Lecture 05

Machine Learning 2:

classical learning - classification

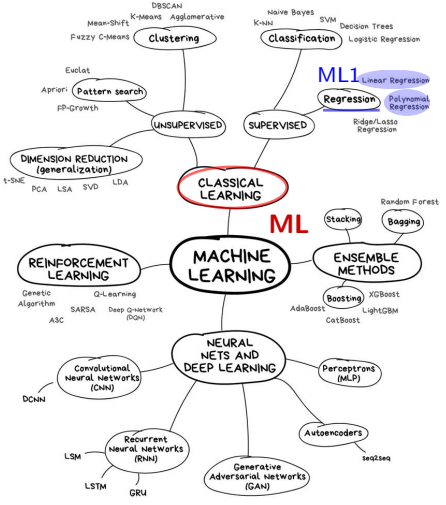
2026-03-20

Sébastien Valade

VNIVERSIDAD NACIONAL
AVTÈNOMA DE
MEXICO

Introduction

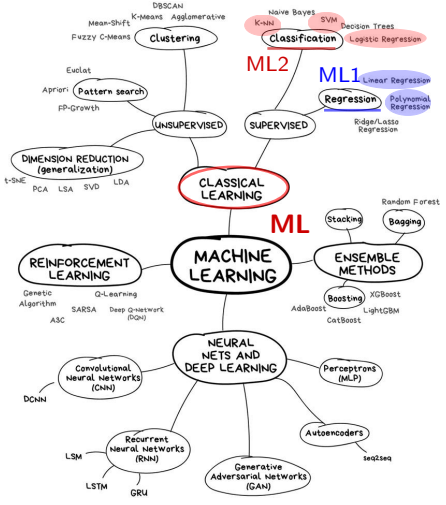
Last week: regression (ML1)



source

Introduction

Last week: regression (ML1)
This week: classification (ML2)



source

Introduction

Reminder:

$\Rightarrow$ the goal of **supervised learning** is to learn a function f which maps low-level image features (X) to high-level image information (Y), using training data (i.e. known pairs of (X_i, Y_i)):

- $\rightarrow$ **classification task** $\Rightarrow$ extract semantic classes (output=discrete labels, categorical values)
- $\rightarrow$ **regression task** $\Rightarrow$ extract measurements (output=continuous number)

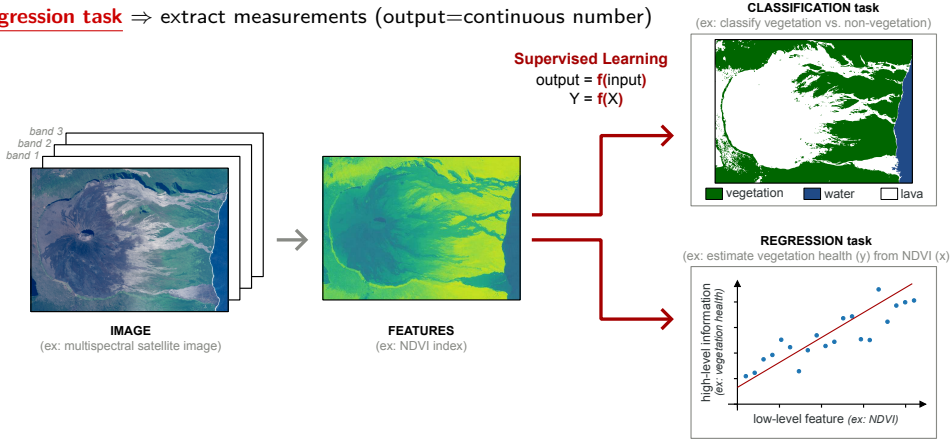
Introduction

Reminder:

⇒ the goal of **supervised learning** is to learn a function f which maps low-level image features (X) to high-level image information (Y), using training data (i.e. known pairs of (X_i, Y_i)):

→ **classification task** ⇒ extract semantic classes (output=discrete labels, categorical values)

→ **regression task** ⇒ extract measurements (output=continuous number)



**the term "feature"* is here used in a broad sense, referring to any information extracted from the image

What is classification?

⇒ the goal of **classification** is to assign **class labels** Y (i.e., discrete categorical values) to data points (e.g., pixels, images)

⇒ extracting **features** from the data is useful to find a space where samples from different classes are well separable (feature crafting can be either *manual*, or *learned* from the data using *unsupervised learning*, e.g. PCA)

⇒ the classification algorithm will have to learn the **decision boundary** in an N-dimensional **feature space**

What is classification?

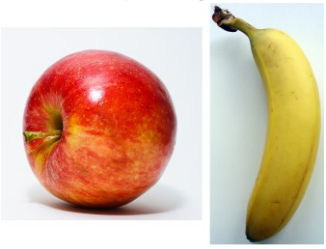
⇒ the goal of **classification** is to assign **class labels** Y (i.e., discrete categorical values) to data points (e.g., pixels, images)

⇒ extracting **features** from the data is useful to find a space where samples from different classes are well separable (feature crafting can be either *manual*, or *learned* from the data using *unsupervised learning*, e.g. PCA)

⇒ the classification algorithm will have to learn the **decision boundary** in an N-dimensional **feature space**

Toy example: *classify fruit images into either bananas or apples*

input image



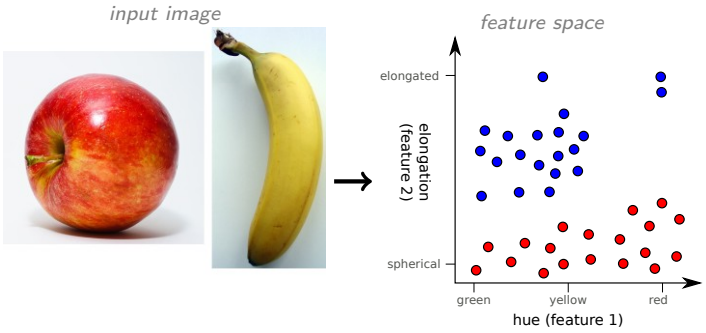
What is classification?

⇒ the goal of **classification** is to assign **class labels** Y (i.e., discrete categorical values) to data points (e.g., pixels, images)

⇒ extracting **features** from the data is useful to find a space where samples from different classes are well separable (feature crafting can be either *manual*, or *learned* from the data using *unsupervised learning*, e.g. PCA)

⇒ the classification algorithm will have to learn the **decision boundary** in an N-dimensional **feature space**

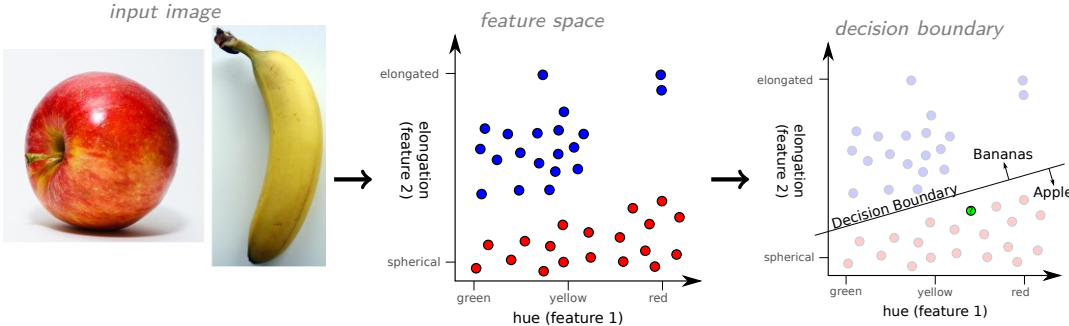
Toy example: *classify fruit images into either bananas or apples*



What is classification?

- ⇒ the goal of **classification** is to assign **class labels** Y (i.e., discrete categorical values) to data points (e.g., pixels, images)
- ⇒ extracting **features** from the data is useful to find a space where samples from different classes are well separable (feature crafting can be either *manual*, or *learned* from the data using *unsupervised learning*, e.g. PCA)
- ⇒ the classification algorithm will have to learn the **decision boundary** in an N-dimensional **feature space**

Toy example: *classify fruit images into either bananas or apples*



How is the decision boundary learned?

⇒ **decision boundary** = surface separating the feature space into different regions (= different classes)

⇒ many algorithms exist to learn this boundary:

1. *probabilistic approaches*:
   - **Logistic Regression**
     ⇒ *estimates class probability using a logistic function, fitting a linear decision boundary (binary classification)*
   - **Softmax Regression**
     ⇒ *multi-class extension of logistic regression, probabilities assigned to each class (multi-class classification)*
   - **Naive Bayes**
     ⇒ *based on Bayes' theorem, probabilistic reasoning to calculate the likelihood of class membership*
2. *deterministic approaches*:
   - **Perceptron**
     ⇒ *finds a hyperplane to separate classes, adjusting weights based on misclassified points*
   - **k-Nearest Neighbors (kNN)**
     ⇒ *non-parametric method that classifies based on the majority class of the nearest neighbors, leading to non-linear boundaries*
   - **Support Vector Machines (SVM)**
     ⇒ *finds the optimal hyperplane that maximizes the margin between classes*
   - **Random Forest**
     ⇒ *ensemble of decision trees that vote on the final classification. Each tree learns decision boundaries that can be linear or non-linear*
   - **Neural Networks** (MLP, CNN)
     ⇒ *next lectures*

How is the decision boundary learned?

⇒ **decision boundary** = surface separating the feature space into different regions (= different classes)

⇒ many algorithms exist to learn this boundary:

1. *probabilistic approaches*:
   - **Logistic Regression**
     ⇒ *estimates class probability using a logistic function, fitting a linear decision boundary (binary classification)*
   - **Softmax Regression**
     ⇒ *multi-class extension of logistic regression, probabilities assigned to each class (multi-class classification)*
   - **Naive Bayes**
     ⇒ *based on Bayes' theorem, probabilistic reasoning to calculate the likelihood of class membership*
2. *deterministic approaches*:
   - **Perceptron**
     ⇒ *finds a hyperplane to separate classes, adjusting weights based on misclassified points*
   - **k-Nearest Neighbors (kNN)**
     ⇒ *non-parametric method that classifies based on the majority class of the nearest neighbors, leading to non-linear boundaries*
   - **Support Vector Machines (SVM)**
     ⇒ *finds the optimal hyperplane that maximizes the margin between classes*
   - **Random Forest**
     ⇒ *ensemble of decision trees that vote on the final classification. Each tree learns decision boundaries that can be linear or non-linear*
   - **Neural Networks** (MLP, CNN)
     ⇒ *next lectures*

How is the decision boundary learned?

⇒ **decision boundary** = surface separating the feature space into different regions (= different classes)

⇒ many algorithms exist to learn this boundary:

1. *probabilistic approaches*:
   - **Logistic Regression**
     ⇒ *estimates class probability using a logistic function, fitting a linear decision boundary (binary classification)*
   - **Softmax Regression**
     ⇒ *multi-class extension of logistic regression, probabilities assigned to each class (multi-class classification)*
   - **Naive Bayes**
     ⇒ *based on Bayes' theorem, probabilistic reasoning to calculate the likelihood of class membership*
2. *deterministic approaches*:
   - **Perceptron**
     ⇒ *finds a hyperplane to separate classes, adjusting weights based on misclassified points*
   - **k-Nearest Neighbors (kNN)**
     ⇒ *non-parametric method that classifies based on the majority class of the nearest neighbors, leading to non-linear boundaries*
   - **Support Vector Machines (SVM)**
     ⇒ *finds the optimal hyperplane that maximizes the margin between classes*
   - **Random Forest**
     ⇒ *ensemble of decision trees that vote on the final classification. Each tree learns decision boundaries that can be linear or non-linear*
   - **Neural Networks** (MLP, CNN)
     ⇒ *next lectures*

Choosing a classifier

- **Logistic regression & Softmax** *(this week lecture)*
  - ⇒ probability-based linear classification method
  - ⇒ *advantage*: simple, fast, interpretable
  - ⇒ *disadvantage*: limited to linear decision boundaries
- **k-Nearest Neighbor (kNN)** *(this week lecture)*
  - ⇒ label images by comparing them to (annotated) images from the training set
  - ⇒ *advantage*: non-linear decision boundaries
  - ⇒ *disadvantage*: classifier needs to keep all training data for future comparisons with the test data *(classifying test images is expensive as it requires comparison to all training images, inefficient with v. large datasets $\geq$GB)*
- **Support Vector Machines** *(this week lecture)*
  - ⇒ parametric linear classification method
  - ⇒ *advantage*: once the parameters are learnt, training data can be discarded *(classification of new images is fast: simple matrix multiplication with learned weights, not an exhaustive comparison to every single training data)*
- **Neural Networks** *(coming weeks)*
  - ⇒ CNNs map image pixels to classes, but the mapping is more complex and will contain more parameters
  - ⇒ *advantage*: very powerful
  - ⇒ *disadvantage*: needs LOTS of data!

Choosing a classifier

- **Logistic regression & Softmax** *(this week lecture)*
  - ⇒ probability-based linear classification method
  - ⇒ *advantage*: simple, fast, interpretable
  - ⇒ *disadvantage*: limited to linear decision boundaries
- **k-Nearest Neighbor (kNN)** *(this week lecture)*
  - ⇒ label images by comparing them to (annotated) images from the training set
  - ⇒ *advantage*: non-linear decision boundaries
  - ⇒ *disadvantage*: classifier needs to keep all training data for future comparisons with the test data *(classifying test images is expensive as it requires comparison to all training images, inefficient with v. large datasets $\geq$GB)*
- **Support Vector Machines** *(this week lecture)*
  - ⇒ parametric linear classification method
  - ⇒ *advantage*: once the parameters are learnt, training data can be discarded *(classification of new images is fast: simple matrix multiplication with learned weights, not an exhaustive comparison to every single training data)*
- **Neural Networks** *(coming weeks)*
  - ⇒ CNNs map image pixels to classes, but the mapping is more complex and will contain more parameters
  - ⇒ *advantage*: very powerful
  - ⇒ *disadvantage*: needs LOTS of data!

Choosing a classifier

- **Logistic regression & Softmax** *(this week lecture)*
  - ⇒ probability-based linear classification method
  - ⇒ *advantage*: simple, fast, interpretable
  - ⇒ *disadvantage*: limited to linear decision boundaries
- **k-Nearest Neighbor (kNN)** *(this week lecture)*
  - ⇒ label images by comparing them to (annotated) images from the training set
  - ⇒ *advantage*: non-linear decision boundaries
  - ⇒ *disadvantage*: classifier needs to keep all training data for future comparisons with the test data *(classifying test images is expensive as it requires comparison to all training images, inefficient with v. large datasets $\geq$GB)*
- **Support Vector Machines** *(this week lecture)*
  - ⇒ parametric linear classification method
  - ⇒ *advantage*: once the parameters are learnt, training data can be discarded *(classification of new images is fast: simple matrix multiplication with learned weights, not an exhaustive comparison to every single training data)*
- **Neural Networks** *(coming weeks)*
  - ⇒ CNNs map image pixels to classes, but the mapping is more complex and will contain more parameters
  - ⇒ *advantage*: very powerful
  - ⇒ *disadvantage*: needs LOTS of data!

Choosing a classifier

- **Logistic regression & Softmax** *(this week lecture)*
  - ⇒ probability-based linear classification method
  - ⇒ *advantage*: simple, fast, interpretable
  - ⇒ *disadvantage*: limited to linear decision boundaries
- **k-Nearest Neighbor (kNN)** *(this week lecture)*
  - ⇒ label images by comparing them to (annotated) images from the training set
  - ⇒ *advantage*: non-linear decision boundaries
  - ⇒ *disadvantage*: classifier needs to keep all training data for future comparisons with the test data *(classifying test images is expensive as it requires comparison to all training images, inefficient with v. large datasets $\geq GB$)*
- **Support Vector Machines** *(this week lecture)*
  - ⇒ parametric linear classification method
  - ⇒ *advantage*: once the parameters are learnt, training data can be discarded *(classification of new images is fast: simple matrix multiplication with learned weights, not an exhaustive comparison to every single training data)*
- **Neural Networks** *(coming weeks)*
  - ⇒ CNNs map image pixels to classes, but the mapping is more complex and will contain more parameters
  - ⇒ *advantage*: very powerful
  - ⇒ *disadvantage*: needs LOTS of data!

Probabilistic classification

Probabilistic classification

⇒ learns $p(y|X)$, i.e. probability of class given features

⇒ decision = threshold (binary) or argmax (multi-class) on $p(y|X)$

⇒ algorithms:

- **Logistic** ⇒ *binary, linear, discriminative (learns $p(y|X)$)*
- **Softmax** ⇒ *multi-class, linear, discriminative (learns $p(y|X)$)*

Non-probabilistic classification

⇒ learns boundary / rule in feature space

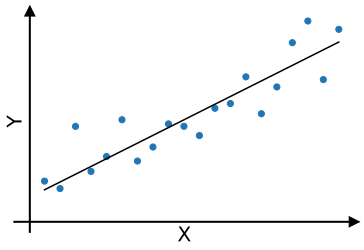
⇒ decision = side of boundary / neighbor vote

⇒ algorithms:

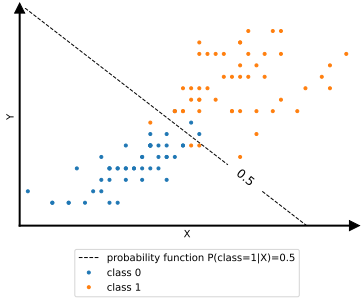
- **kNN** ⇒ *multi-class, non-parametric, non-linear (instance-based voting)*
- **SVM** ⇒ *linear margin maximization ("perceptron of optimal stability"), kernel linear | non-linear*

Logistic Regression

- Linear Regression *(recap)*
  - ⇒ used to predict continuous values of Y given X
  - ⇒ models the relationship between X and Y as a *linear equation*: $Y = \beta_0 + \beta_1 X$
  - ⇒ best model parameters (β_0, β_1) are found by *minimizing Mean Squared Error (MSE)*

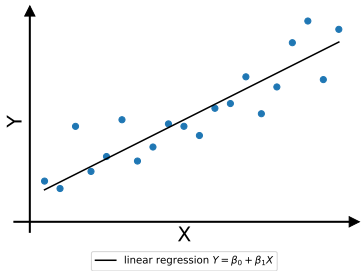


- Logistic Regression
  - ⇒ used to predict binary class values (discrete categorical values $y \in 0, 1$) of a data point, given features (X, Y)
  - ⇒ models a probability function using the *logistic function*: $p(y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$
  - ⇒ best model parameters (β_0, β_1) are found by *maximizing Likelihood*

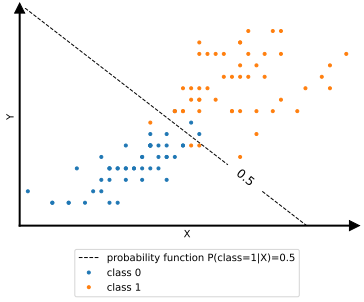


Logistic Regression

- Linear Regression *(recap)*
  - ⇒ used to predict continuous values of Y given X
  - ⇒ models the relationship between X and Y as a *linear equation*: $Y = \beta_0 + \beta_1 X$
  - ⇒ best model parameters (β_0, β_1) are found by *minimizing Mean Squared Error (MSE)*



- Logistic Regression
  - ⇒ used to predict binary class values (discrete categorical values $y \in 0, 1$) of a data point, given features (X, Y)
  - ⇒ models a probability function using the *logistic function*: $p(y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$
  - ⇒ best model parameters (β_0, β_1) are found by *maximizing Likelihood*



Logistic Regression

⇒ in order to model the binary class probabilities, we use the **logistic function** $\sigma(X)$ (a.k.a. *sigmoid function*, or *S-shaped curve*), which maps any real value of feature X to the range [0,1]:

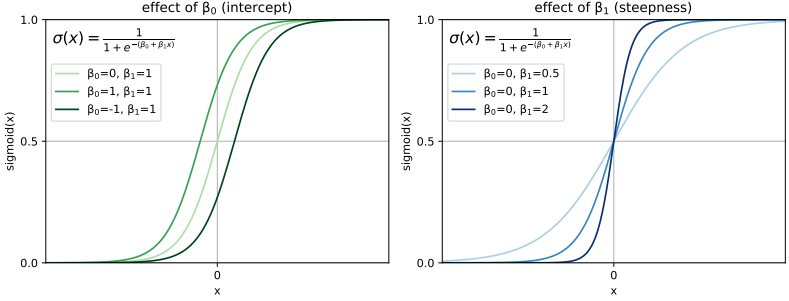
$$\sigma(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

Logistic Regression

⇒ in order to model the binary class probabilities, we use the **logistic function** $\sigma(X)$ (a.k.a. *sigmoid function*, or *S-shaped curve*), which maps any real value of feature X to the range [0,1]:

$$\sigma(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

⇒ depends on 2 parameters: β_0 (intercept) and β_1 (slope)



Logistic Regression

$\Rightarrow$ the logistic function can also be expressed in **vectorized form** *(convenient because it is compact and scales naturally to multiple features)*:

$$\sigma(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} = \frac{1}{1 + e^{-\theta^\top X}}$$

with: $\theta = [\beta_0, \beta_1]$ and $X = [1, X]$

Logistic Regression

⇒ the logistic function can also be expressed in **vectorized form** *(convenient because it is compact and scales naturally to multiple features)*:

$$\sigma(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} = \frac{1}{1 + e^{-\theta^\top X}}$$

$$\text{with:} \quad \theta = [\beta_0, \beta_1] \quad \text{and} \quad X = [1, X]$$

⇒ the probability p is then calculated as:

$$p(y = 1|X) = \sigma(X)$$
$$p(y = 0|X) = 1 - \sigma(X)$$

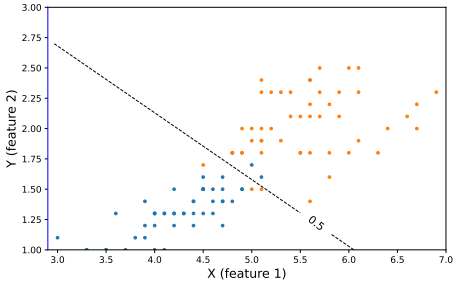
⇒ the prediction of the class $\hat{y} \in \{0, 1\}$ is made by comparing the probability $p(X)$ to a threshold (e.g. 0.5):

$$\hat{y} = \begin{cases} 1 & \text{if } p(X) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

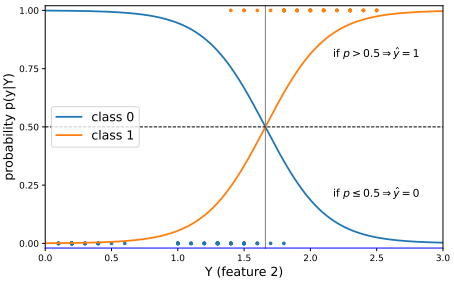
Logistic Regression

EX: estimate data point class $\hat{y} \in \{0|1\}$ from estimated probabilities $p(y|Y)$

feature space



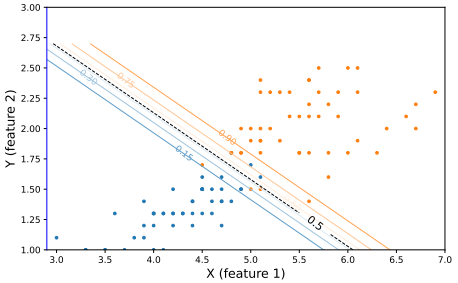
estimated probabilities and decision boundary



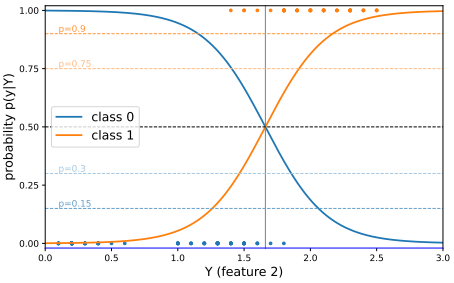
Logistic Regression

EX: estimate data point class $\hat{y} \in \{0|1\}$ from estimated probabilities $p(y|Y)$

feature space



estimated probabilities and decision boundary



Logistic Regression

$\Rightarrow$ the coefficients β_0 and β_1 are unknown, and must be estimated based on the available training data

$\Rightarrow$ the coefficients are estimated by using the **likelihood function**

- $\rightarrow$ *find best estimates of β_0 and β_1, such that the predicted probability $\hat{p}(x_i)$ for each data point returns as closely as possible to the expected class*
- $\rightarrow$ *this can be formalized using a mathematical equation called a likelihood function:*

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

- $\rightarrow$ *the maximization of the likelihood function allows for the estimation of the best parameters β_0 and β_1:*

$$\hat{\beta}_0, \hat{\beta}_1 = \arg\max_{\beta_0, \beta_1} L(\beta_0, \beta_1)$$

- $\rightarrow$ *maximizing the likelihood has no analytical solution*
  $\Rightarrow$ *must use an iterative solver (e.g., gradient descent $\rightarrow$ next lecture)*

Logistic Regression

$\Rightarrow$ the coefficients β_0 and β_1 are unknown, and must be estimated based on the available training data

$\Rightarrow$ the coefficients are estimated by using the **likelihood function**

$\rightarrow$ *find best estimates of β_0 and β_1, such that the predicted probability $\hat{p}(x_i)$ for each data point returns as closely as possible to the expected class*

$\rightarrow$ *this can be formalized using a mathematical equation called a likelihood function:*

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

$\rightarrow$ *the maximization of the likelihood function allows for the estimation of the best parameters β_0 and β_1:*

$$\hat{\beta}_0, \hat{\beta}_1 = \arg\max_{\beta_0, \beta_1} L(\beta_0, \beta_1)$$

$\rightarrow$ *maximizing the likelihood has no analytical solution*
$\Rightarrow$ must use an iterative solver (e.g., gradient descent $\rightarrow$ next lecture)

Logistic Regression

$\Rightarrow$ the coefficients β_0 and β_1 are unknown, and must be estimated based on the available training data

$\Rightarrow$ the coefficients are estimated by using the **likelihood function**

$\rightarrow$ *find best estimates of β_0 and β_1, such that the predicted probability $\hat{p}(x_i)$ for each data point returns as closely as possible to the expected class*

$\rightarrow$ *this can be formalized using a mathematical equation called a likelihood function:*

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

$\rightarrow$ *the maximization of the likelihood function allows for the estimation of the best parameters β_0 and β_1:*

$$\hat{\beta}_0, \hat{\beta}_1 = \arg\max_{\beta_0, \beta_1} L(\beta_0, \beta_1)$$

$\rightarrow$ *maximizing the likelihood has no analytical solution*
$\Rightarrow$ *must use an iterative solver (e.g., gradient descent $\rightarrow$ next lecture)*

Logistic Regression

$\Rightarrow$ the coefficients β_0 and β_1 are unknown, and must be estimated based on the available training data

$\Rightarrow$ the coefficients are estimated by using the **likelihood function**

- $\rightarrow$ *find best estimates of β_0 and β_1, such that the predicted probability $\hat{p}(x_i)$ for each data point returns as closely as possible to the expected class*
- $\rightarrow$ *this can be formalized using a mathematical equation called a likelihood function:*

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

- $\rightarrow$ *the maximization of the likelihood function allows for the estimation of the best parameters β_0 and β_1:*

$$\hat{\beta}_0, \hat{\beta}_1 = \arg\max_{\beta_0, \beta_1} L(\beta_0, \beta_1)$$

- $\rightarrow$ *maximizing the likelihood has no analytical solution*
  $\Rightarrow$ *must use an iterative solver (e.g., gradient descent $\rightarrow$ next lecture)*

Logistic Regression

$\Rightarrow$ the coefficients β_0 and β_1 are unknown, and must be estimated based on the available training data

$\Rightarrow$ the coefficients are estimated by using the **likelihood function**

- $\rightarrow$ *find best estimates of β_0 and β_1, such that the predicted probability $\hat{p}(x_i)$ for each data point returns as closely as possible to the expected class*
- $\rightarrow$ *this can be formalized using a mathematical equation called a likelihood function:*

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

- $\rightarrow$ *the maximization of the likelihood function allows for the estimation of the best parameters β_0 and β_1:*

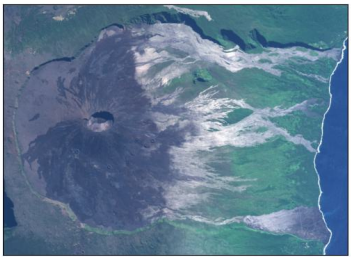
$$\hat{\beta}_0, \hat{\beta}_1 = \arg\max_{\beta_0, \beta_1} L(\beta_0, \beta_1)$$

- $\rightarrow$ *maximizing the likelihood has no analytical solution*
  $\Rightarrow$ *must use an iterative solver (e.g., gradient descent $\rightarrow$ next lecture)*

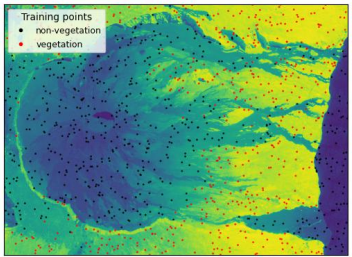
Logistic Regression

EX: estimate pixel class $\hat{y} \in \{0|1\} \Rightarrow$ *vegetation* ($\hat{y} = 1$) or *non-vegetation* ($\hat{y} = 0$)

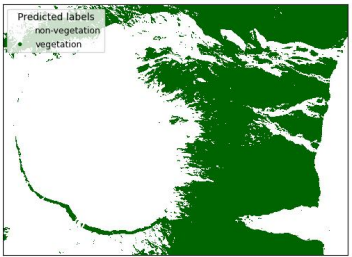
RGB image



NDVI image with training data points



Predicted vegetation map



Logistic Regression

⇒ Logistic Regression can also be used to to predict a binary response but using *multiple predictors* (in the previous slides, the binary class prediction was done using just the 1 feature Y)

⇒ if we consider k predictors, the model is then defined as:

$$p(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k)}}$$
$$= \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k}}$$

NB: the classification results obtained using one predictor or several may be quite different, especially when there is correlation among the predictors!

⇒ nevertheless, **Logistic Regression** is limited as it can only model **binary classification**
⇒ for **multi-class classification**, we need to use **Softmax Regression**

Logistic Regression

⇒ Logistic Regression can also be used to to predict a binary response but using *multiple predictors* (in the previous slides, the binary class prediction was done using just the 1 feature Y)

⇒ if we consider k predictors, the model is then defined as:

$$p(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k)}} = \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k}}$$

NB: the classification results obtained using one predictor or several may be quite different, especially when there is correlation among the predictors!

⇒ nevertheless, **Logistic Regression** is limited as it can only model **binary classification**
⇒ for **multi-class classification**, we need to use **Softmax Regression**

Logistic Regression

⇒ Logistic Regression can also be used to to predict a binary response but using *multiple predictors*
(in the previous slides, the binary class prediction was done using just the 1 feature Y)

⇒ if we consider k predictors, the model is then defined as:

$$p(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k)}}$$
$$= \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k}}$$

NB: the classification results obtained using one predictor or several may be quite different, especially when there is correlation among the predictors!

⇒ nevertheless, **Logistic Regression** is limited as it can only model **binary classification**
⇒ for **multi-class classification**, we need to use **Softmax Regression**

Softmax Regression

⇒ the **Softmax Regression** (a.k.a. *Multinomial Logistic Regression*) generalizes the logistic regression for multiple classes, by calculating probabilities $p(y)$ for each class $y \in \{1, ..., K\}$

$$p(y = k|X; \theta) = \frac{\exp(\theta^{(k)\top} X)}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} X)}$$

where
- the numerator gives the exponentiated score for each class k
- the denominator normalizes the scores into a valid probability distribution

In other words:

$$\begin{bmatrix} p(y = 1|X; \theta) \\ p(y = 2|X; \theta) \\ \vdots \\ p(y = K|X; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} X)} \begin{bmatrix} exp(\theta^{(1)\top} X) \\ exp(\theta^{(2)\top} X) \\ \vdots \\ exp(\theta^{(K)\top} X) \end{bmatrix}$$

where $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$ are the model parameters

⇒ like with the *Logistic Regression classifier*, the *Softmax Regression classifier* predicts the class with the highest estimated probability:

$$\hat{y} = \arg\max_{k} p(y = k|X; \theta)$$

Softmax Regression

⇒ the **Softmax Regression** (a.k.a. *Multinomial Logistic Regression*) generalizes the logistic regression for multiple classes, by calculating probabilities $p(y)$ for each class $y \in \{1, ..., K\}$

$$p(y = k|X; \theta) = \frac{\exp(\theta^{(k)\top} X)}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} X)}$$

where { the numerator gives the exponentiated score for each class k
the denominator normalizes the scores into a valid probability distribution

In other words:

$$\begin{bmatrix} p(y = 1|X; \theta) \\ p(y = 2|X; \theta) \\ \vdots \\ p(y = K|X; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} X)} \begin{bmatrix} exp(\theta^{(1)\top} X) \\ exp(\theta^{(2)\top} X) \\ \vdots \\ exp(\theta^{(K)\top} X) \end{bmatrix}$$

where $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$ are the model parameters

⇒ like with the *Logistic Regression classifier*, the *Softmax Regression classifier* predicts the class with the highest estimated probability:

$$\hat{y} = \arg\max_{k} p(y = k|X; \theta)$$

Softmax Regression

⇒ the **Softmax Regression** (a.k.a. *Multinomial Logistic Regression*) generalizes the logistic regression for multiple classes, by calculating probabilities $p(y)$ for each class $y \in \{1, ..., K\}$

$$p(y = k|X; \theta) = \frac{\exp(\theta^{(k)\top} X)}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} X)}$$

where
- the numerator gives the exponentiated score for each class k
- the denominator normalizes the scores into a valid probability distribution

In other words:

$$\begin{bmatrix} p(y = 1|X; \theta) \\ p(y = 2|X; \theta) \\ \vdots \\ p(y = K|X; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} X)} \begin{bmatrix} exp(\theta^{(1)\top} X) \\ exp(\theta^{(2)\top} X) \\ \vdots \\ exp(\theta^{(K)\top} X) \end{bmatrix}$$

where $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$ are the model parameters

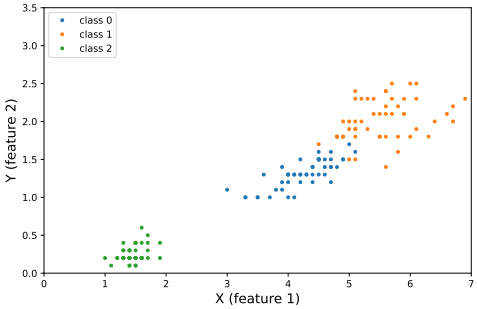
⇒ like with the *Logistic Regression classifier*, the *Softmax Regression classifier* predicts the class with the highest estimated probability:

$$\hat{y} = \arg\max_{k} p(y = k|X; \theta)$$

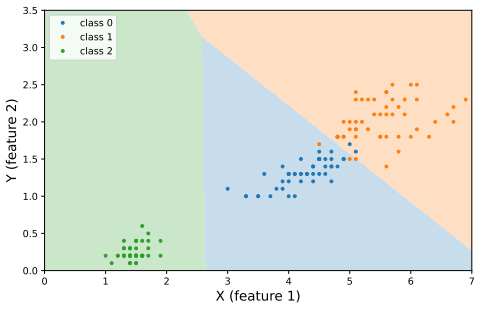
Softmax Regression

EX: estimate data point class $\hat{y} \in \{0, 1, 2\}$

data points



multiclass classification

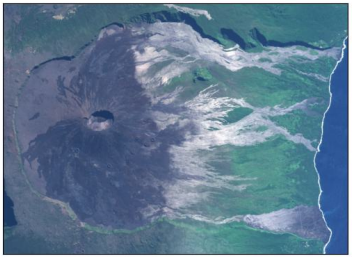


$\Rightarrow$ **Softmax Regression** finds linear boundaries between multiple classes

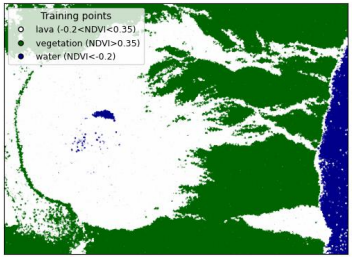
Logistic Regression

EX: estimate pixel class $\hat{y} \in \{0, 1, 2\} \Rightarrow$ *vegetation*, *water*, or *other*

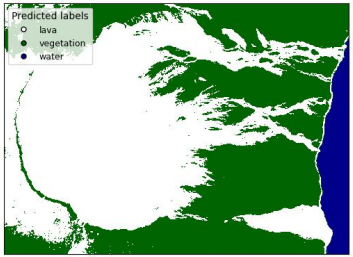
RGB image



NDVI image with training data points



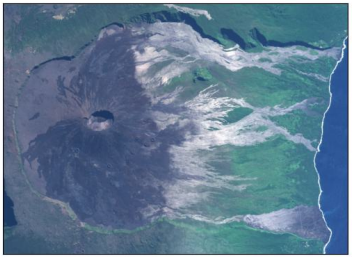
Predicted map



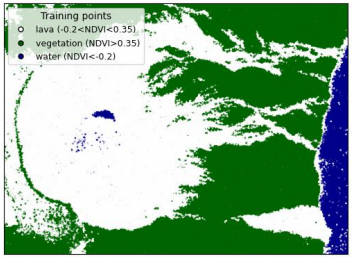
Logistic Regression

EX: estimate pixel class $\hat{y} \in \{0, 1, 2\} \Rightarrow$ *vegetation*, *water*, or *other*

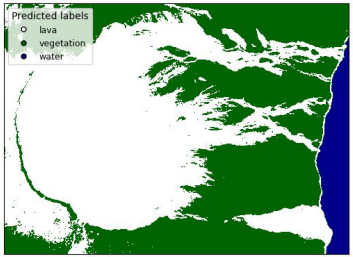
RGB image



NDVI image with training data points



Predicted map



$\Rightarrow$ **Softmax Regression** is commonly used as the final layer in **neural networks** for classification tasks

Non-probabilistic classification

Probabilistic classification

⇒ learns $p(y|X)$, i.e. probability of class given features

⇒ decision = threshold (binary) or argmax (multi-class) on $p(y|X)$

⇒ algorithms:

- **Logistic** ⇒ *binary, linear, discriminative (learns $p(y|X)$)*
- **Softmax** ⇒ *multi-class, linear, discriminative (learns $p(y|X)$)*

Non-probabilistic classification

⇒ learns boundary / rule in feature space

⇒ decision = side of boundary / neighbor vote

⇒ algorithms:

- **kNN** ⇒ *multi-class, non-parametric, non-linear (instance-based voting)*
- **SVM** ⇒ *linear margin maximization ("perceptron of optimal stability"), kernel linear | non-linear*

1. kNN classification

⇒ Idea: **classify** a data point based on the majority class of its **"k Nearest Neighbors"** (in feature space)

1. kNN classification

⇒ Idea: **classify** a data point based on the majority class of its **"k Nearest Neighbors"** (in feature space)

⇒ Method:

- → supervised (requires labeled training data)
- → non-parametric (no assumption on decision boundary)
- → non-linear decision boundary can be recovered

1. kNN classification

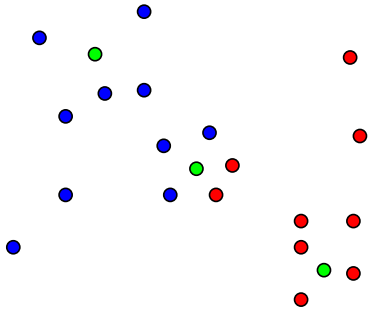
⇒ Idea: **classify** a data point based on the majority class of its **"k Nearest Neighbors"** (in feature space)

⇒ Method:

- → supervised (requires labeled training data)
- → non-parametric (no assumption on decision boundary)
- → non-linear decision boundary can be recovered

⇒ Steps:

1. *take random data points in the training dataset*



1. kNN classification

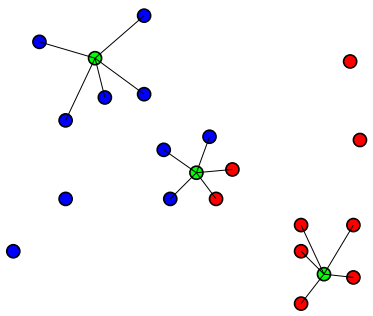
⇒ Idea: **classify** a data point based on the majority class of its **"k Nearest Neighbors"** (in feature space)

⇒ Method:

- → supervised (requires labeled training data)
- → non-parametric (no assumption on decision boundary)
- → non-linear decision boundary can be recovered

⇒ Steps:

1. *take random data points in the training dataset*
2. *for a sample find the k (e.g. 5) closest data points in the training dataset (k is a hyperparameter)*



1. kNN classification

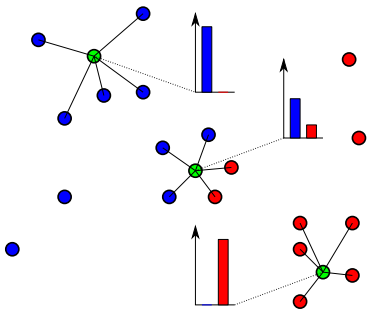
⇒ Idea: **classify** a data point based on the majority class of its **"k Nearest Neighbors"** (in feature space)

⇒ Method:

- → supervised (requires labeled training data)
- → non-parametric (no assumption on decision boundary)
- → non-linear decision boundary can be recovered

⇒ Steps:

1. *take random data points in the training dataset*
2. *for a sample find the k (e.g. 5) closest data points in the training dataset (k is a hyperparameter)*
3. *look at the neighbor labels, return/assign the mode*



1. kNN classification

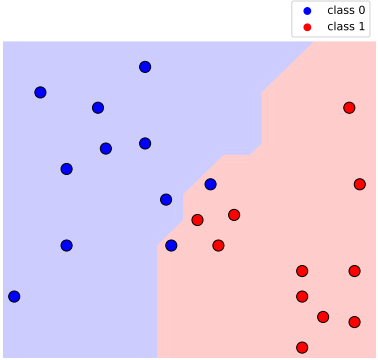
⇒ Idea: **classify** a data point based on the majority class of its **"k Nearest Neighbors"** (in feature space)

⇒ Method:

- → supervised (requires labeled training data)
- → non-parametric (no assumption on decision boundary)
- → non-linear decision boundary can be recovered

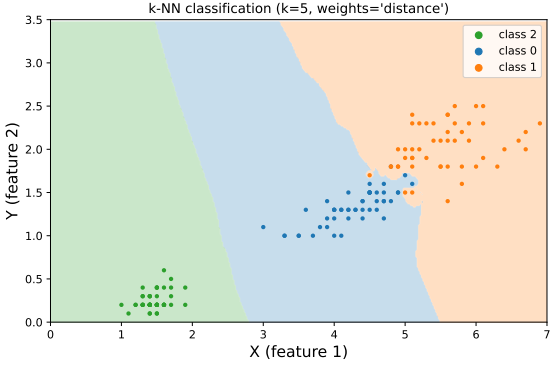
⇒ Steps:

1. *take random data points in the training dataset*
2. *for a sample find the k (e.g. 5) closest data points in the training dataset (k is a hyperparameter)*
3. *look at the neighbor labels, return/assign the mode*



1. kNN classification

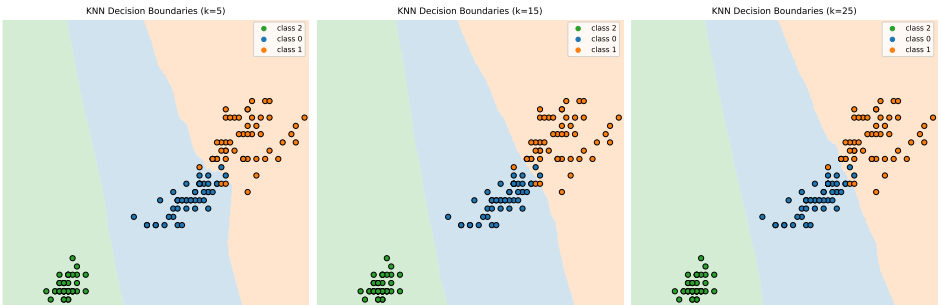
EX: estimate classification boundary using the kNN algorithm



1. kNN classification

EX: estimate classification boundary using the kNN algorithm

$\Rightarrow$ *influence of the hyperparameter k on the decision boundary (controls boundary smoothness):*

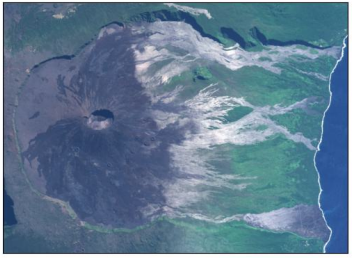


1. kNN classification

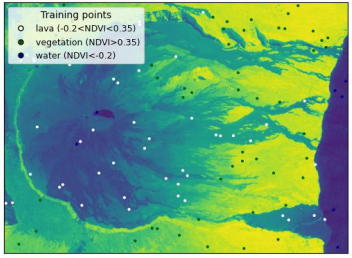
EX: estimate pixel class $\hat{y} \in \{0, 1, 2\} \Rightarrow$ *vegetation*, *water*, or *other*

$\Rightarrow$ supervised classification in **feature space** (e.g., bands, NDVI)
$\Rightarrow$ requires **labeled** samples (training pixels)
$\Rightarrow$ produces a **classified map** (pixel labels)

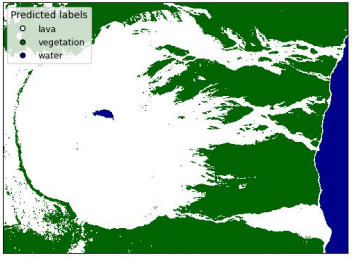
RGB image



NDVI image with training data points



Predicted map

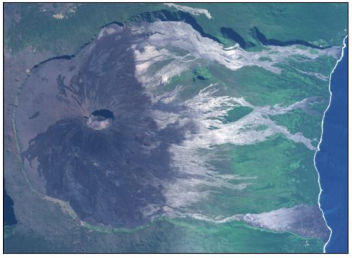


1. kNN classification

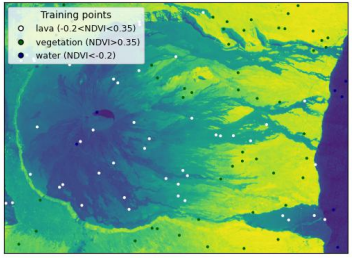
EX: estimate pixel class $\hat{y} \in \{0, 1, 2\} \Rightarrow$ *vegetation*, *water*, or *other*

$\Rightarrow$ supervised classification in **feature space** (e.g., bands, NDVI)
$\Rightarrow$ requires **labeled** samples (training pixels)
$\Rightarrow$ produces a **classified map** (pixel labels)

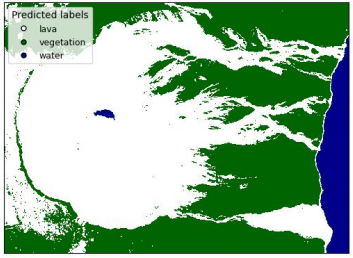
RGB image



NDVI image with training data points



Predicted map



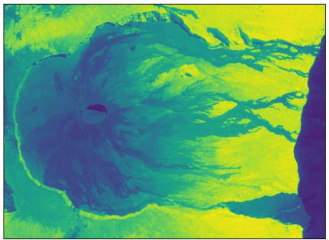
→ Pros: simple, easy to understand, works well with small datasets
→ Cons: slow for large datasets, sensitive to choice of distance metric and the value of k

Note: kNN $\neq$ k-means

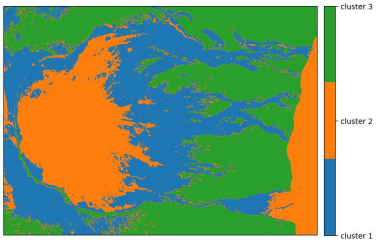
$\Rightarrow$ do not confuse "kNN" with "k-means":

- **kNN** = supervised *classification* algorithm (needs labels)
  $\Rightarrow$ *assigns class based on nearest neighbors*
- **k-means** = unsupervised *clustering/segmentation* algorithm (no labels)
  $\Rightarrow$ *groups pixels by similarity*

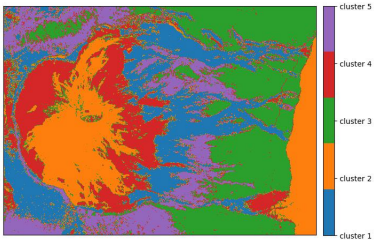
NDVI image



k=3

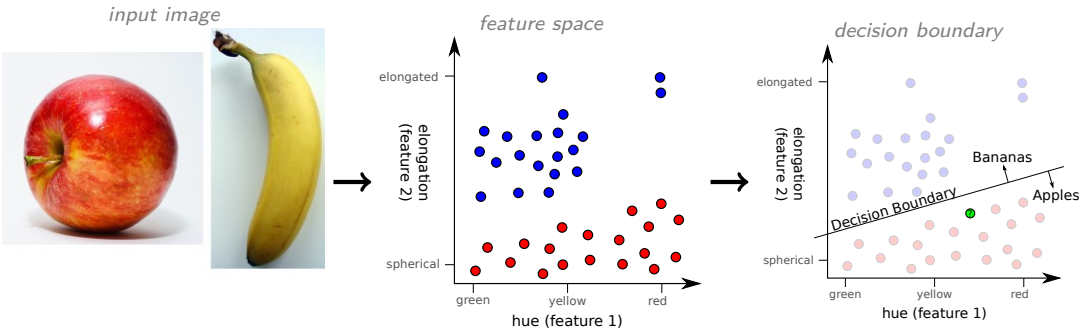


k=5



2. Support Vector Machine (SVM)

⇒ *Recall our toy example*: *classify fruit images into either bananas or apples*



⇒ *how can the linear decision boundary be learned?*

2. Support Vector Machine (SVM)

⇒ algorithm which classifies data based on linear decision boundary

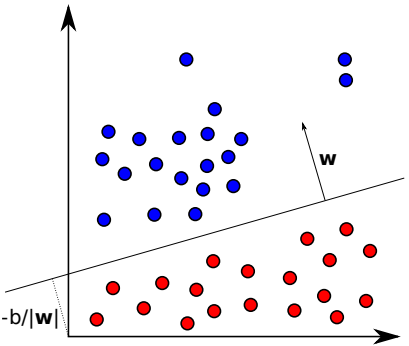
NB: the basic linear classifier is a binary classifier, similar to logistic regression, but non-probabilistic

NB: in an N-dimensional feature space, the decision boundary is a ***hyperplane***

⇒ **linear classifier** (a.k.a. "perceptron"):

$$\hat{y} = \text{sign}(\mathbf{w}^T x + \mathbf{b})$$

- $\hat{y} \in \{-1, 1\}$: predicted class → *banana or apple*
- $x \in \mathbb{R}^2$: feature vector → *hue, elongation*
- $\mathbf{w} \in \mathbb{R}^2$: **weight vector** → needs to be learned
- $\mathbf{b} \in \mathbb{R}$: **bias** → needs to be learned
- *sign*: sign function returning the sign of a real number



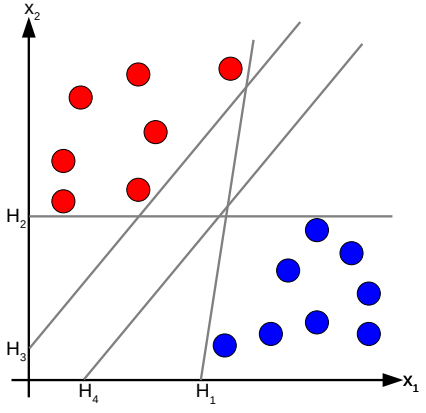
2. Support Vector Machine (SVM)

⇒ **perceptron**: $\boxed{\hat{y} = \text{sign}(\mathbf{w}^T x + \mathbf{b})}$

⇒ problem: multiple "good" boundaries can be found
⇒ need to find the *optimal hyperplane*

⇒ learning rule: several algorithms exist

- **perceptron algorithm** (Rosenblatt, 1958)
  → *next week*
- **SVM** (Vapnik & Cortes, 1995)
  ⇒ boundary with maximal margins
  ⇒ perceptron of maximal stability to new inputs



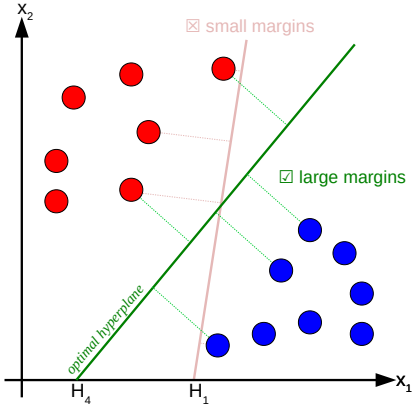
2. Support Vector Machine (SVM)

⇒ **perceptron**: $\hat{y} = \text{sign}(\mathbf{w}^T x + \mathbf{b})$

⇒ problem: multiple "good" boundaries can be found
⇒ need to find the *optimal hyperplane*

⇒ learning rule: several algorithms exist

- **perceptron algorithm** (Rosenblatt, 1958)
  → *next week*
- **SVM** (Vapnik & Cortes, 1995)
  ⇒ boundary with maximal margins
  ⇒ perceptron of maximal stability to new inputs



2. Support Vector Machine (SVM)

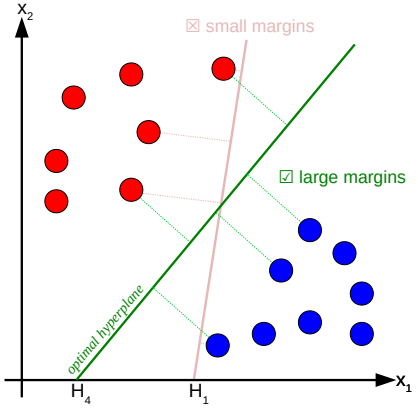
⇒ **perceptron**: $\hat{y} = \text{sign}(\mathbf{w}^T x + \mathbf{b})$

⇒ problem: multiple "good" boundaries can be found
⇒ need to find the *optimal hyperplane*

⇒ learning rule: several algorithms exist

- **perceptron algorithm** (Rosenblatt, 1958)
  → *next week*
- **SVM** (Vapnik & Cortes, 1995)
  ⇒ boundary with maximal margins
  ⇒ perceptron of maximal stability to new inputs

⇒ the **Support Vector Machine** (SVM) algorithm will find the *optimal hyperplane* and learn the best *weights* **w** and *bias* **b** to classify the data



2. Support Vector Machine (SVM)

⇒ **perceptron**: $\hat{y} = \text{sign}(\mathbf{w}^T x + \mathbf{b})$

⇒ definitions:

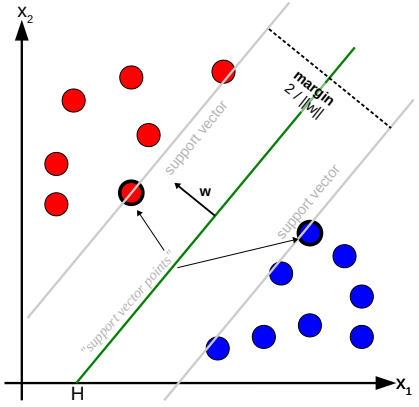
- support vector points = points closest to the hyperplane
  (only these points are contributing to the result, other points are not)
- margin = distance between hyperplane & support vector points
  $= \frac{2}{||w||}$

⇒ maximize margin:

$$\max_w \frac{2}{||w||}, \text{ subject to } \begin{cases} w^T x_i + b \geq 1 & \text{if } y_i = +1 \\ w^T x_i + b \leq 1 & \text{if } y_i = -1 \end{cases}$$

which is equivalent to:

$$\min_w ||w||^2, \text{ subject to } y_i(w^T x_i - b) \geq 1$$



2. Support Vector Machine (SVM)

⇒ **perceptron**: $\hat{y} = \text{sign}(\mathbf{w}^T x + \mathbf{b})$

⇒ definitions:

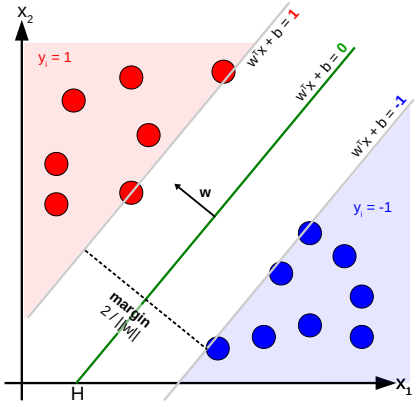
- support vector points = points closest to the hyperplane
  (only these points are contributing to the result, other points are not)
- margin = distance between hyperplane & support vector points
  $= \frac{2}{||w||}$

⇒ maximize margin:

$$\max_w \frac{2}{||w||}, \text{ subject to } \begin{cases} w^T x_i + b \geq 1 & \text{if } y_i = +1 \\ w^T x_i + b \leq 1 & \text{if } y_i = -1 \end{cases}$$

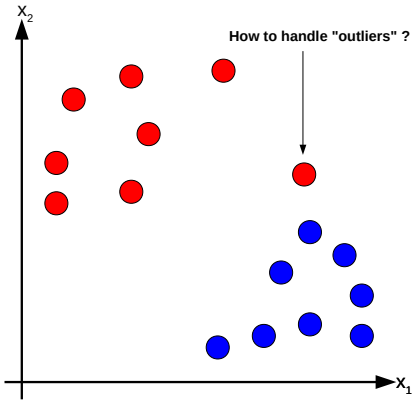
which is equivalent to:

$$\min_w ||w||^2, \text{ subject to } y_i(w^T x_i - b) \geq 1$$



2. Support Vector Machine (SVM)

⇒ **How can outliers be handled?**



2. Support Vector Machine (SVM)

⇒ **How can outliers be handled?**

⇒ is a hard-margin with 100% accuracy good?

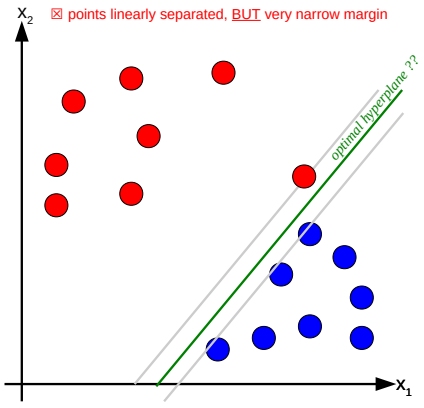
⇒ no, allow small errors (**soft-margin**) to favour overall better model

⇒ tolerate margin violation & favour large margin boundaries

⇒ optimization becomes:

$$\min_{w,\xi_i} ||w||^2 + C \sum_{i}^{N} \xi_i, \text{ subject to } y_i(w^T x_i - b) \geq 1 - \xi_i$$

where:
- C regularization parameter
  - small C ⇒ constraints easily ignored, large margin
  - large C ⇒ towards hard-margin SVM
- ξ_i slack variable for each data point



2. Support Vector Machine (SVM)

⇒ **How can outliers be handled?**

⇒ is a hard-margin with 100% accuracy good?

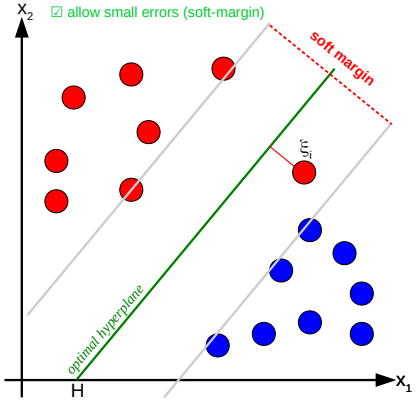
⇒ no, allow small errors (**soft-margin**) to favour overall better model

⇒ tolerate margin violation & favour large margin boundaries

⇒ optimization becomes:

$$\min_{w,\xi_i} ||w||^2 + C \sum_{i}^{N} \xi_i, \text{ subject to } y_i(w^T x_i - b) \geq 1 - \xi_i$$

where:
- C regularization parameter
  - small C ⇒ constraints easily ignored, large margin
  - large C ⇒ towards hard-margin SVM
- ξ_i slack variable for each data point



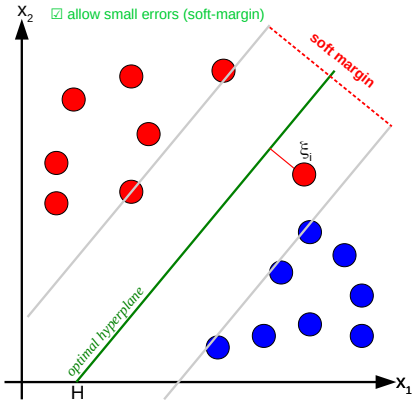
2. Support Vector Machine (SVM)

⇒ **How can outliers be handled?**

- ⇒ is a hard-margin with 100% accuracy good?
- ⇒ no, allow small errors (**soft-margin**) to favour overall better model
- ⇒ tolerate margin violation & favour large margin boundaries
- ⇒ optimization becomes:

$$\min_{w,\xi_i} ||w||^2 + C \sum_{i}^{N} \xi_i, \text{ subject to } y_i(w^T x_i - b) \geq 1 - \xi_i$$

where:
- C regularization parameter
  - small C ⇒ constraints easily ignored, large margin
  - large C ⇒ towards hard-margin SVM
- ξ_i slack variable for each data point



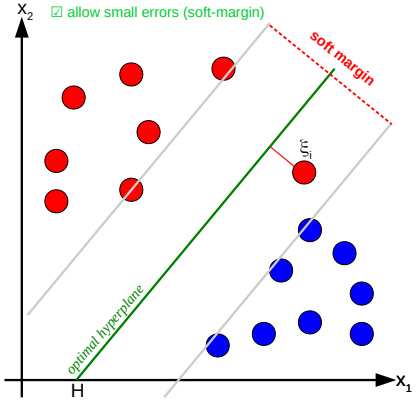
2. Support Vector Machine (SVM)

⇒ **How can outliers be handled?**

- ⇒ is a hard-margin with 100% accuracy good?
- ⇒ no, allow small errors (**soft-margin**) to favour overall better model
- ⇒ tolerate margin violation & favour large margin boundaries
- ⇒ optimization becomes:

$$\min_{w,\xi_i} ||w||^2 + C \sum_{i}^{N} \xi_i, \text{ subject to } y_i(w^T x_i - b) \geq 1 - \xi_i$$

where:
- C regularization parameter
  - small C ⇒ constraints easily ignored, large margin
  - large C ⇒ towards hard-margin SVM
- ξ_i slack variable for each data point



Side note: *reformulating optimization in terms of* ***regularization*** *and* ***loss function*** *(anticipating DL lectures)*

Learning an SVM has been formulated as a constrained optimization problem over w and ξ:

$$\min_{w,\xi_i} ||w||^2 + C \sum_{i}^{N} \xi_i \qquad \text{subject to:} \quad y_i(w^T x_i - b) \geq 1 - \xi_i$$

The constraint $y_i(w^T x_i - b) \geq 1 - \xi_i$ can be written more concisely as: $y_i f(x_i) \geq 1 - \xi_i$

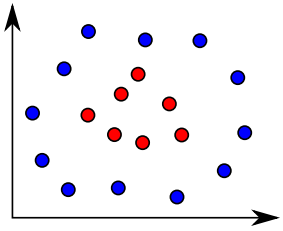
Together with $\xi_i > 0$, it is equivalent to: $\xi_i = max(0, 1 - y_i f(x_i))$

Hence the learning problem is equivalent to the unconstrained optimization problem over w:

$$\min_{w} \underbrace{||w||^2}_{\text{regularization}} + C \sum_{i}^{N} \underbrace{max(0,\ 1 - y_i f(x_i))}_{\text{loss function (Hinge loss)}}$$

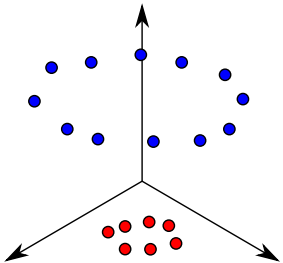
2. Support Vector Machine (SVM)

- **What if the features x_i are not linearly separable?**



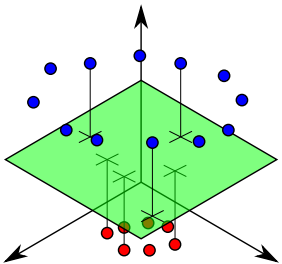
2. Support Vector Machine (SVM)

- **What if the features x_i are not linearly separable**?
  $\Rightarrow$ compute new features $x_i \mapsto \phi(x)$
  $\phi(x)$ is a **feature map**, mapping x to $\phi(x)$ where data is separable



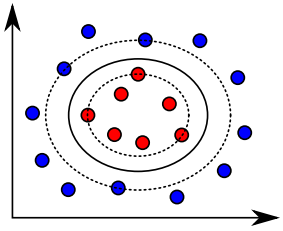
2. Support Vector Machine (SVM)

- **What if the features x_i are not linearly separable**?
  $\Rightarrow$ compute new features $x_i \mapsto \phi(x)$
  $\phi(x)$ is a **feature map**, mapping x to $\phi(x)$ where data is separable
  $\Rightarrow$ solve for $\mathbf{w}$ in high dimensional feature space



2. Support Vector Machine (SVM)

- **What if the features x_i are not linearly separable**?
  $\Rightarrow$ compute new features $x_i \mapsto \phi(x)$
  $\phi(x)$ is a **feature map**, mapping x to $\phi(x)$ where data is separable
  $\Rightarrow$ solve for $\mathbf{w}$ in high dimensional feature space
  $\Rightarrow$ data not lineary-seperable in original feature space become separable



Kernel trick

The Representer Theorem states that the solution **w** can be written as a linear combination of the training data:

$$w = \sum_{j=1}^{N} \alpha_j y_j x$$

Kernel trick

The Representer Theorem states that the solution $\mathbf{w}$ can be written as a linear combination of the training data:

$$w = \sum_{j=1}^{N} \alpha_j y_j x$$

The linear classifier can therefore be reformulated as:

$$\begin{aligned} f(x) &= w^T x + b \\ &= \sum_{i}^{N} \alpha_i y_i (x_i{}^T x) + b \end{aligned}$$

Kernel trick

The Representer Theorem states that the solution $\mathbf{w}$ can be written as a linear combination of the training data:

$$w = \sum_{j=1}^{N} \alpha_j y_j x$$

The linear classifier can therefore be reformulated as:

$$\begin{aligned} f(x) &= w^T x + b \\ &= \sum_{i}^{N} \alpha_i y_i (x_i^T x) + b \end{aligned}$$

NB: this reformulation seems to have the disadvantage of a kNN classifier, i.e. requires the training data points x_i. However, many of the $\alpha_i = 0$: the ones that are non-zero define the support vector points x_i

Kernel trick

The Representer Theorem states that the solution $\mathbf{w}$ can be written as a linear combination of the training data:

$$w = \sum_{j=1}^{N} \alpha_j y_j x$$

The linear classifier can therefore be reformulated as:

$$\begin{aligned} f(x) &= w^T x + b \\ &= \sum_{i}^{N} \alpha_i y_i (x_i^T x) + b \end{aligned}$$

NB: this reformulation seems to have the disadvantage of a kNN classifier, i.e. requires the training data points x_i. However, many of the $\alpha_i = 0$: the ones that are non-zero define the support vector points x_i

Using the feature map $\phi(x)$, it can be reformulated as:

$$\begin{aligned} f(x) &= \sum_{i}^{N} \alpha_i y_i (\phi(x_i)^T \phi(x)) + b \\ &= \sum_{i}^{N} \alpha_i y_i k(x_i, x) + b \end{aligned}$$

where $k(x_i, x)$ is known as a **Kernel**

Kernel trick

- Classifier can be learnt and applied without explicitly computing $\phi(x)$
- All that is required is the kernel $k(x, x')$
- Multiple kernels exist:
  - linear kernels: $k(x, x') = x^T x'$
    $\rightarrow$ *very fast and easy to train, but very simple*
  - polynomial kernels: $k(x, x') = (1 + x^T x')^d$
    $\rightarrow$ *contains all polynomial terms up to degree d*
  - gaussian kernels: $k(x, x') = exp(-||x - x'||^2/2\sigma^2)$ *(RBF kernel)*
    $\rightarrow$ *kernel very powerful and most often used*

Kernel trick

- Classifier can be learnt and applied without explicitly computing $\phi(x)$
- All that is required is the kernel $k(x, x')$
- Multiple kernels exist:
  - linear kernels: $k(x, x') = x^T x'$
    $\rightarrow$ *very fast and easy to train, but very simple*
  - polynomial kernels: $k(x, x') = (1 + x^T x')^d$
    $\rightarrow$ *contains all polynomial terms up to degree d*
  - gaussian kernels: $k(x, x') = exp(-||x - x'||^2/2\sigma^2)$ *(RBF kernel)*
    $\rightarrow$ *kernel very powerful and most often used*

Kernel trick

- Classifier can be learnt and applied without explicitly computing $\phi(x)$
- All that is required is the kernel $k(x, x')$
- Multiple kernels exist:
  - linear kernels: $k(x, x') = x^T x'$
    $\rightarrow$ *very fast and easy to train, but very simple*
  - polynomial kernels: $k(x, x') = (1 + x^T x')^d$
    $\rightarrow$ *contains all polynomial terms up to degree d*
  - gaussian kernels: $k(x, x') = exp(-||x - x'||^2/2\sigma^2)$ *(RBF kernel)*
    $\rightarrow$ *kernel very powerful and most often used*

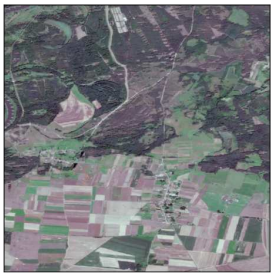
2. Support Vector Machine (SVM)

EX: use PCA + SVM to *classify land-use in satellite images (Sentinel-2)*

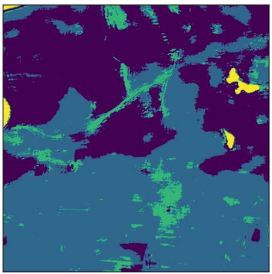
→ *refer to CV4GS 2024 lecture for details on PCA*

PCA dimensionality reduction

train SVM & apply

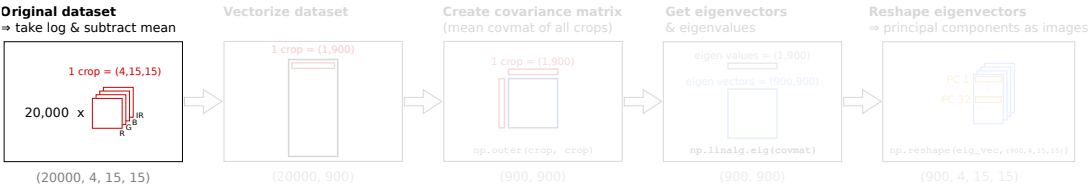


land-use classification



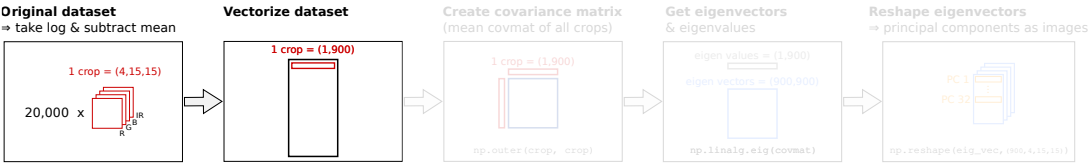
Non-probabilistic classification

Part 1: apply PCA on satellite image crops



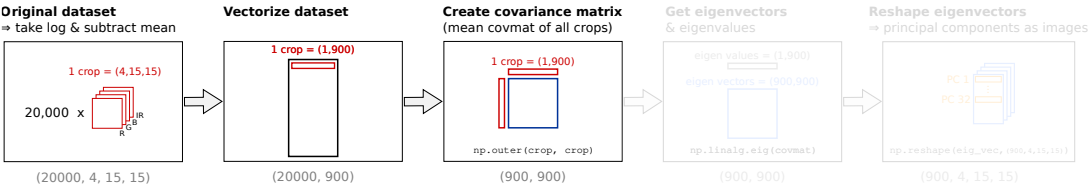
Non-probabilistic classification

Part 1: apply PCA on satellite image crops



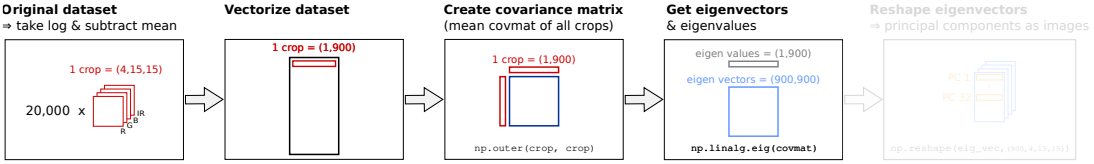
Non-probabilistic classification

Part 1: apply PCA on satellite image crops



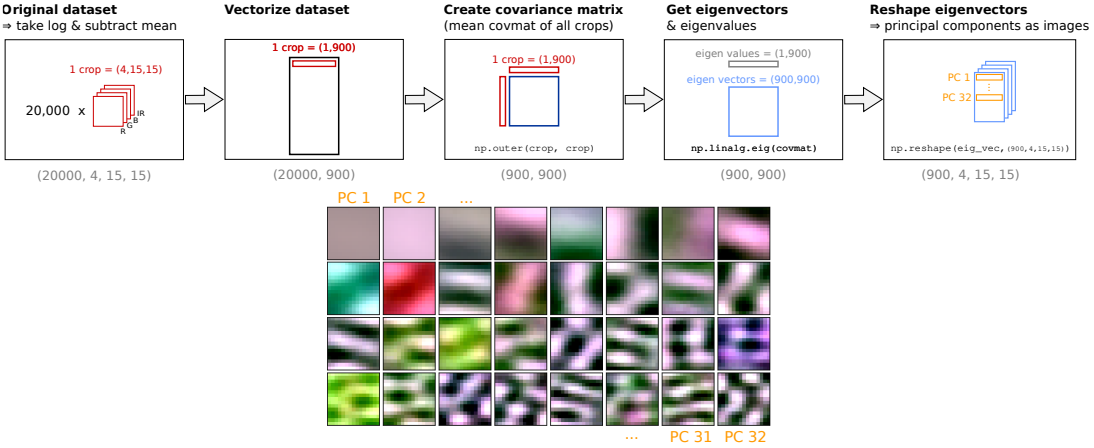
Non-probabilistic classification

Part 1: apply PCA on satellite image crops



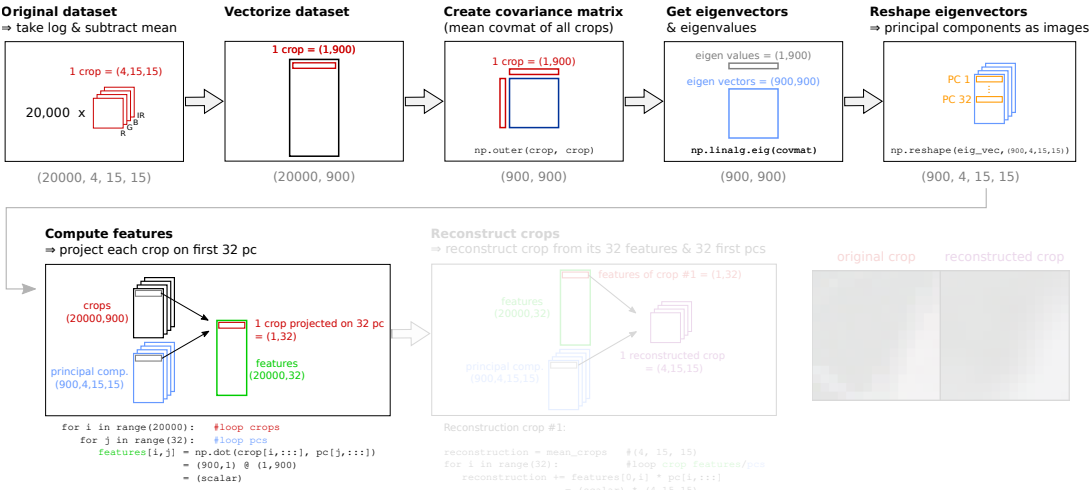
Non-probabilistic classification

Part 1: apply PCA on satellite image crops



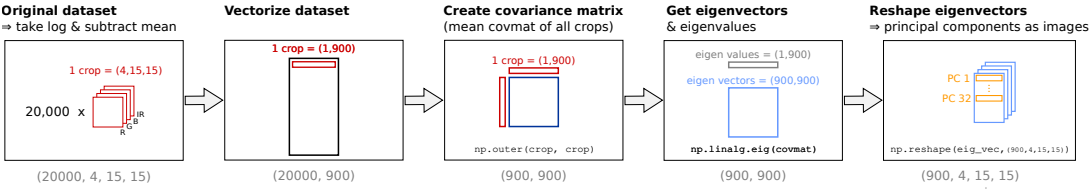
Non-probabilistic classification

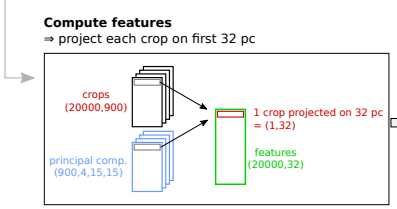
Part 1: apply PCA on satellite image crops

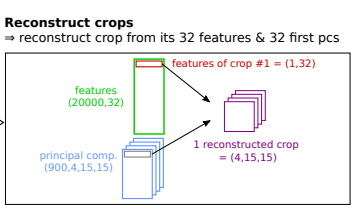


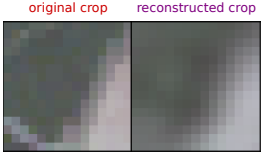
Non-probabilistic classification

Part 1: apply PCA on satellite image crops



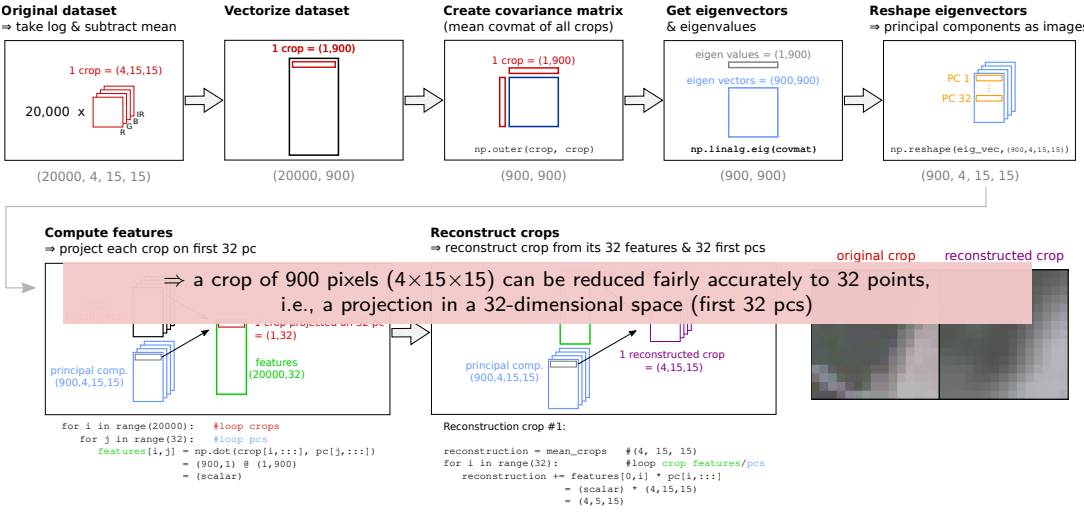






Non-probabilistic classification

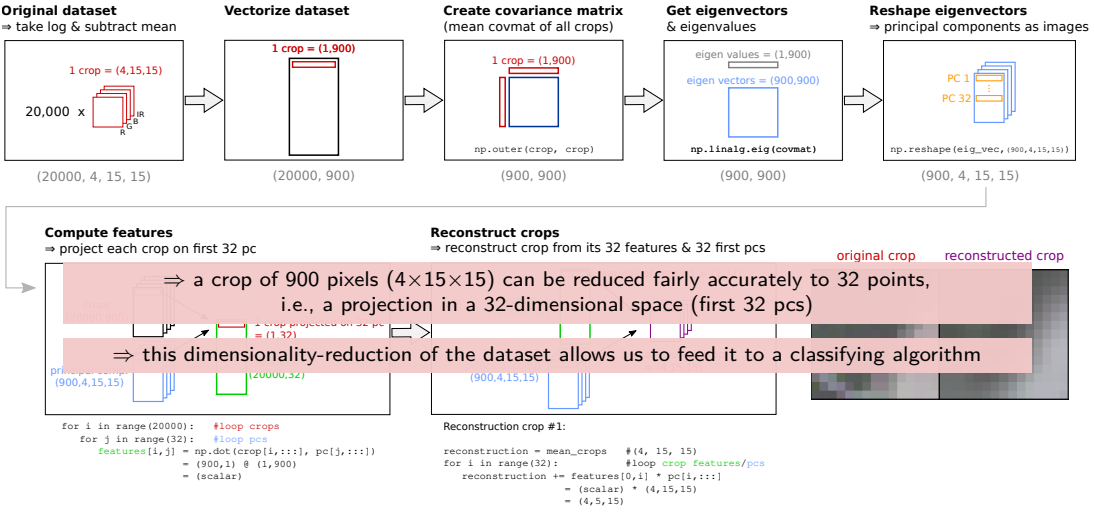
Part 1: apply PCA on satellite image crops



⇒ a crop of 900 pixels (4×15×15) can be reduced fairly accurately to 32 points,
i.e., a projection in a 32-dimensional space (first 32 pcs)

Non-probabilistic classification

Part 1: apply PCA on satellite image crops



Non-probabilistic classification

Part 2: train SVM on principal components and apply to classify full image

PCA dimensionality reduction

land-use classification