Machine Learning

Classification

Alberto Maria Metelli - Francesco Trovò

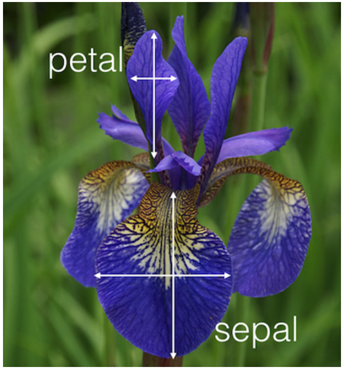
Politecnico di Milano

Outline

Dataset

Consider the **Iris Dataset**
(`https://en.wikipedia.org/wiki/Iris_flower_data_set`):

- Sepal length
- Sepal width
- Petal length
- Petal width
- Species (Iris setosa, Iris virginica e Iris versicolor)



$N = 150$ total samples (50 per species)

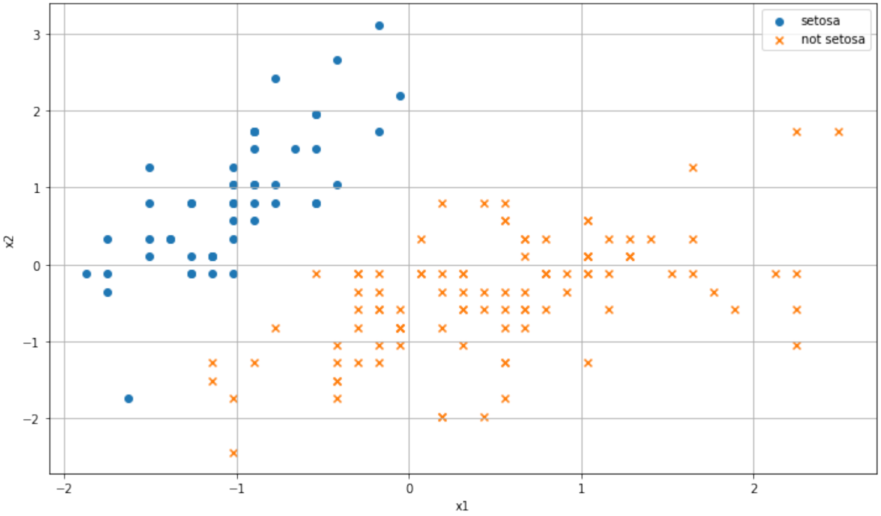
Scientific Questions

- Can we extract some information from the data?
- What can we infer from them?
- Can we provide predictions on some of the quantities on newly seen data?
- Can we predict the **petal width** (*target*) of a specific kind of Iris setosa by using the petal length, sepal length and width (*variables*)?
- In this case, the target is *continuous* ($t_n \in \mathbb{R}$) $\rightarrow$ **Regression**

A Classification Problem

- Can we predict the **kind of Iris** (*target*) using petal/sepal length and width (*variables*)?
- In this case, the target are *discrete* and *non-metric* ($t_n \in \{\text{setosa, virginica, versicolor}\}$) $\rightarrow$ **classification**
    - Targets are called **classes**
- Initially, we solve the problem of discriminating between setosa and non-setosa flowers
    - We have just two classes: $t_n \in \{\text{non-setosa, setosa}\}$ or $t_n \in \{0, 1\}$ $\rightarrow$ **binary classification**
    - As input $\mathbf{x}_n$ we choose sepal length and width (for visualization purposes)
- Then, we will consider the original problem
    - We have three classes: $t_n \in \{\text{setosa, virginica, versicolor}\}$ $\rightarrow$ **multi-class classification**

Iris Dataset - Binary Classification Problem



Different Approaches for Classification

Three possible approaches:

- **Discriminant function approach**
  - the model is a function that maps inputs to classes $f(\mathbf{x}) = C_k \in \{C_1, \dots, C_K\}$
  - fit model to data
- **Probabilistic discriminative approach**
  - the model is a conditional probability distribution $P(C_k|\mathbf{x}) \in [0, 1]$
  - fit model to data
- **Probabilistic generative approach**
  - the model model is the likelihood $P(\mathbf{x}|C_k) \in [0, 1]$ and the prior $P(C_k) \in [0, 1]$
  - fit model to data
  - make inference using posterior $P(C_k|\mathbf{x}) = \dfrac{P(C_k)P(\mathbf{x}|C_k)}{P(\mathbf{x})}$
  - can generate new samples from the joint $P(C_k, \mathbf{x}) = P(\mathbf{x}|C_k)P(C_k)$

Possible Solutions

- Linear classification
    - Perceptron
    - Logistic regression
- Naïve Bayes
- K-nearest neighbor

Preliminary Operations

As usual before solving the problem we need to perform some preliminary operations:

- Load the data
- Consistency checks
- Select and normalize the input
- **Shuffle the data** (`shuffle()`)
- Generate the output ($t_n \in \{0, 1\}$ or $t_n \in \{-1, 1\}$)
- Explore the selected data (`scatter`)

Perceptron

Discriminant function approach

- Hypothesis space:

$$y(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x}) = \text{sign}(w_0 + x_1 w_1 + x_2 w_2) \qquad \text{sign}(z) = \begin{cases} -1 & \text{if } z < 0 \\ +1 & \text{if otherwise} \end{cases}$$

- Loss function: distance of misclassified points in $\{(\mathbf{x}_n, t_n)\}_{n=1}^N$ with $t_n \in \{-1, 1\}$

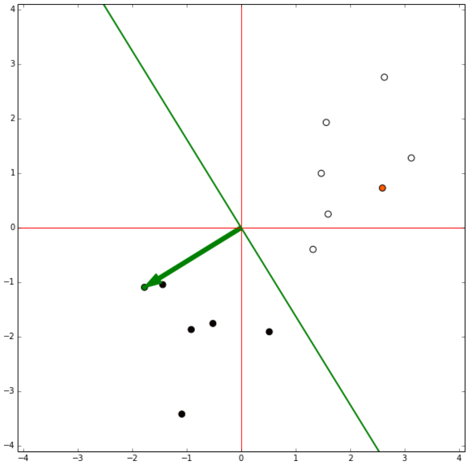
$$L_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^T \mathbf{x}_n t_n \qquad \text{where} \qquad \mathcal{M} = \{n \in \{1, \dots, N\} : t_n \neq y(\mathbf{x}_n)\}$$

- Optimization method: online gradient descent

Implementation in Python via:

- Scikit-learn (`Perceptron`)
- By hand

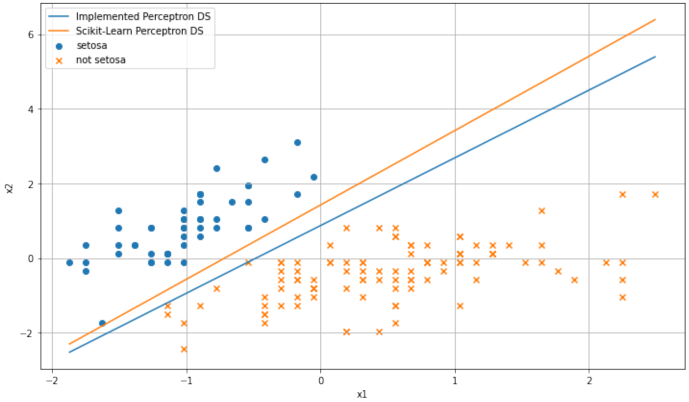
Learning Example



Plotting the results

To visualize the **separating hyperplane** or **decision boundary** (line) we need to plot:

$$\text{sign}(\mathbf{w}^\top \mathbf{x}) = 0 \quad \rightarrow \quad \text{sign}(w_0 + x_1 w_1 + x_2 w_2) = 0 \quad \rightarrow \quad x_2 = -\frac{w_1 x_1 + w_0}{w_2}$$



Evaluating the Results

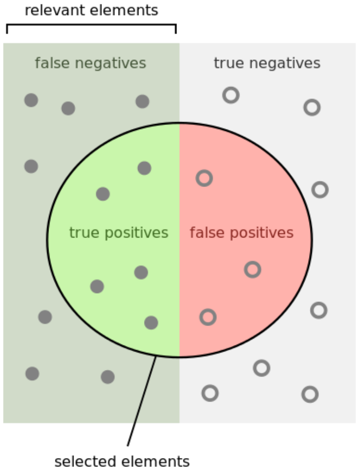
To evaluate the performance of a classifier, we can to compute the **confusion matrix** which tells us the number of points which have been correctly classified and those which have been misclassified

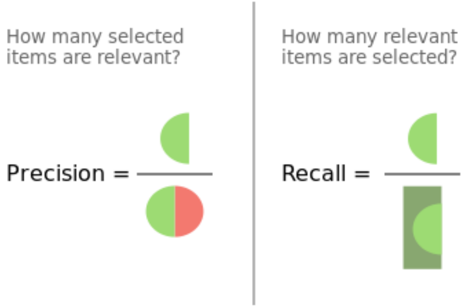
	Actual Class: 1	Actual Class: 0
Predicted Class: 1	tp	fp
Predicted Class: 0	fn	tn

Evaluating the Results

- **Accuracy**: fraction of the samples correctly classified in the dataset
$$Acc = \frac{tp + tn}{N}$$
- **Precision**: fraction of samples correctly classified in the positive class among the ones classified in the positive class
$$Pre = \frac{tp}{tp + fp}$$
- **Recall**: fraction of samples correctly classified in the positive class among the ones belonging to the positive class
$$Rec = \frac{tp}{tp + fn}$$
- **F1 score**: harmonic mean of precision and recall
$$F1 = \frac{2 \cdot Pre \cdot Rec}{Pre + Rec}$$

Precision and Recall




How many selected items are relevant?
Precision =
How many relevant items are selected?
Recall =

Evaluating the Results

Remember that:

- The higher these performance measures the better the algorithm is performing
- These performance measures are **not symmetric**, but depend on the class we selected as positive
- Depending on the **application** one might switch the classes to have measures which better evaluate the predictive power of the classifier

Logistic Regression

Probabilistic discriminative approach

- Hypothesis space:

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \sigma(w_0 + x_1 w_1 + x_2 w_2) \qquad \sigma(z) = \frac{1}{1+e^{-z}}$$

- Loss function: log likelihood of $\{(\mathbf{x}_n, t_n)\}_{n=1}^N$ with $t_n \in \{0,1\}$

$$L_P(\mathbf{w}) = p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \sum_{n=1}^{N} t_n \ln y(\mathbf{x}_n) + (1-t_n)\ln(1-y(\mathbf{x}_n))$$

- Optimization method: online gradient descent

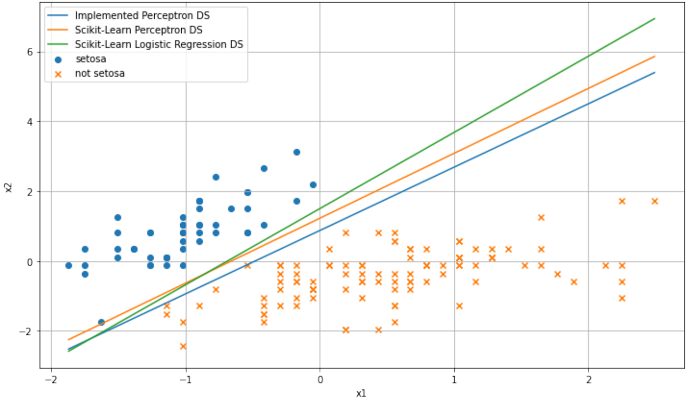
Implementation in Python via:

- Scikit-learn (`LogisticRegression`)

Plotting the results

To visualize the **separating hyperplane** or **decision boundary** (line) we need to plot:

$$\sigma(\mathbf{w}^\top \mathbf{x}) = 1/2 \quad \to \quad \sigma(w_0 + x_1 w_1 + x_2 w_2) = 1/2 \quad \to \quad x_2 = -\frac{w_1 x_1 + w_0}{w_2}$$



Logit

Consider the function:

$$\text{logit}(y) = \log\left(\frac{y}{1-y}\right)$$

we can apply it to the output of logistic regression:

$$\text{logit}(y(\mathbf{x})) = \mathbf{w}^T\mathbf{x} = w_0 + x_1 w_1 + x_2 w_2$$

- same characterization as linear regression
- we can perform **hypothesis testing** on the significance of the parameters which linearly influence the *log-odds* of the output

Naïve Bayes (NB)

Naïve assumption: given the class C_k each input is conditionally independent from each other

$$
\begin{aligned}
p(C_k|\mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x}|C_k)}{p(\mathbf{x})} &\propto p(x_1, \dots, x_M, C_k) \\
&= p(x_1|x_2, \dots, x_M, C_k)p(x_2, \dots, x_M, C_k) \\
&= p(x_1|x_2, \dots, x_M, C_k)p(x_2|x_3, \dots, x_M, C_k)p(x_3, \dots, x_n, C_k) \\
&= p(x_1|x_2, \dots, x_M, C_k) \dots p(x_M|C_k)p(C_k) \\
&= p(x_1|C_k) \dots p(x_M|C_k)p(C_k) = p(C_k) \prod_{j=1}^{M} p(x_j|C_k)
\end{aligned}
$$

Decision function: given a prior $p(C_k)$, maximize the Maximum A Posteriori (MAP) probability:

$$
y(\mathbf{x}) = \arg\max_k p(C_k) \prod_{j=1}^{M} p(x_j|C_k)
$$

Naïve Bayes (NB)

Probabilistic generative approach

- Hypothesis space:

$$y(\mathbf{x}) = \arg\max_k p(C_k) \prod_{j=1}^{M} p(x_j|C_k)$$

- Loss function: log likelihood for fitting both the priors $p(C_k)$ and the likelihoods $p(x_j|C_k)$
- Optimization method: maximum likelihood estimation (MLE)

Example: Multinomial and Gaussian Priors

In our classification problem, we select:

- Prior: $p(C_k)$ multinomial distribution with parameters $(p_1, \dots, p_k)$
- Likelihood: $p(x_j|C_k) \sim \mathcal{N}(\mu_{jk}, \sigma^2_{jk})$, i.e., a normal distribution for each feature x_j and each class C_k

Depending on the input we might choose different distribution for the features

Implementing Naïve Bayes

- Preimplemented Scikit-learn `GaussianNB`:
  - Prior: multinomial distribution
  - Likelihood: Gaussian distributions
- By hand:
  - Estimate the prior: $\hat{p}(C_k) = \frac{\sum_{i=1}^{N} I\{\mathbf{x}_n \in C_k\}}{N}$
  - Estimate the MLE parameters: $p(x_j|C_k) = \mathcal{N}(x_j; \hat{\mu}_{jk}, \hat{\sigma}^2_{jk})$, where we compute $\hat{\mu}_{jk}$ and $\hat{\sigma}^2_{jk}$ maximizing the likelihood
  - Compute $p(C_k) \prod_{j=1}^{M} p(x_j|C_k)$ for each class C_k and choose the maximum one

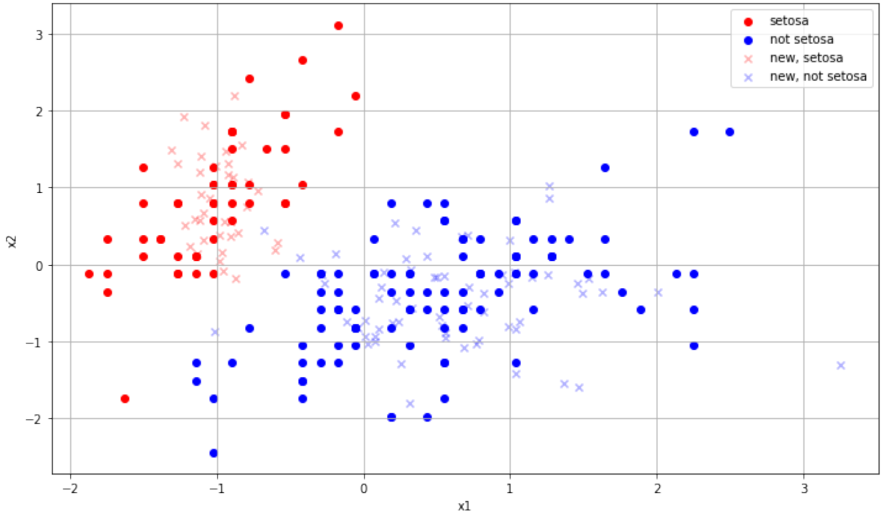
Notice that the Naïve Bayes is **not** a Bayesian method
The priors we compute are estimated from data, and not updated using likelihoods

Generative Method

Thanks to the generative abilities of the Naïve Bayes classifier we are able to generate dataset which resembles the original one:

- Select a class $C_{\hat{k}}$ according to prior multinomial distribution with parameters $\hat{p}(C_1), \dots, \hat{p}(C_K)$
- For each feature j, draw a sample x_j from $\mathcal{N}(\hat{\mu}_{j\hat{k}}, \hat{\sigma}^2_{j\hat{k}})$
- Repeat every time you want a new sample

Naïve Bayes Sample Generation



1-Nearest Neighbor (1NN)

Discriminative function approach

- **Idea**: look at the nearby points to predict the target of a new point
- Given a dataset $\{(\mathbf{x}_n, t_n)\}_{i=1}^N$ and a new data point $\mathbf{x}_q$, we predict the target as:

$$i_q \in \arg \min_{n \in \{1,\ldots,N\}} \|\mathbf{x}_q - \mathbf{x}_n\|_2 \quad \rightarrow \quad \text{Predicted target: } \hat{t}_q = t_{i_q}$$

- Works transparently for both regression and classification
- No need for explicit training. Training is querying the dataset

Design choices

- Which distance to choose? Euclidean $\|\cdot\|_2$, Manhattan Distance $\|\cdot\|_1, \dots$
- How many neighbors? K**-Nearest Neighbor**
- If $K > 1$, how to combine the targets?

$$\mathcal{N}_K(\mathbf{x}_q) = \{i \in \{1, \dots, N\} : \mathbf{x}_i \text{ are the } K \text{ points closest to } \mathbf{x}_q\}$$

  - For classification, predict the **mode class** (+ define a tie braking rule)

$$\hat{t}_q \in \arg\max_{C_k} |\{i \in \mathcal{N}_K(\mathbf{x}_q) : t_i = C_k\}|$$

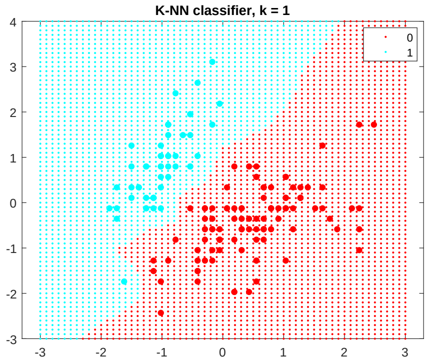
  - For regression, predict the **average target**

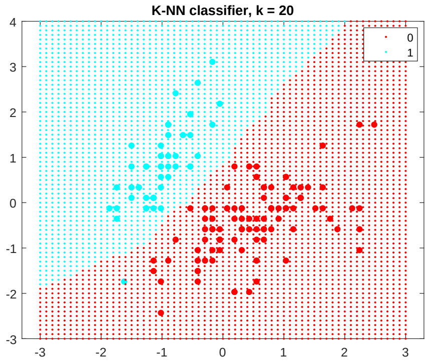
$$\hat{t}_q = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\mathbf{x}_q)} t_i$$

  - Other approaches: provide a probability distribution instead of a class, use weights proportional to the inverse of the distance, ...

Regularizing with KNN

Depending on the number of neighbors K we are introducing strong or mild regularization





Categorization of the Classification Algorithms

Perceptron

- Parametric
- Frequentist
- Discriminative function

Logistic regression

- Parametric
- Frequentist
- Probabilistic discriminative

Naïve Bayes

- Parametric
- Frequentist
- Probabilistic generative

K-Nearest Neigbour

- Non-parametric
- N.A.
- Discriminative function

Multiple Classes

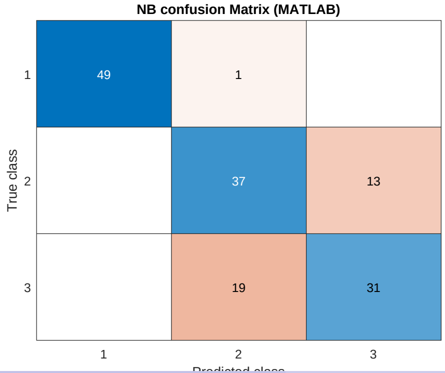
- In the case we have multiple classes we can use the same function, feeding a target with more than two labels
- It will train K different models, one for each class vs. the rest
- The parameter vector is now a matrix W

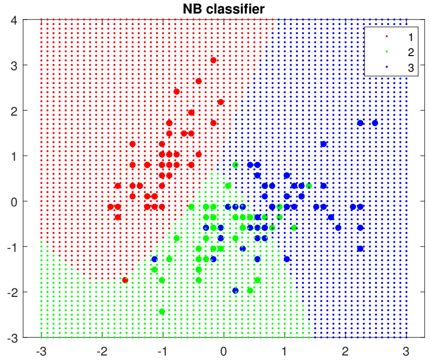
We can display the separating surfaces, but it would be a little more difficult than the case with two classes

Multiple Classes for K-NN and Naïve Bayes

We do not have to change anything to extend these two methods to deal with multiple classes

New definition of the target $y_n \in \{1, 2, 3\}$ in this specific case and estimated prior and likelihood parameters for the three classes


NB confusion Matrix (MATLAB)
True class
Predicted class


NB classifier