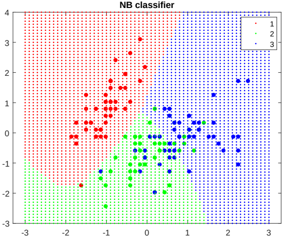
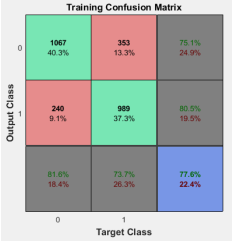
Machine Learning

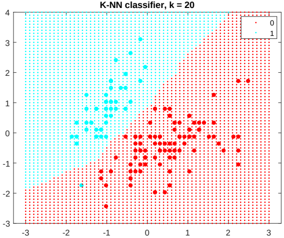
Classification



Alberto Maria Metelli

Credits to Francesco Trovò



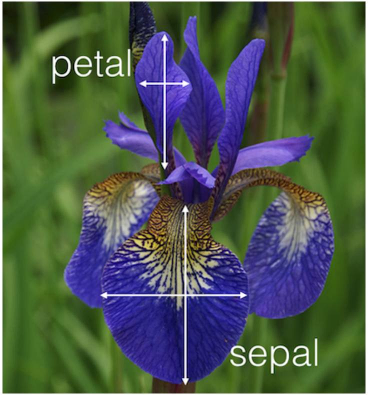


Outline

Dataset

Consider the `iris_dataset`:

- 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* ($y_n \in \mathbb{R}$) $\rightarrow$ **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 *unordered* ($y_n \in \{\text{setosa, virginica, versicolor}\}$) $\rightarrow$ **Classification**
- Initially, we solve the problem of discriminating between setosa and non-setosa flowers
  - We have just two classes: $y_n \in \{\text{non-setosa, setosa}\}$ or $y_n \in \{0, 1\} \rightarrow$ **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: $y_n \in \{\text{setosa, virginica, versicolor}\}$ or $y_n \in \{1, 2, 3\} \rightarrow$ **multi-class classification**

Different Approaches for Classification

Three possible approaches:

- **Discriminant function approach**
  - model 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**
  - model a conditional probability $P(C_k|\mathbf{x})$
  - fit model to data
- **Probabilistic generative approach**
  - model the likelihood $P(\mathbf{x}|C_k)$ and prior $P(C_k)$
  - 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
- Naive Bayes
- K-nearest neighbour

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**
- Generate the output ($y_n \in \{0, 1\}$)
- Explore the selected data (`scatter`)

Perceptron

- Hypothesis space: $y(\mathbf{x}_n) = sgn(\mathbf{w}^T\mathbf{x}_n) = sgn(w_0 + x_{n1}w_1 + x_{n2}w_2)$
- Loss measure: Distance of misclassified points

$$L_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^T\mathbf{x}_n C_n$$

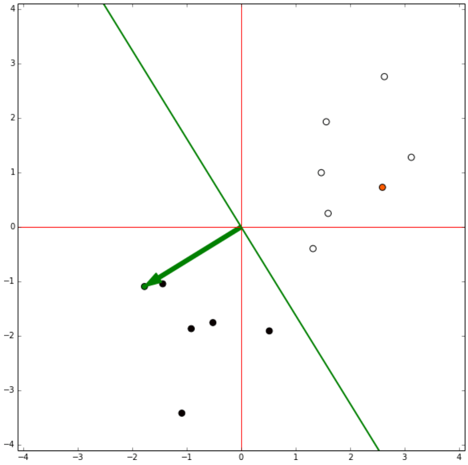
- Optimization method: Online Gradient Descent

where $sgn(\cdot)$ is the sign function

Optimization in Python via:

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

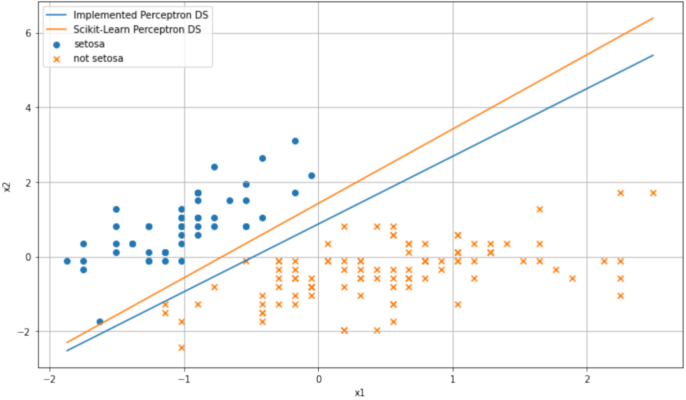
Learning Example



Plotting the results

To visualize the separating hyperplane (line) we need to plot:

$$sgn(w_0 + x_1 w_1 + x_2 w_2) = 0 \quad \rightarrow \quad x_2 = -\frac{w_1 x_1 + w_0}{w_2}$$



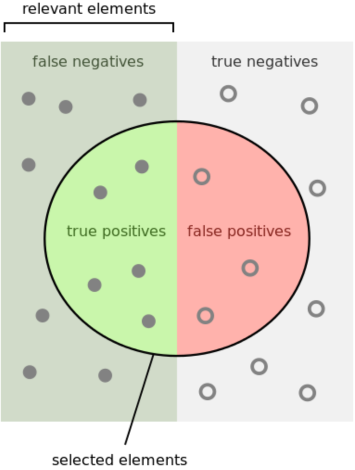
Evaluating the Results

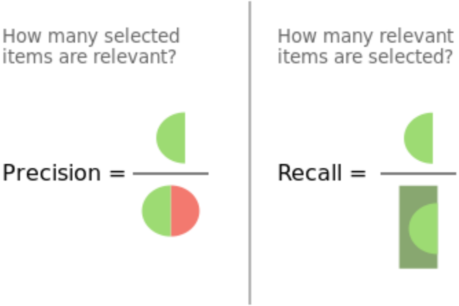
Confusion Matrix:

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

- Accuracy: $Acc = \dfrac{tp + tn}{N}$
- Precision: $Pre = \dfrac{tp}{tp + fp}$
- Recall: $Rec = \dfrac{tp}{tp + fn}$
- F1 score: $F1 = \dfrac{2 \cdot Pre \cdot Rec}{Pre + Rec}$

Precision and Recall





Logistic Regression

- Hypothesis space:

$$y(\mathbf{x}_n) = \sigma(\mathbf{w}^T \mathbf{x}_n) = \sigma(w_0 + x_{n1} w_1 + x_{n2} w_2)$$

- Loss measure: Loglikelihood

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

- Optimization method: Online Gradient Descent

Optimization in Python via:

- Scikit-learn (`LogisticRegression`)

Naive Bayes (NB)

- Hypothesis space: $y_n = y(x_n) = \arg\max_k p(C_k) \prod_{j=1}^{M} p(x_j|C_k)$
- Loss measure: Log Likelihood
- Optimization method: Maximum Likelihood Estimation (MLE)

Derivation of Naive Bayes Decision Boundary

Naive assumption: each input is conditionally (w.r.t. the class) 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: maximize the MAP probability:

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

Specific Method

In our classification problem the classes discriminant function:

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

has:

- Prior: $p(C_k)$ multinomial distribution with parameters $(p_1, \ldots, p_k)$
- Data likelihood: $p(x_j|C_k) \sim \mathcal{N}(\mu_{jk}, \sigma_{jk}^2)$, 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 Naive 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

Generative Method

Thanks to the generative abilities of the Naive 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

Discriminant Approach

Idea: look at the nearby points to classify a new point
Given a dataset $(x_n, t_n), \forall i \in \{1, \dots, N\}$ and a new data point $\mathbf{x}_q$ we decide the class as follows:

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

We choose:

- Distance metric: Euclidean
- How many neighbours: 1
- Weight function: equal weights
- How to fit with local points: N.A.

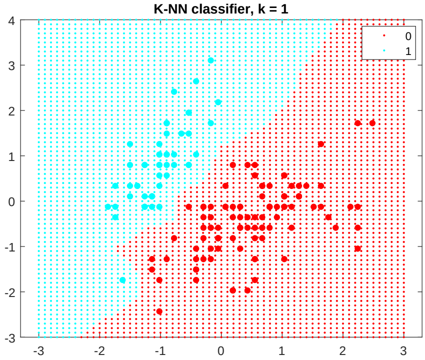
K-Nearest Neighbour (KNN)

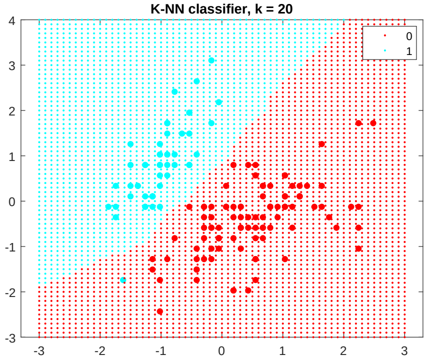
Different definition of the method:

- Distance metric: Euclidean
- How many neighbours: K
- Weight function: equal weights
- How to fit with local points: Majority voting

Regularizing with KNN

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





Categorization of the Classification Algorithms

Naive Bayes

- Parametric
- Frequentist
- Generative

K-Nearest Neigbour

- Non-parametric
- N.A.
- Discriminative

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

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