doc is a sample (for example: a person)
- Y are the labels (for example: the granted loan amount)

II **Extract the features** = transform documents into vectors
- $\mathbf{X}$ is a vector of observations (for example: age, gender, salary)
- Y are the labels (for example: the granted loan amount)

III **Create a mathematical model** f_θ

IV **Implement a cost (error) function** ℓ **to minimize**

V **Find the parameters** θ^* **such that** $\ell(f_{\theta^*}(\mathbf{X}_i), Y_i)$ **is small**

VI **Test** f_{θ^*} **on new data using an appropriate evaluation metric**

In Summary

I **Have labeled data**
- Dataset of size n, $\mathcal{D}_n = \{(\text{Doc}_i, Y_i), i = 1..n\}$
- Doc is a sample (for example: a person)
- Y are the labels (for example: the granted loan amount)

II **Extract the features** = transform documents into vectors
- $\mathbf{X}$ is a vector of observations (for example: age, gender, salary)
- Y are the labels (for example: the granted loan amount)

III **Create a mathematical model** f_θ
- A model f_θ such that $f_\theta(\mathbf{X})$ is close to Y (for regression)
- θ is the set of parameters of the mathematical model

IV **Implement a cost (error) function** ℓ **to minimize**

V **Find the parameters** θ^* **such that** $\ell(f_{\theta^*}(\mathbf{X}_i), Y_i)$ **is small**

VI **Test** f_{θ^*} **on new data using an appropriate evaluation metric**

In Summary

I **Have labeled data**
- Dataset of size n, $\mathcal{D}_n = \{(\text{Doc}_i, Y_i), i = 1..n\}$
- Doc is a sample (for example: a person)
- Y are the labels (for example: the granted loan amount)

II **Extract the features** = transform documents into vectors
- $\mathbf{X}$ is a vector of observations (for example: age, gender, salary)
- Y are the labels (for example: the granted loan amount)

III **Create a mathematical model** f_θ
- A model f_θ such that $f_\theta(\mathbf{X})$ is close to Y (for regression)
- θ is the set of parameters of the mathematical model

IV **Implement a cost (error) function** ℓ **to minimize**
- The more the model is wrong, the higher the cost
- In general, a small cost is desired

V **Find the parameters** θ^* **such that** $\ell(f_{\theta^*}(\mathbf{X}_i), Y_i)$ **is small**

VI **Test** f_{θ^*} **on new data using an appropriate evaluation metric**

In Summary

I **Have labeled data**
- Dataset of size n, $\mathcal{D}_n = \{(\text{Doc}_i, Y_i), i = 1..n\}$
- Doc is a sample (for example: a person)
- Y are the labels (for example: the granted loan amount)

II **Extract the features** = transform documents into vectors
- $\mathbf{X}$ is a vector of observations (for example: age, gender, salary)
- Y are the labels (for example: the granted loan amount)

III **Create a mathematical model** f_θ
- A model f_θ such that $f_\theta(\mathbf{X})$ is close to Y (for regression)
- θ is the set of parameters of the mathematical model

IV **Implement a cost (error) function** ℓ **to minimize**
- The more the model is wrong, the higher the cost
- In general, a small cost is desired

V **Find the parameters** θ^* **such that** $\ell(f_{\theta^*}(\mathbf{X}_i), Y_i)$ **is small**
- $\theta^* = \arg\min_\theta \sum_i \ell(f_\theta(\mathbf{X}_i), Y_i)$

VI **Test** f_{θ^*} **on new data using an appropriate evaluation metric**

In Summary

I **Have labeled data**
- Dataset of size n, $\mathcal{D}_n = \{(\text{Doc}_i, Y_i), i = 1..n\}$
- Doc is a sample (for example: a person)
- Y are the labels (for example: the granted loan amount)

II **Extract the features** = transform documents into vectors
- $\mathbf{X}$ is a vector of observations (for example: age, gender, salary)
- Y are the labels (for example: the granted loan amount)

III **Create a mathematical model** f_θ
- A model f_θ such that $f_\theta(\mathbf{X})$ is close to Y (for regression)
- θ is the set of parameters of the mathematical model

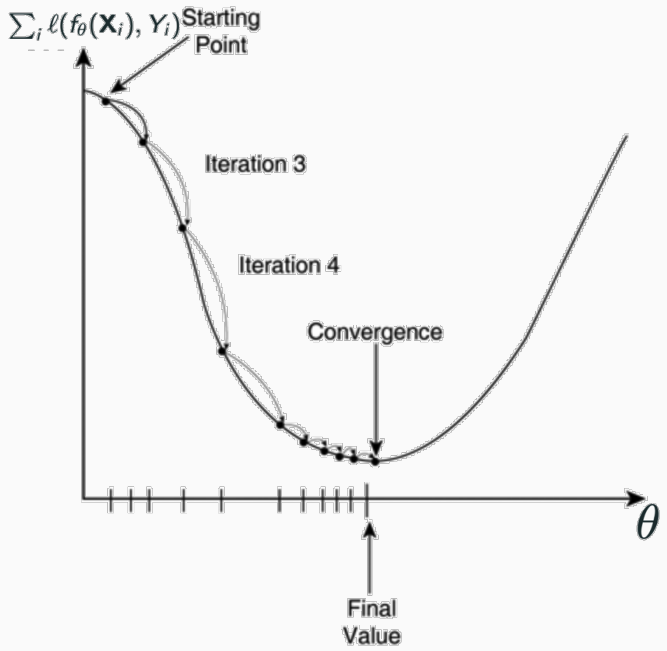
IV **Implement a cost (error) function** ℓ **to minimize**
- The more the model is wrong, the higher the cost
- In general, a small cost is desired

V **Find the parameters** θ^* **such that** $\ell(f_{\theta^*}(\mathbf{X}_i), Y_i)$ **is small**
- $\theta^* = \arg\min_\theta \sum_i \ell(f_\theta(\mathbf{X}_i), Y_i)$

VI **Test** f_{θ^*} **on new data using an appropriate evaluation metric**

Outline : Optimization

Optimization



Optimization of the Cost Function

- It is used to converge to the minimum value of the cost function on the training dataset
- Best case: fast and accurate
- Often approximations are made to speed up the process

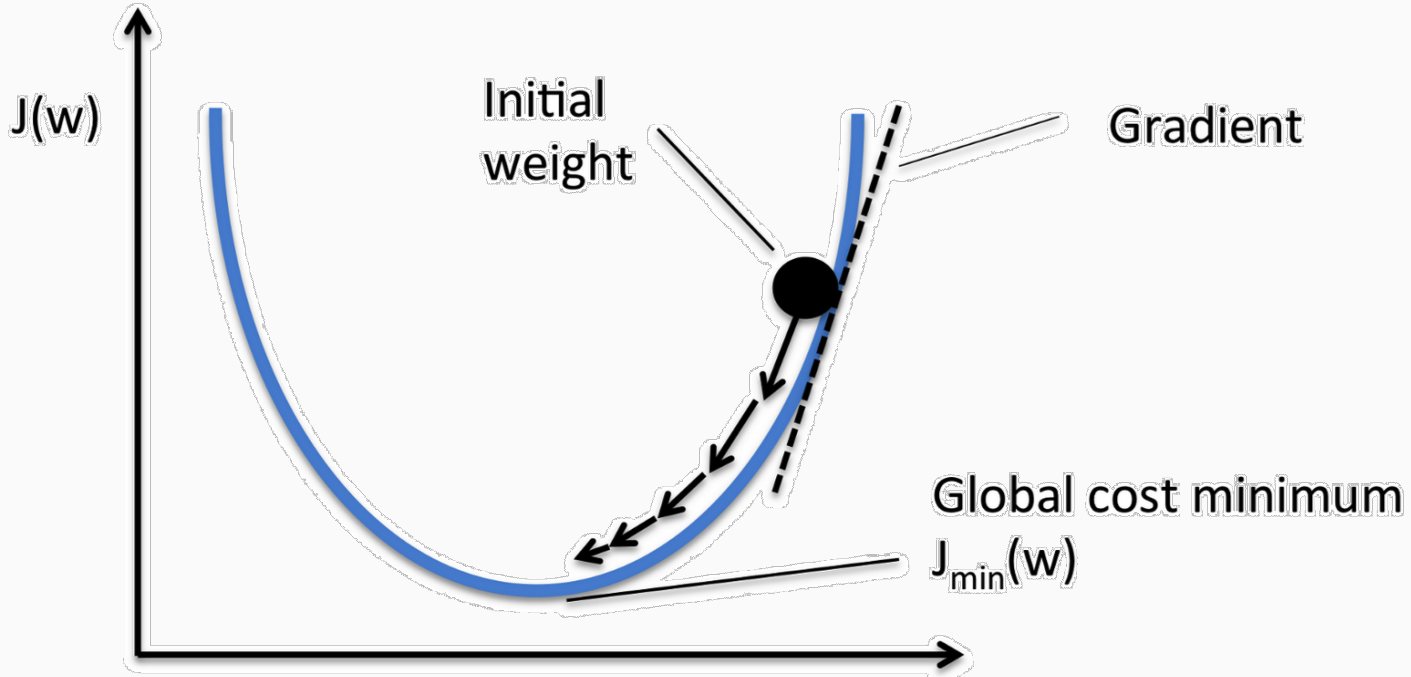
Gradient

A multivariate function $f(x)$ can be written as a Taylor series:

$$f(x + \delta_x) = f(x) + \nabla_x f(x)\delta_x + \mathcal{O}(||\delta_x^2||)$$

Definition

The gradient of a function $\nabla_x f(x) = (\frac{\partial f}{\partial x_i})_{i=1..n}$ is its derivative with respect to each dimension. It is a linear approximation of the function locally.



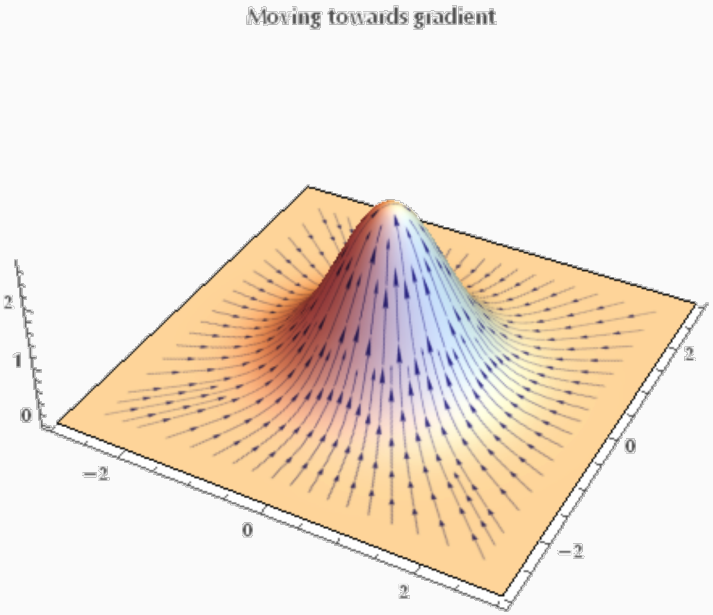
Gradient

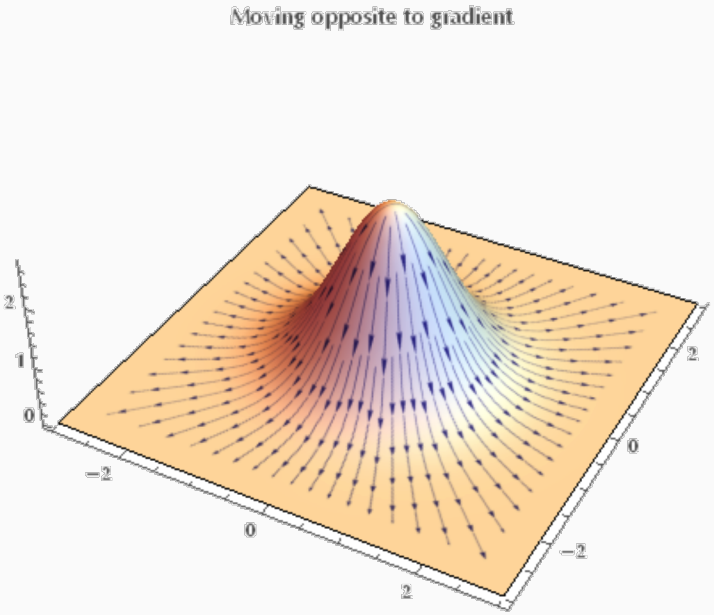
A multivariate function $f(x)$ can be written as a Taylor series:

$$f(x + \delta_x) = f(x) + \nabla_x f(x)\delta_x + \mathcal{O}(||\delta_x^2||)$$

Definition

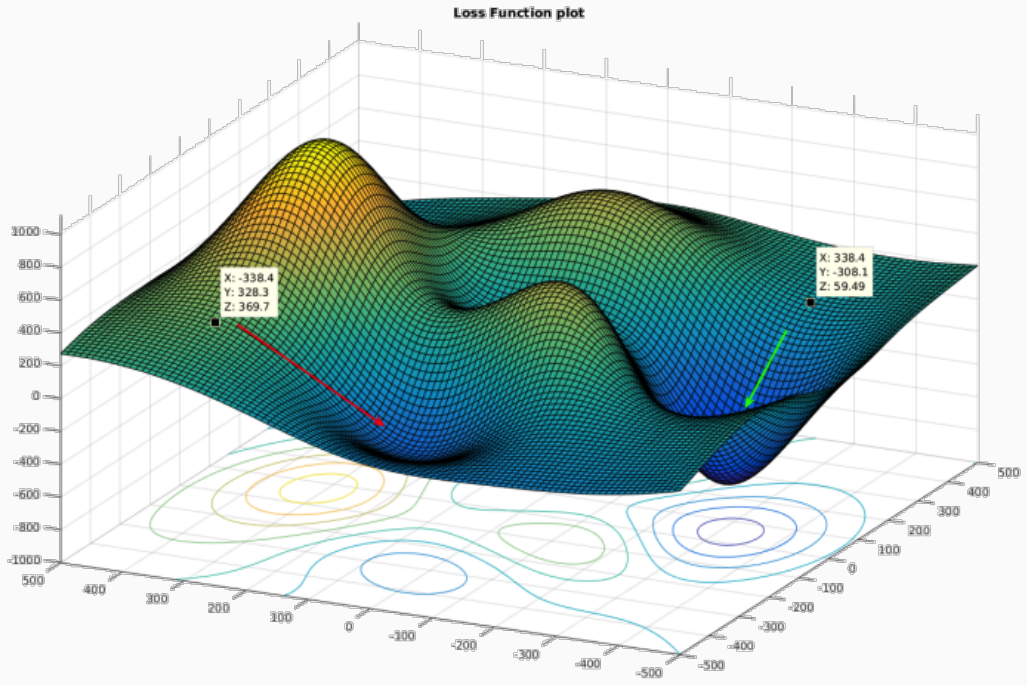
The gradient of a function $\nabla_x f(x) = (\frac{\partial f}{\partial x_i})_{i=1..n}$ is its derivative with respect to each dimension. It is a linear approximation of the function locally.





More details at here.

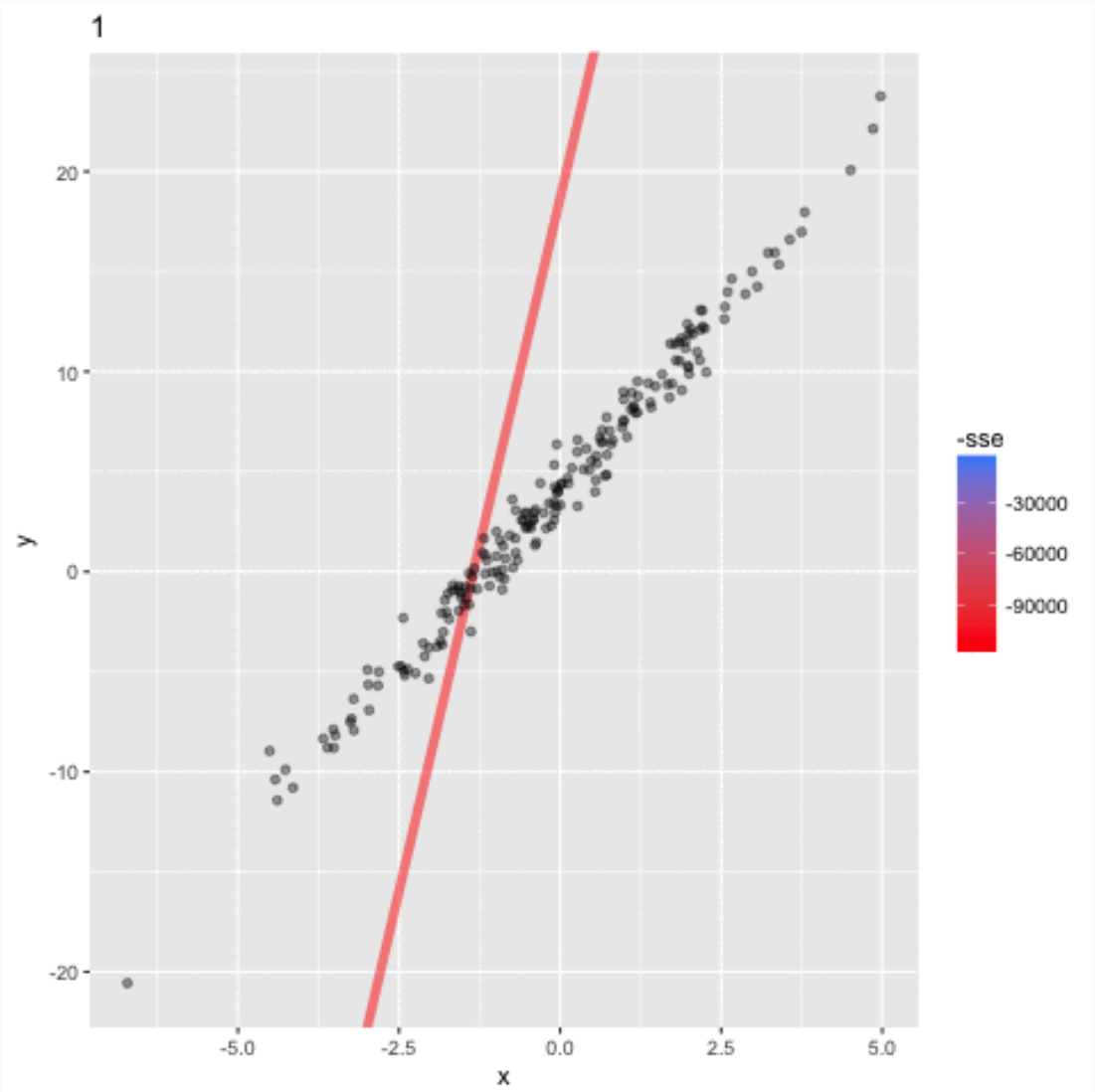
Visualization of the Cost Function



The cost function can be visualized as a surface:

- The parameter values θ vary on the plane, and the value of the function $\ell(\mathcal{D}_n; \theta)$ varies in height.
- Convergence occurs when the parameters are located in a dip in this surface (local or global minimum, depending on the model)

Optimization: Visualization



Optimization: Stochastic Gradient Descent

Gradient Descent

After each evaluation of the cost function $\ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$, the gradient of this function is computed to update the parameters θ:

$$\theta \leftarrow \theta - \alpha * \nabla_\theta \ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$$

Example

- Let $f_\theta(\mathbf{X}) = \theta^T \mathbf{X} = \sum_k^d \theta_k X^{(k)}$ and $\ell(Y, f_\theta(\mathbf{X})) = \frac{1}{2}(Y - f_\theta(\mathbf{X}))^2$

$$\nabla_\theta \ell(Y_i, f_\theta(\mathbf{X}_i); \theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_d} \end{pmatrix} \ell(Y_i, f_\theta(\mathbf{X}_i); \theta) = \begin{pmatrix} X_i^{(1)}(Y - f_\theta(\mathbf{X}_i)) \\ \vdots \\ X_i^{(d)}(Y_i - f_\theta(\mathbf{X}_i)) \end{pmatrix}$$

Optimization: Stochastic Gradient Descent

Gradient Descent

After each evaluation of the cost function $\ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$, the gradient of this function is computed to update the parameters θ:

$$\theta \leftarrow \theta - \alpha * \nabla_\theta \ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$$

Example

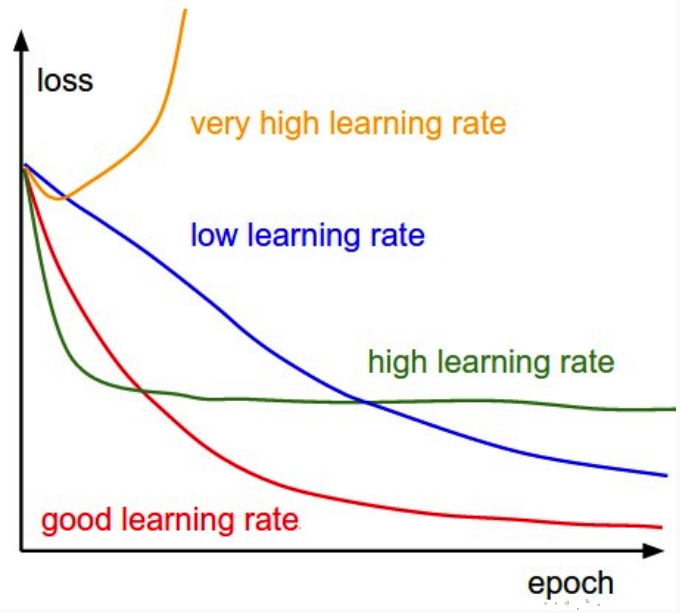
- Let $f_\theta(\mathbf{X}) = \theta^T \mathbf{X} = \sum_k^d \theta_k X^{(k)}$ and $\ell(Y, f_\theta(\mathbf{X})) = \frac{1}{2}(Y - f_\theta(\mathbf{X}))^2$

$$\begin{pmatrix} \theta_1 \\ \vdots \\ \theta_d \end{pmatrix} \leftarrow \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_d \end{pmatrix} - \alpha * \begin{pmatrix} X_i^{(1)}(Y - f_\theta(\mathbf{X}_i)) \\ \vdots \\ X_i^{(d)}(Y_i - f_\theta(\mathbf{X}_i)) \end{pmatrix}$$

Optimization: Importance of the Learning Rate

Learning Rate

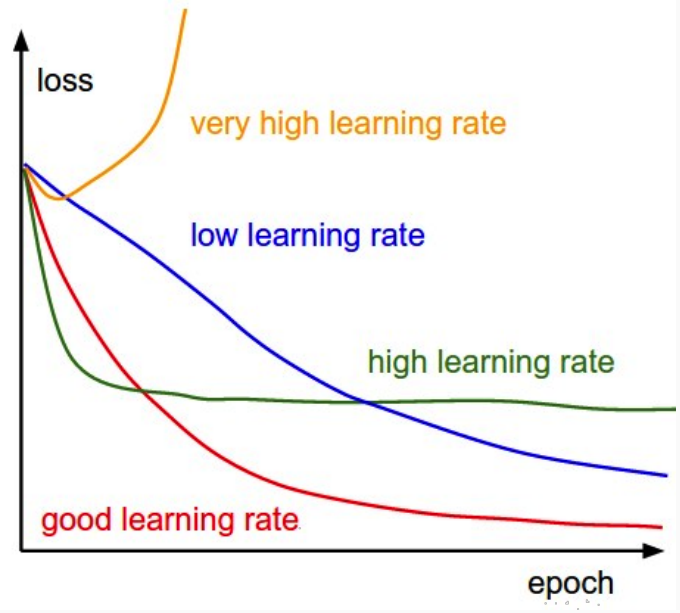
- A learning rate α that is too small does not advance learning
- A learning rate that is too small lengthens the training time
- A learning rate that is too large prevents reaching the minimum (updates circle around the dip in the surface)
- A learning rate that is too large prevents any progress
- One solution: decrease α over time

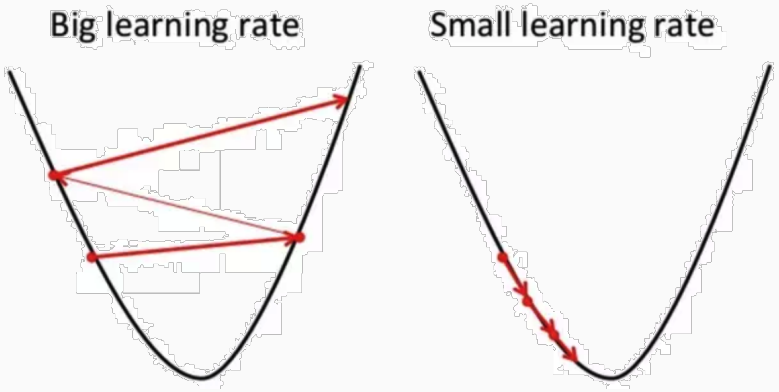


Optimization: Importance of the Learning Rate

Learning Rate

- A learning rate α that is too small does not advance learning
- A learning rate that is too small lengthens the training time
- A learning rate that is too large prevents reaching the minimum (updates circle around the dip in the surface)
- A learning rate that is too large prevents any progress
- One solution: decrease α over time





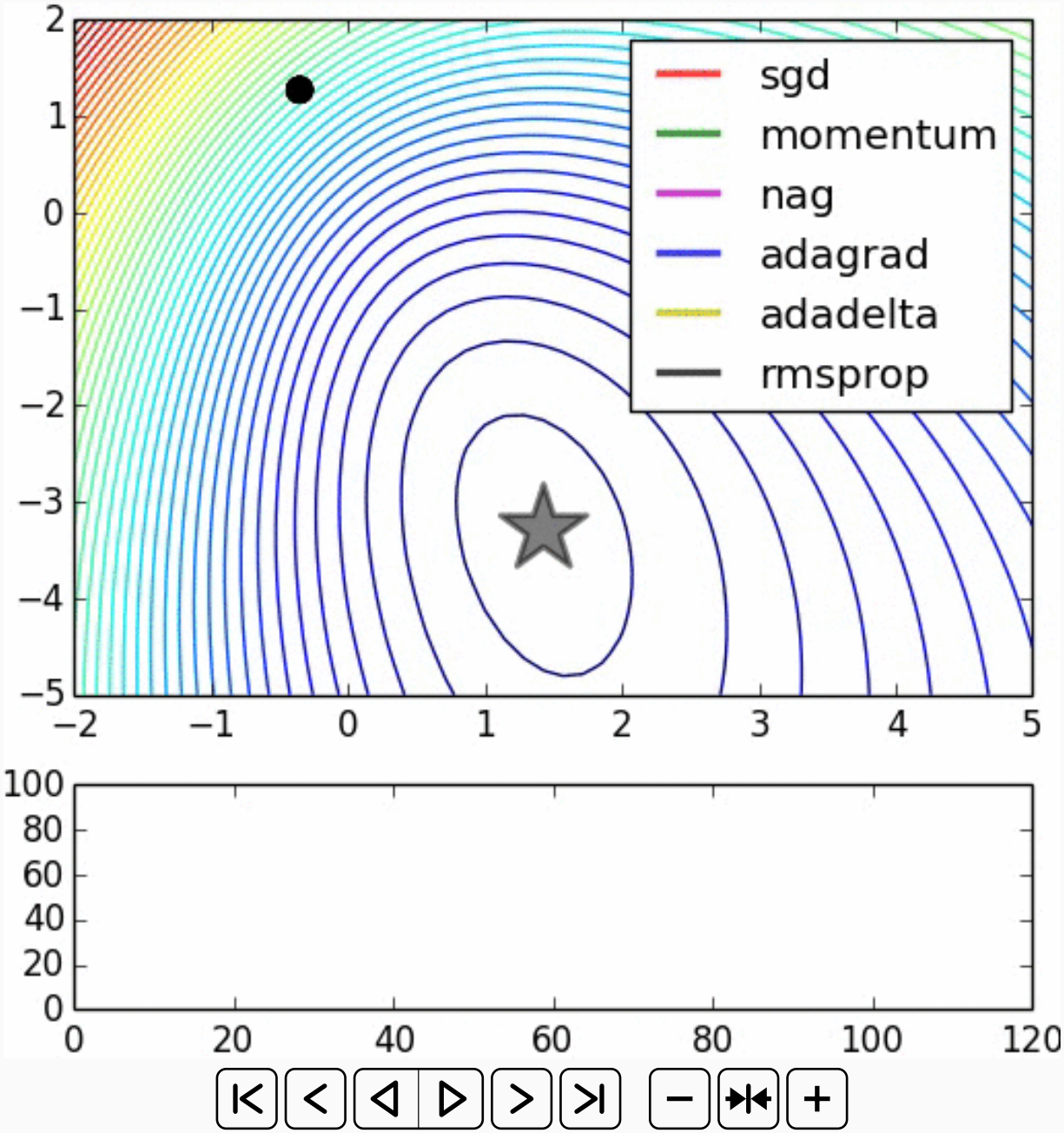
Gradient Descent Algorithms

There are other existing algorithms based on stochastic gradient descent (SGD) with specific modifications to improve efficiency:

- Stochastic gradient descent with momentum
- Nesterov Accelerated Gradient (NAG)
- Adaptive Gradient (AdaGrad)
- Adam
- RMSprop

Finally, there are also other more classical descent methods: BFGS, L-BFGS, Quasi-Newton, ...

Gradient Descent Algorithms

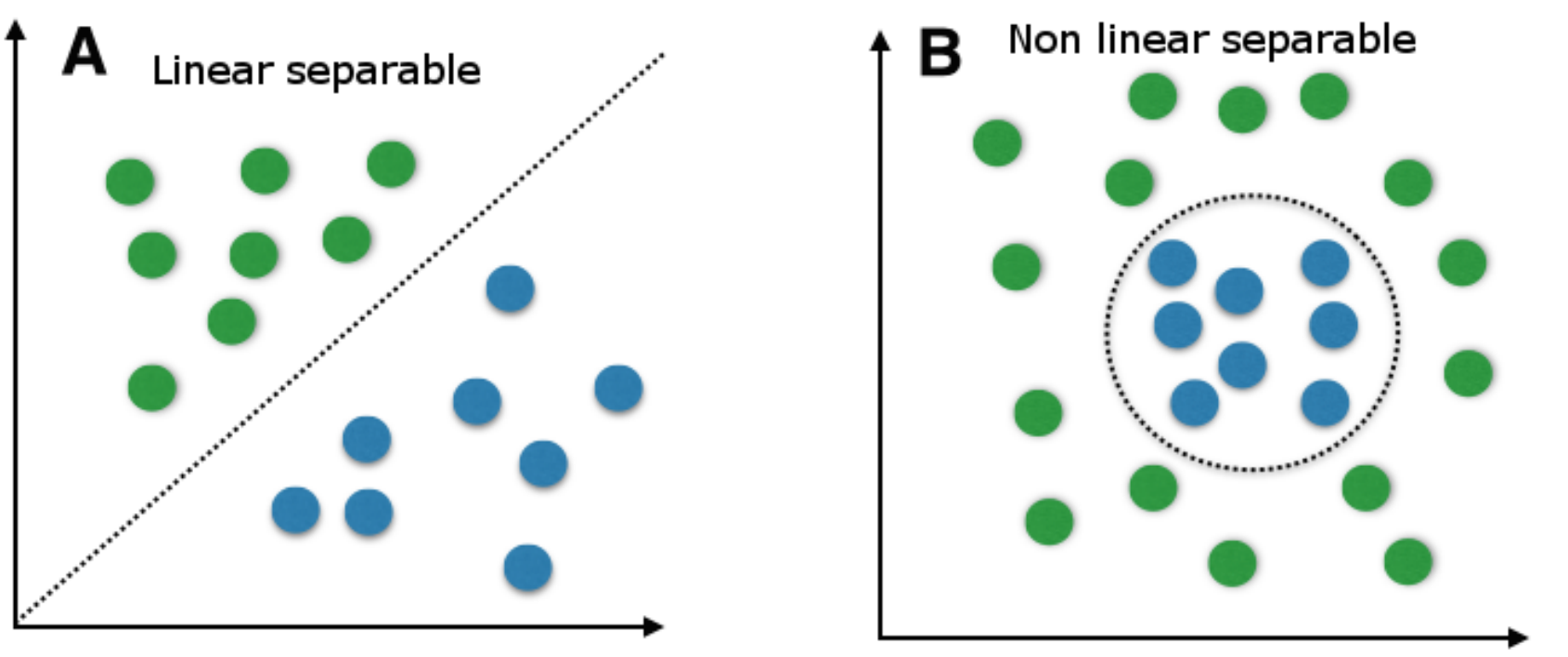


Classification Models

Outline : Classification Models

Outline : Linear Classifier

Linear and Nonlinear Problems



Linear Function

A function $f : \mathcal{X} \to \mathcal{Y}$ is linear if:

$$\forall(\mathbf{X}, \mathbf{X}') \in \mathcal{X} \text{ and } \lambda \in \mathbb{R}, \text{ then } f(\lambda\mathbf{X} + \mathbf{X}') = \lambda f(\mathbf{X}) + f(\mathbf{X}')$$

This is the case for functions of the form $f(\mathbf{X}) = \theta^T\mathbf{X} = \sum_i \theta_i X^{(i)}$

Geometry Reminders: Hyperplane I

Definition

In a d-dimensional space, such as $\mathbb{R}^d$, a hyperplane $\mathcal{H}$ is the set of points $\mathbf{X} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$ that satisfy an equation of the form:

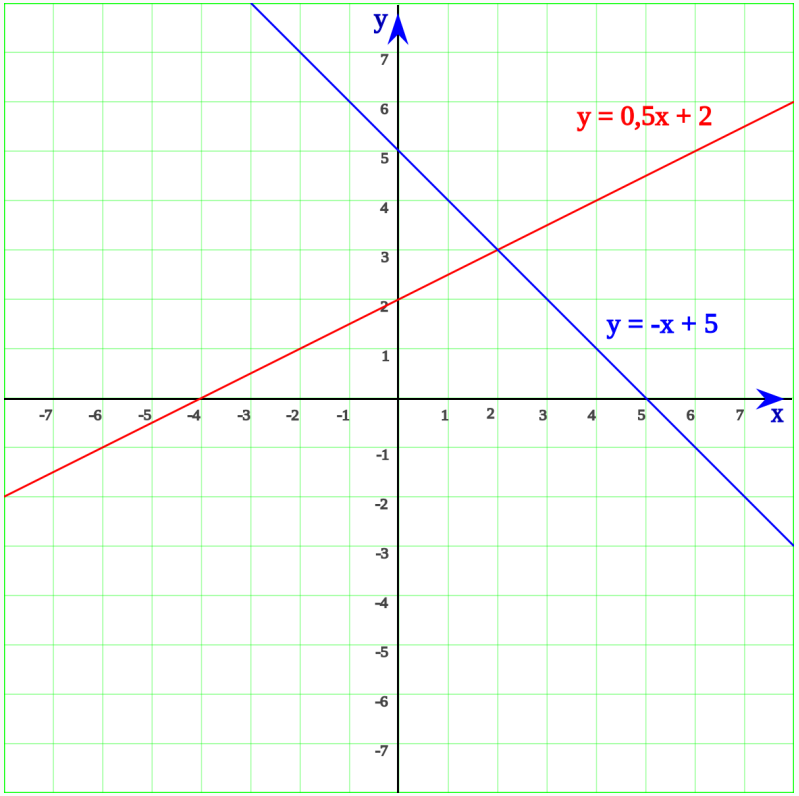
$$\mathcal{H} : w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_0 = 0, \quad \text{where } \forall i,\ w_i \in \mathbb{R}$$

A hyperplane divides the space into 2 parts (it is a $(d-1)$-dimensional space). In $\mathbb{R}^2$ (the plane) it is 1-dimensional (a line), in $\mathbb{R}^3$ it is 2-dimensional (a plane)...

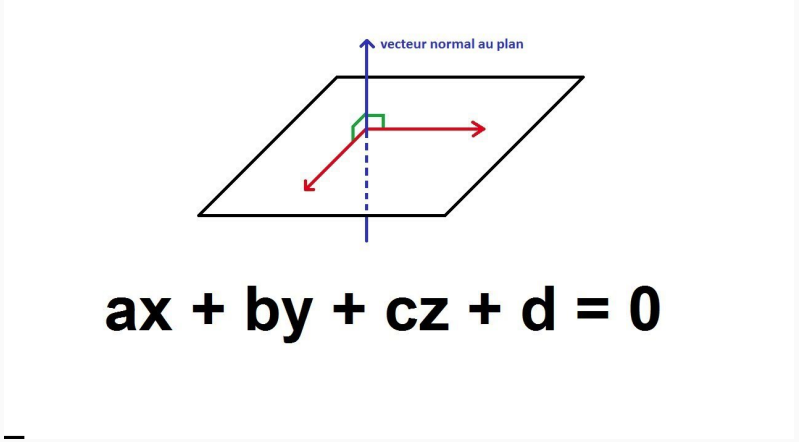
The sign of $w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_0$ indicates on which side of the hyperplane $\mathbf{X}$ is located: if > 0, then $\mathbf{X}$ is on one side, otherwise on the other.

Geometry Reminders: Hyperplane II

A hyperplane in 2D (i.e. a line) defined by a normal vector $\vec{n} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ and a bias w_0.



A hyperplane in 3D (i.e. a plane) defined by a normal vector $\vec{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ and a bias d.

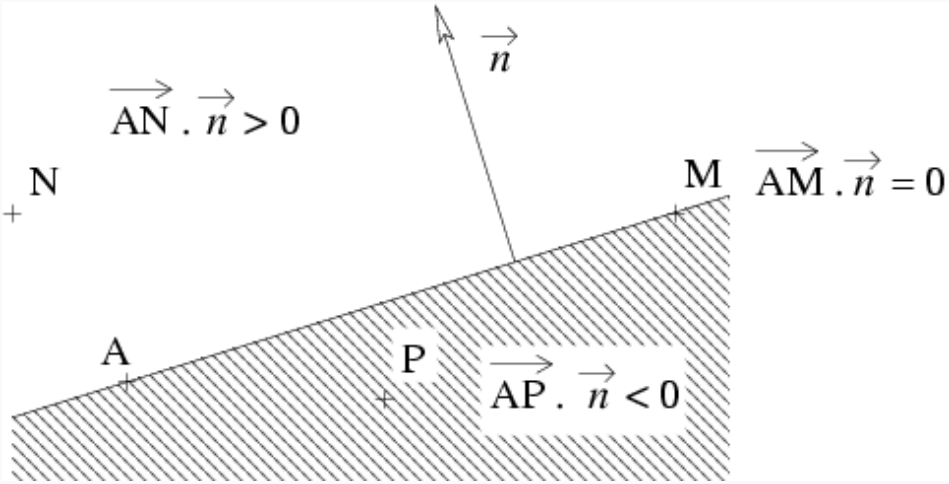


Geometry Reminders: Hyperplane III

Equation

For any point $\mathbf{X} \in \mathbb{R}^d$, the sign of $w_1 x_1 + \cdots + w_d x_d$ indicates on which side of the hyperplane the point $\mathbf{X}$ lies.

We have the set of points on one side of the plane and the other side based on the dot product with the normal vector $\vec{n}$.



Geometry Reminders: Hyperplane IV

With $\mathbf{X} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \\ 1 \end{pmatrix}$ and $\theta = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_d \\ \theta_0 \end{pmatrix}$, the previous equation can be written as:

$$\mathcal{H} : w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_0 = w_0 + \sum_{i=1}^{d} w_i x_i = \langle \theta, \mathbf{X} \rangle = 0$$

where $\langle \theta, \mathbf{X} \rangle$ is the dot product between θ and $\mathbf{X}$, a linear operation.

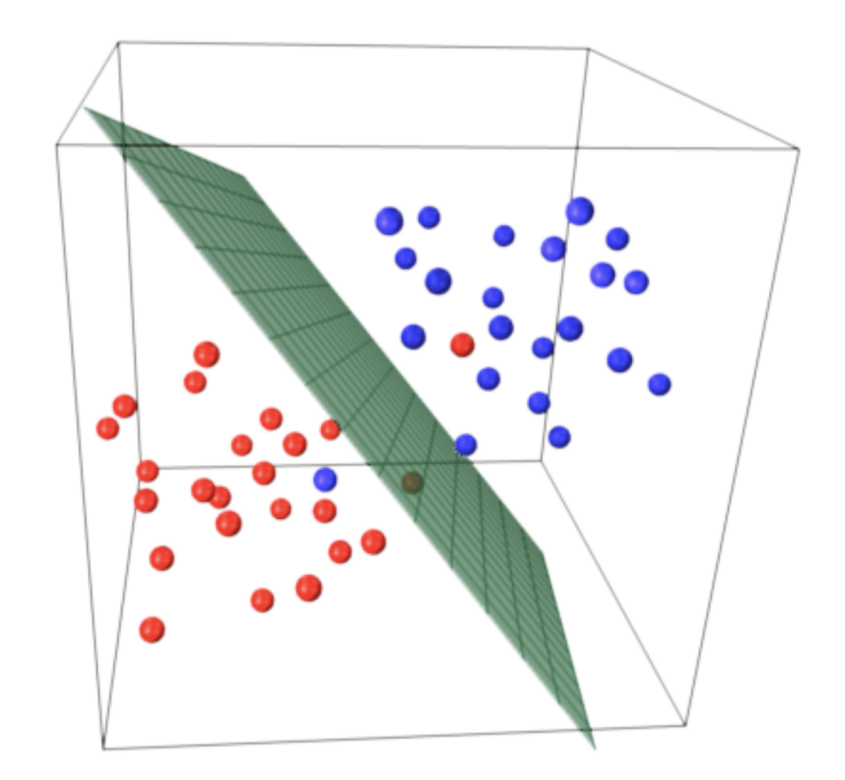
Lifting to Achieve Linearity

By increasing the dimensionality, an affine hyperplane can be represented as a linear hyperplane.

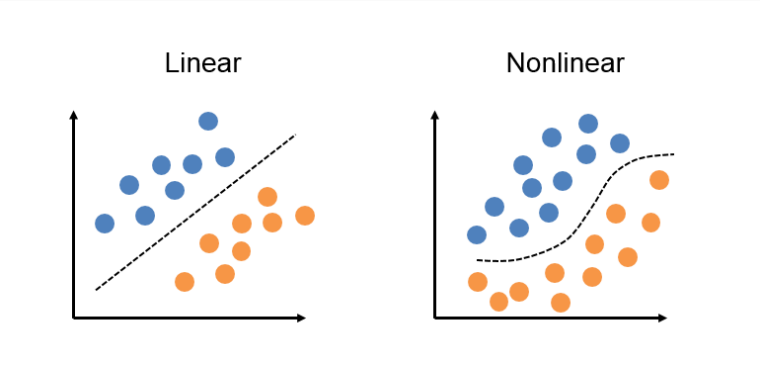
Linear Classifier: Summary

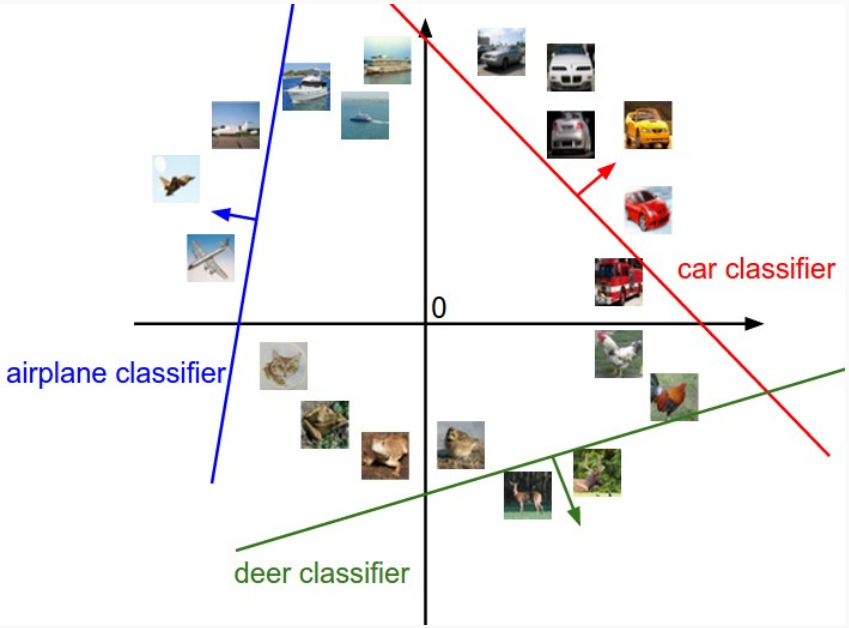
Summary

- $\mathbf{X} \in \mathbb{R}^d$ is the feature vector
- The equation $\mathbf{W}^T\mathbf{X} + b = 0$ defines a hyperplane in $\mathbb{R}^d$
- $f_{\mathbf{W},b}(\mathbf{X}) = \text{sign}(\mathbf{W}^T\mathbf{X} + b)$ gives the class of $\mathbf{X}$



Linear Classifier

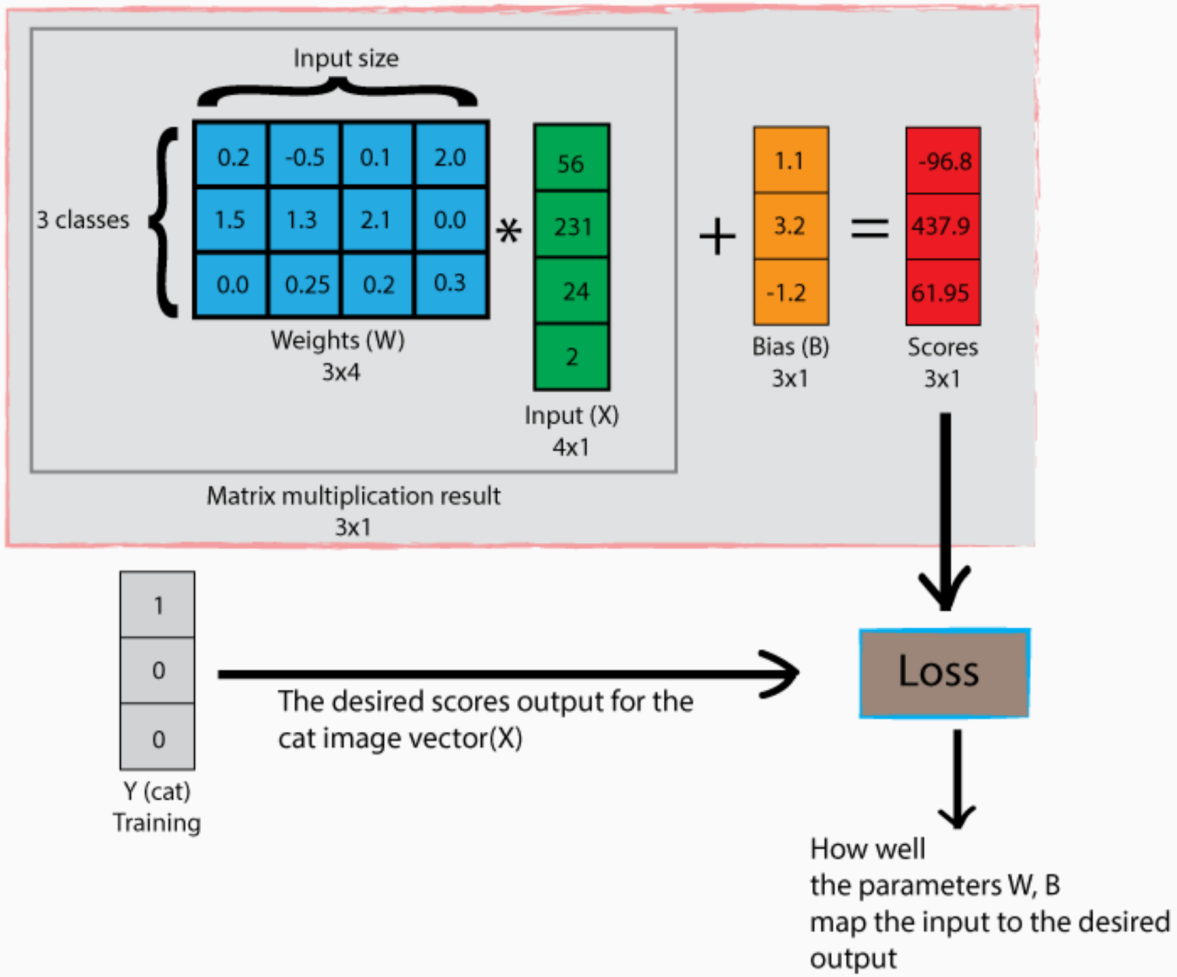




Linearity

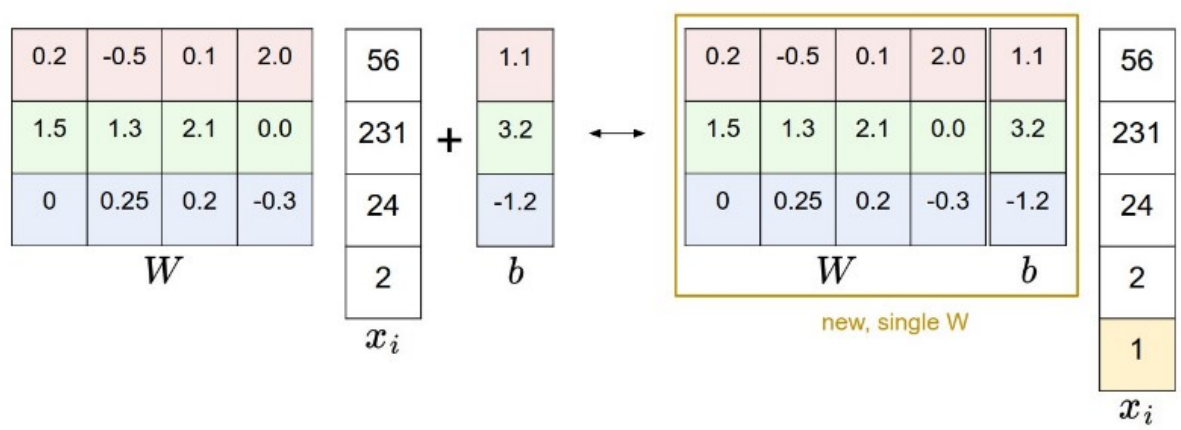
- It separates the space with a hyperplane
- Classifier of the form $f_{\mathbf{W},b}(\mathbf{X}) = \text{sign}(\mathbf{W}^T\mathbf{X} + b)$
- Two examples: binary (2 classes) and multinomial (n classes)
- Linear matrix operation

Multiclass Linear Classifier: Matrix Operation



We no longer have a simple parameter vector $\mathbf{W} = (w_1 \cdots w_D)$ as before, but a matrix $\mathbf{W} = (w_{cd})_{c=1..C,d=1..D}$.

Linear Classifier: Bias Integration



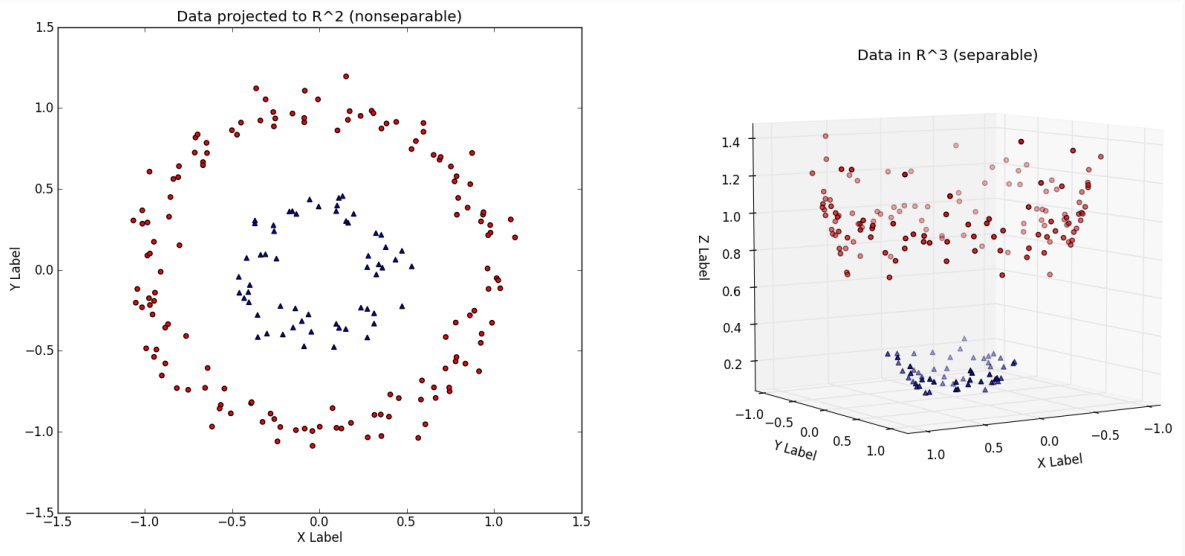
Linearity

By increasing the dimensionality and concatenating a vector of 1 to the features, we obtain a linear operation:

- It separates the space with a hyperplane
- Classifier of the form $f_{\mathbf{W},b}(\mathbf{X}) = \arg\max_c(\mathbf{W}^T\mathbf{X} + b)$
- If there are only 2 classes, the argmax can be replaced by a sign function

Linear Classifier: Feature Lifting

By increasing the dimensionality and adding new features, we can make our model nonlinear.

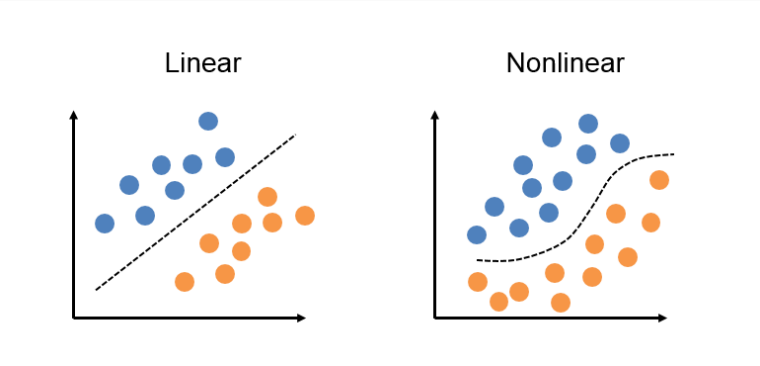


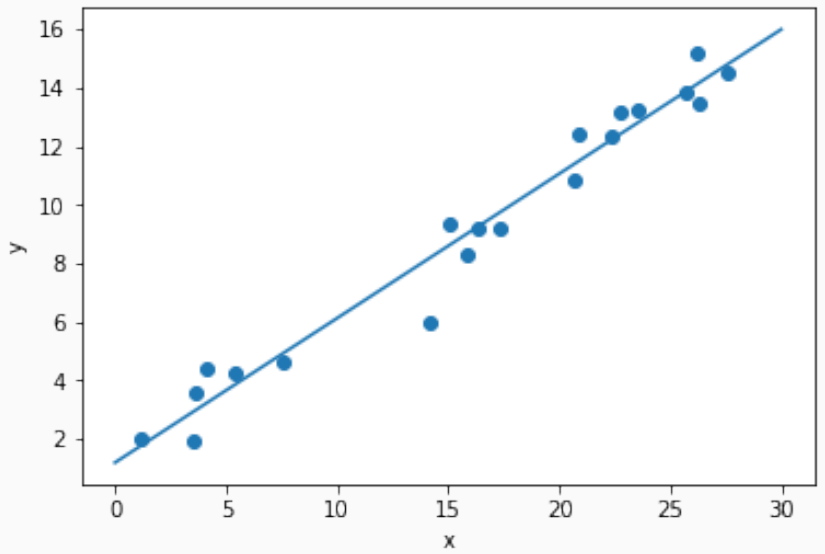
Nonlinearly Separable Data

- Data that is not linearly separable
- We have features of the form $\mathbf{X} = (x, y)$.
- By defining $z = x^2 + y^2$, we obtain features in a 3-dimensional space $\mathbf{X} = (x, y, x^2 + y^2)$, and thus we can linearly separate the data in 3D.

Outline : Linear Regression

Linear Regression





Linearity

- It is modeled with a hyperplane.
- The predictor is of the form $f_{\mathbf{W},b}(\mathbf{X}) = \mathbf{W}^T\mathbf{X} + b = \sum_{i=1}^{D} w_i X_i + b$
- Linear matrix operation.
- It can be adapted to nonlinear data by adding nonlinear features.

Simple Solution

To find the parameters that minimize the error, we compute the partial derivatives of the Sum of Squared Errors (SSE) with respect to β_0 and β_1, then set them equal to zero and solve for the parameters.

$$\frac{\partial R}{\partial \beta_0} = -2 \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_i) = 0$$

$$\frac{\partial R}{\partial \beta_1} = -2 \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_i) x_i = 0$$

Which yields:

$$\beta_1 = \frac{\sum_{i=1}^{N} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{N} (x_i - \overline{x})^2}$$

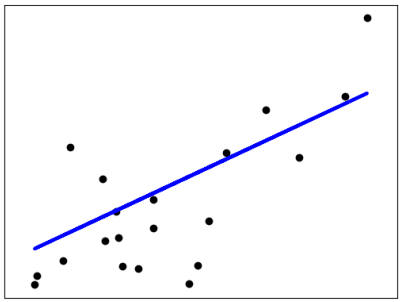
$$\beta_0 = \overline{y} - \beta_1 \overline{x}$$

Evaluation of Prediction: Distance

Different Functions to Evaluate Regression Performance

- Mean Absolute Error: $\frac{1}{n}\sum_{i=1}^{n}|Y_i - f(\mathbf{X}_i)|$
- Mean Squared Error: $\frac{1}{n}\sum_{i=1}^{n}|Y_i - f(\mathbf{X}_i)|^2$
- Median Absolute Error: $\text{median}(|Y_1 - f(\mathbf{X}_1)|, \ldots, |Y_n - f(\mathbf{X}_n)|)$
- Coefficient of Determination R^2:

$$1 - \frac{\sum_{i=1}^{n}|Y_i - f(\mathbf{X}_i)|^2}{\sum_{i=1}^{n}|Y_i - \overline{Y}|^2} = \frac{\sum_{i=1}^{n}|f(\mathbf{X}_i) - \overline{Y}|^2}{\sum_{i=1}^{n}|Y_i - \overline{Y}|^2}$$



R^2 represents the proportion of variance explained by the model

Linear Regression with 2 Features: A Plane

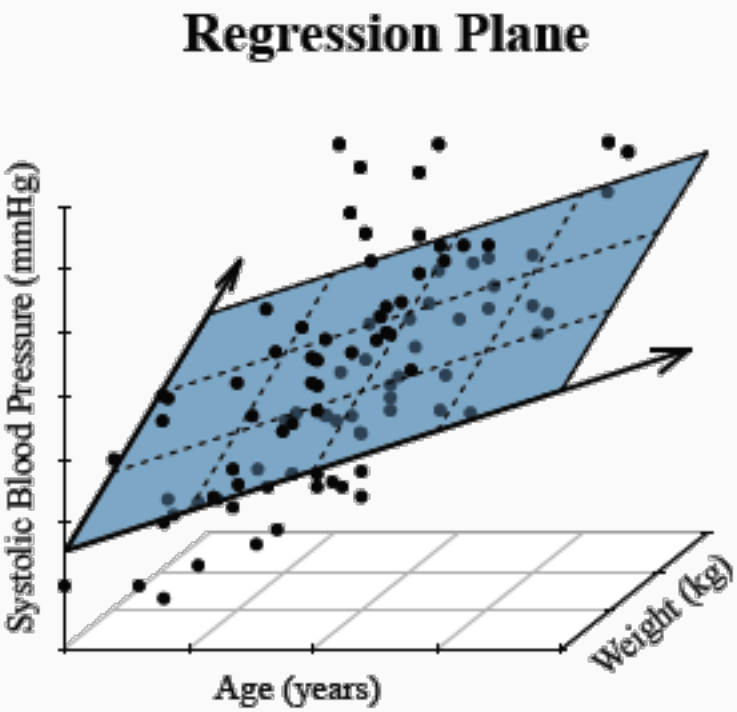


Figure 2.25: Systolic blood pressure linearly increases with age, but also with bodyweight. A line in two directions forms a plane.

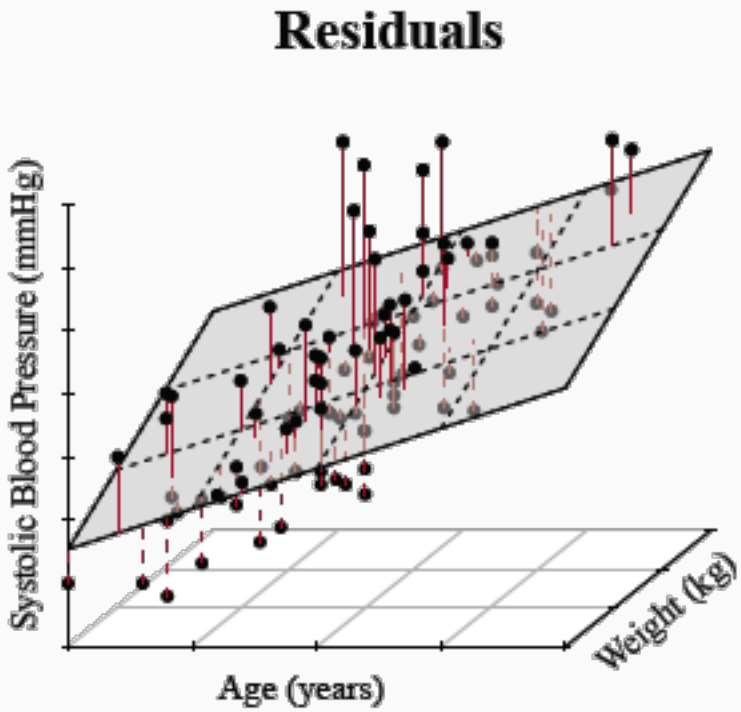


Figure 2.26: The residuals of figure 2.25 are the vertical distances to the plane. Negative residuals are indicated by dashed linepieces.

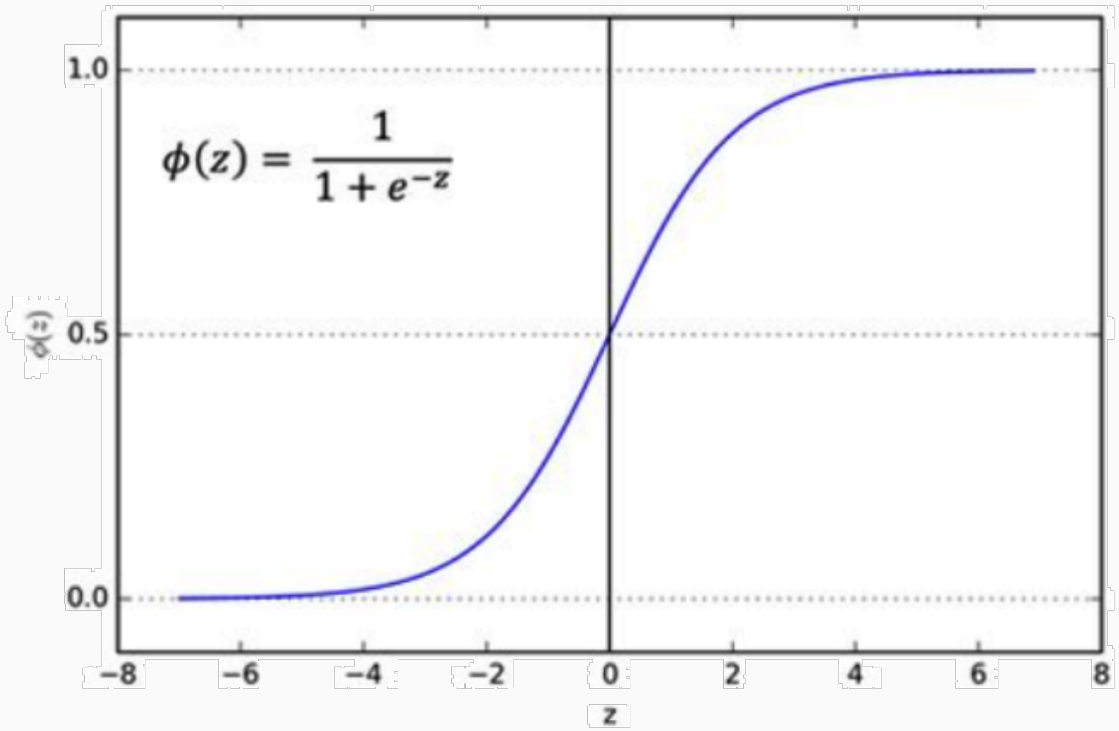
Figure 1: The 2 features + the target are real: 3 dimensions

Outline : Logistic Regression

Logistic Regression

To convert the "distances" $\mathbf{W}^T\mathbf{X} + b$ between the feature vector and the hyperplane into class probabilities, we use a normalization function, the sigmoidal or *softmax* function:

$$\Phi(\text{dist}) = \frac{1}{1 + e^{-\text{dist}}}$$



Logistic Regression

Binary Logistic Regression

- Training: Cost: $\ell(\mathbf{X}, Y) = \frac{1}{1+\exp^{Y\langle\theta,\mathbf{X}\rangle}}$, (here, ??)
- Classification:

$$P(Y=1) = \frac{1}{1+\exp^{-\langle\theta,\mathbf{X}\rangle}} = \frac{1}{1+\exp^{-\sum_{k=1}^{d}\theta_k X^{(k)}}}$$

- Class selection is based on the value of $\langle\theta, \mathbf{X}\rangle$:

$$P(Y=1) = \frac{1}{1+\exp^{-\langle\theta,\mathbf{X}\rangle}}$$

- For the other class:

$$P(Y=-1) = 1 - P(Y=1) = \frac{\exp^{-\langle\theta,\mathbf{X}\rangle}}{1+\exp^{-\langle\theta,\mathbf{X}\rangle}} = \frac{1}{1+\exp^{\langle\theta,\mathbf{X}\rangle}}$$

- Final class: $\hat{c} = \arg\max_c P(Y=c)$

Logistic Regression

Binary Logistic Regression

- Training: Cost: $\ell(\mathbf{X}, Y) = \frac{1}{1+\exp^{Y\langle\theta,\mathbf{X}\rangle}}$, (here, logistic)
- Classification:

$$P(Y=1) = \frac{1}{1+\exp^{-\langle\theta,\mathbf{X}\rangle}} = \frac{1}{1+\exp^{-\sum_{k=1}^{d}\theta_k X^{(k)}}}$$

- Class selection is based on the value of $\langle\theta, \mathbf{X}\rangle$:

$$P(Y=1) = \frac{1}{1+\exp^{-\langle\theta,\mathbf{X}\rangle}}$$

- For the other class:

$$P(Y=-1) = 1 - P(Y=1) = \frac{\exp^{-\langle\theta,\mathbf{X}\rangle}}{1+\exp^{-\langle\theta,\mathbf{X}\rangle}} = \frac{1}{1+\exp^{\langle\theta,\mathbf{X}\rangle}}$$

- Final class: $\hat{c} = \arg\max_c P(Y=c)$

Multinomial Logistic Regression and Softmax

From Binary to Multinomial Logistic Regression

- For a binary linear classifier with the logistic function:

$$P(Y=1) = \frac{1}{1+\exp^{-\langle\theta,\mathbf{x}\rangle}} = \frac{\exp^{\langle\theta,\mathbf{x}\rangle}}{\exp^{\langle\theta,\mathbf{x}\rangle}+1} = \frac{\exp^{\langle\theta^{(1)}-\theta^{(-1)},\mathbf{x}\rangle}}{\exp^{\langle\theta^{(1)}-\theta^{(-1)},\mathbf{x}\rangle}+1}$$

$$= \frac{\exp^{\langle\theta^{(1)},\mathbf{x}\rangle}}{\exp^{\langle\theta^{(1)},\mathbf{x}\rangle}+\exp^{\langle\theta^{(-1)},\mathbf{x}\rangle}}$$

- For multiple classes: $P(Y=c) = \frac{\exp^{\langle\theta^{(c)},\mathbf{x}\rangle}}{\sum_{j=1}^{C}\exp^{\langle\theta^{(j)},\mathbf{x}\rangle}}$, with:

$$\theta = \begin{pmatrix} | & | & | & | \\ \theta^{(1)} & \theta^{(2)} & \dots & \theta^{(C)} \\ | & | & | & | \end{pmatrix}$$

- Final class: $\hat{c} = \arg\max_c P(Y=c)$

Logistic Regression – Cost Functions

Key Points

- Logistic cost function, with $Y \in \{-1, 1\}$: $\ell(\mathbf{X}, Y) = \frac{1}{1+\exp^{Y\langle\theta,\mathbf{X}\rangle}}$
- Cross-entropy cost function, with $Y \in \{0, 1\}$:

$$\begin{aligned}\ell(\mathbf{X}, Y) &= -(Y \log(\frac{1}{1+\exp^{-\langle\theta,\mathbf{X}\rangle}}) + (1-Y)\log(\frac{1}{1+\exp^{\langle\theta,\mathbf{X}\rangle}})) \\ &= -\sum_{c=1}^{C} \mathbb{1}_{\{Y=c\}} \log(P(Y=c))\end{aligned}$$

Exercise: What is the derivative $\nabla_{\theta^{(k)}}\ell(\mathbf{X}, Y; \theta)$ for the cross-entropy cost function?

Logistic Regression – Cost Functions

Key Points

- Logistic cost function, with $Y \in \{-1, 1\}$: $\ell(\mathbf{X}, Y) = \frac{1}{1+\exp^{Y\langle\theta,\mathbf{X}\rangle}}$
- Cross-entropy cost function, with $Y \in \{0, 1\}$:

$$\ell(\mathbf{X}, Y) = -(Y \log(\frac{1}{1+\exp^{-\langle\theta,\mathbf{X}\rangle}}) + (1-Y)\log(\frac{1}{1+\exp^{\langle\theta,\mathbf{X}\rangle}}))$$
$$= -\sum_{c=1}^{C} \mathbb{1}_{\{Y=c\}} \log(P(Y=c))$$

Solution:

$$\nabla_{\theta^{(k)}} \ell(\mathbf{X}, Y; \theta) = -\mathbf{X}(\mathbb{1}_{\{Y=k\}} - P(Y=k))$$

(intuition: $\sigma' = \sigma \cdot (1-\sigma)$ and $\ln(u) = \frac{u'}{u}$)

Assumptions of the Linear Model

Linearity

The response variable is linearly related to the features.

Normality

The errors are normally distributed with zero mean: $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

Homoscedasticity

The errors have constant variance (same σ^2).

Independence

The errors are mutually independent.

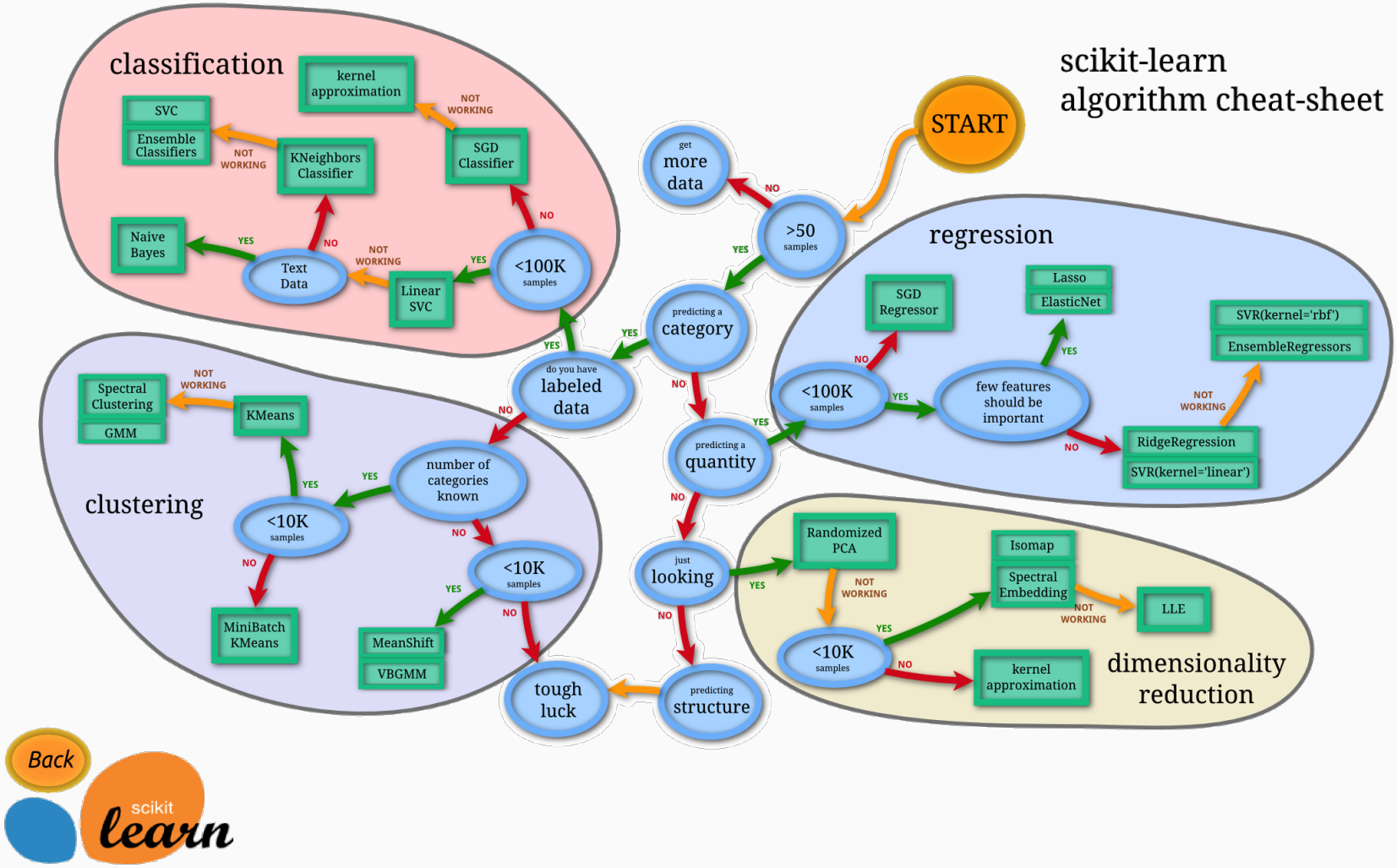
API: Scikit-learn

Outline : API: Scikit-learn

Scikit-learn: A Python Library

A very user-friendly Machine Learning library:
https://scikit-learn.org/

Scikit-learn: Possibility Map



Scikit-learn: Standardized Functions

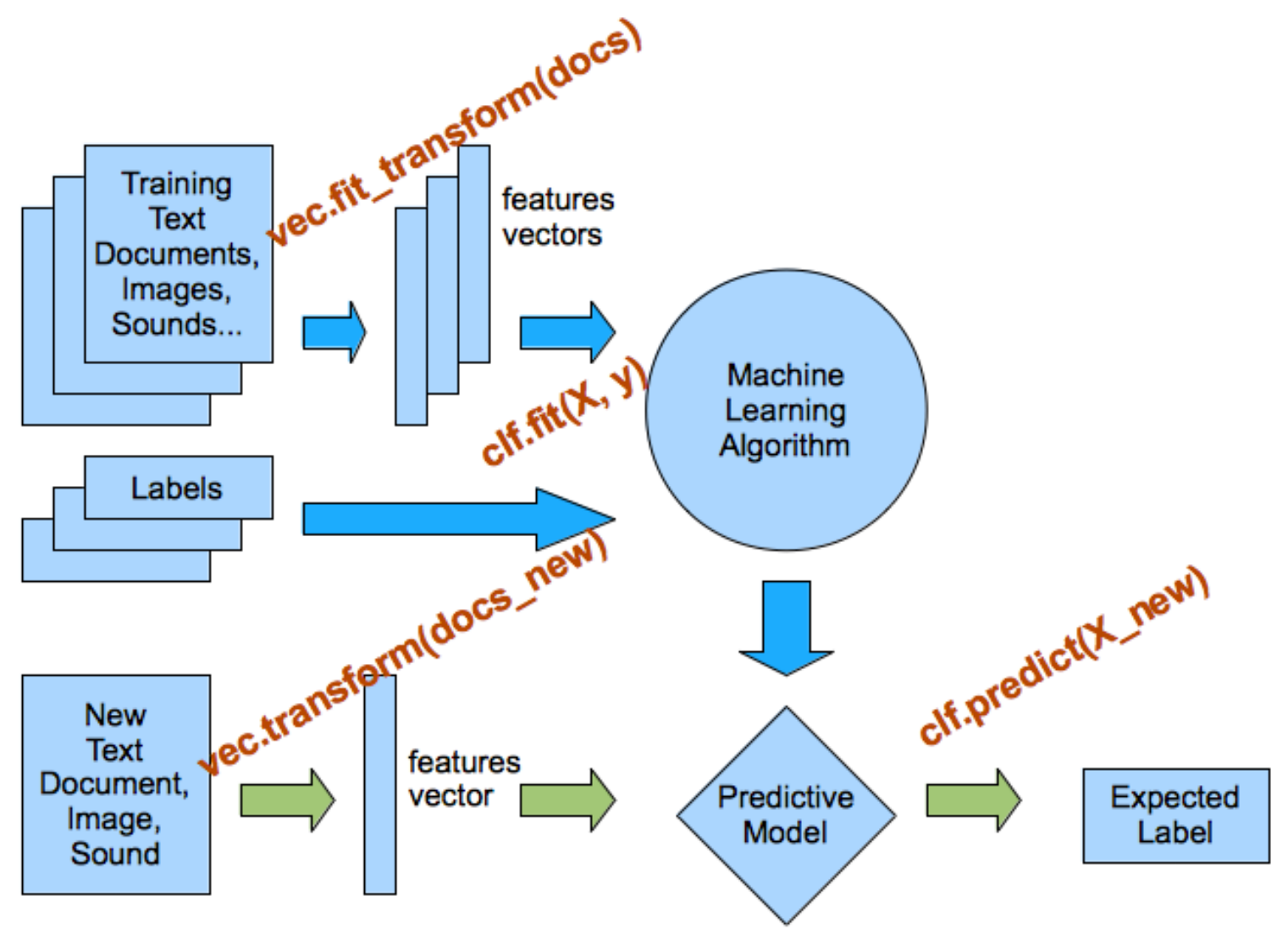
Standardized functions for fast learning and code clarity:

- Different models (or datasets) are implemented as classes (data types)
- Methods can be called on these objects for different stages of the algorithm: Feature extraction, Parameter learning, Prediction, etc.

```
>>> from sklearn.datasets import load_iris
>>> data = load_iris()
>>> X, y = data.data, data.target
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)
>>> from sklearn.linear_model import SGDClassifier
>>> clf = SGDClassifier(max_iter=1000, tol=1e-3)
>>> clf.fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.92...
```

Scikit-learn: Standardized Functions

Standardized functions for fast learning and code clarity:



Scikit-learn: Datasets

Multiple available datasets :

- Toy datasets: Boston housing prices, Iris, Diabetes, Wine recognition, etc.
- Real-world datasets: Olivetti Faces (images), Forest cover types (geographic), RCV1 dataset (text), etc.

```
>>> from sklearn.datasets import load_wine
>>> data = load_wine()
>>> data.keys()
['target_names', 'data', 'target', ...
'DESCR', 'feature_names']
```

Scikit-learn: Feature Extraction

Pre-coded functions to easily vectorize features

- From dictionaries: `sklearn.feature_extraction`
- From images: `sklearn.feature_extraction.image`
- From text: `sklearn.feature_extraction.text`

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This is the second second document.',
...     'And the third.',
...     'Is this the first document?',
... ]
>>> X = vectorizer.fit_transform(corpus).toarray()
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 2, 1, 0, 1],
       [1, 0, 0, 0, 1, 0, 1, 1, 0],
       [0, 1, 1, 1, 0, 0, 1, 0, 1]]...)
```

Scikit-learn: Feature Selection

Techniques to select features

- Remove low-variance features: `VarianceThreshold`
- Use models to select features: `SelectFromModel`
- Recursive Feature Elimination: `RFE`

```
>>> from sklearn.svm import LinearSVC
>>> from sklearn.datasets import load_iris
>>> from sklearn.feature_selection import SelectFromModel
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X.shape
(150, 4)
>>> lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
>>> model = SelectFromModel(lsvc, prefit=True)
>>> X_new = model.transform(X)
>>> X_new.shape
(150, 3)
```

Scikit-learn: Data Preprocessing

Normalization, standardization necessary for better learning

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
scaler.transform(X_test)
```

Missing values

```
>>> from sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
SimpleImputer(copy=True, fill_value=None, missing_values=nan, strateg
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> print(imp.transform(X))
[[4.          2.        ]
 [6.          3.666...]
 [7.          6.        ]]
```

Scikit-learn: Cross-Validation

Functions for model validation and splitting into training/test sets:

- Simply using `train_test_split`, for K-Fold cross-validation `KFold`, for Leave-One-Out `LeaveOneOut`, preserving label proportions with `StratifiedKFold`
- Direct cross-validation with `cross_val_score`

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)

>>> X_train.shape, y_train.shape
((90, 4), (90,))
>>> X_test.shape, y_test.shape
((60, 4), (60,))
```

Scikit-learn: Supervised Learning Models

Numerous standardized supervised models for easy use:

- Logistic/Linear Regression, Linear/Quadratic Discriminant Analysis, SVM, K-NN, Stochastic Gradient Descent, Naive Bayes, Decision Trees, Random Forests, etc.
- Each of these models is implemented as a Python class with common methods: `fit`, `score`, `predict_proba`, `predict`, etc.

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0, solver='lbfgs',
...                          multi_class='multinomial').fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

Scikit-learn: Unsupervised Learning Models

Numerous standardized unsupervised models for clustering, component analysis, dimensionality reduction:

- Clustering: K-Means, PCA, Non-negative Matrix Factorization, Latent Dirichlet Allocation, t-SNE, ...
- These models are implemented as Python classes with common methods: `fit`, `fit_transform`, `predict`, etc.

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [4, 2], [4, 4], [4, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([0, 0, 0, 1, 1, 1], dtype=int32)
>>> kmeans.predict([[0, 0], [4, 4]])
array([0, 1], dtype=int32)
>>> kmeans.cluster_centers_
array([[1., 2.],
       [4., 2.]])
```

Scikit-learn: Metrics

Multiple available metrics for model evaluation:

- For classification: Recall, Precision, F1, AUC, ROC, Confusion Matrix, etc.
- For regression: R^2, Mean Squared Error, etc.
- For clustering: Completeness, V-measure, etc.

```
>>> import numpy as np
>>> from sklearn.metrics import accuracy_score
>>> y_pred = [0, 2, 1, 3]
>>> y_true = [0, 1, 2, 3]
>>> accuracy_score(y_true, y_pred)
0.5
```

Scikit-learn: Hyperparameter Search

Functions to find optimal hyperparameters using cross-validation for different hyperparameters.

- Grid search: `GridSearchCV`
- Random search: `RandomizedSearchCV`
- These models are implemented as Python classes with common methods: `fit`, `fit_transform`, `predict`, etc.

```
>>> from sklearn import svm, datasets
>>> from sklearn.model_selection import GridSearchCV
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svc = svm.SVC(gamma="scale")
>>> clf = GridSearchCV(svc, parameters, cv=5)
>>> clf.fit(iris.data, iris.target)
```

Outline : TP: Linear Regression

TP Introduction to Scikit-learn and Linear Regression

Study of regression by fitting a linear plane to a quadratic surface:

$$Y = 3X^{(1)} - 2X^{(2)^2} + \epsilon$$

Introduction to Scikit-learn:

- Splitting into training and test sets
- Creation, training, and testing of Scikit-learn regression models
- Feature expansion to create a nonlinear model
- Comparison with other less suitable models

Questions?

References i