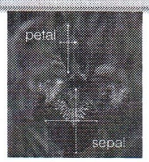
Machine Learning

Linear Regression



Francesco Trovò

Outline

Introduction

Outline

Introduction

Practical Information

- Course materials (slides, pdfs, links to recorded lectures) uploaded to Beep
- For questions or clarifications, ask after (or before) the lecture or ask by email for an appointment for a call: francesco1.trovo@polimi.it

Suggested literature to follow the course lectures:

- Bishop, C.M., "Pattern recognition and machine learning", 2006, Springer
- James, G., Witten, D., Hastie, T., Tibshirani, R., "An introduction to statistical learning", 2013, Springer

Regression Problem

Outline

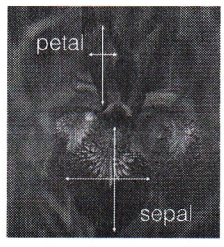
Regression Problem

Where Everything Starts

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)



Example of Dataset

Sepal length	Sepal width	Petal length	Petal width	Class
5.1000	3.5000	1.4000	0.2000	Iris-setosa
4.9000	3.0000	1.4000	0.2000	Iris-setosa
4.7000	3.2000	1.3000	0.2000	Iris-setosa
4.6000	3.1000	1.5000	0.2000	Iris-setosa
5.0000	3.6000	1.4000	0.2000	Iris-setosa
5.4000	3.9000	1.7000	0.4000	Iris-setosa
4.6000	3.4000	1.4000	0.3000	Iris-setosa
5.0000	3.4000	1.5000	0.2000	Iris-setosa
4.4000	2.9000	1.4000	0.2000	Iris-setosa
4.9000	3.1000	1.5000	0.1000	Iris-setosa
5.4000	3.7000	1.5000	0.2000	Iris-setosa

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** of a specific kind of *Iris setosa* by using the **petal length**

Solution: Linear Regression

We need to specify:

- Hypothesis space: $\hat{t}_n = f(x_n, w) = w_0 + x_n w_1$
- Loss measure: $J(w, x_n, t_n) = RSS(w) = \sum_n (\hat{t}_n - t_n)^2$
- Optimization method: Least Square (LS) method

where $w \in \mathbb{R}^M, M = 2$

We need to practically apply these operations to the dataset:

- By hand
- With a calculator
- With some code

Outline

Preliminary Operations

- Load Data
- Inspect Data
- Select the interesting Data
- Preprocessing
  - shuffling (`shuffle()`)
  - remove inconsistent data
  - remove outliers
  - normalize or standardize data (`zscore()`)
  - fill missing data

Standardization of sample s:

$$s \leftarrow \frac{s - \bar{s}}{S}$$

$$s \leftarrow \frac{s - \bar{s}}{\max_n\{s_n\} - \min_n\{s_n\}}$$

Linear Regression in Python

- We have many different solution from different libraries
- In this session we use `sklearn` and `statsmodels`
- Alternatively, we can implement the linear regression by hand ($w^* = (X^\top X)^{-1} X^\top t$)

First option:

- Create a linear model (`LinearRegression()`)
- Fit the model to the data (`fit()`)
- Analyse the results

Evaluating the Results

- Residual Sum of Squares (RSS), Sum Of Squared Errors (SSE):

$$RSS(w) = \sum_n (\hat{t}_n - t_n)^2$$

- Root Mean Square Error: $RMSE = \sqrt{\frac{RSS(w)}{N}}$
- Coefficient of determination: $R^2 = 1 - \frac{RSS(w)}{\sum_n (\bar{t} - t_n)^2}$
- Degrees of Freedom: $dfe = N - M$
- Adjusted coefficient of determination $R^2_{adj} = 1 - (1 - R^2)\frac{N-1}{dfe}$

Mean of the targets: $\bar{t} = \frac{\sum_n t_n}{N}$

Example of `OLS(x, t).fit()` Output

Dep. Variable:	y	R-squared (uncentered):	0.927
Model:	OLS	Adj. R-squared (uncentered):	0.926
Method:	Least Squares	F-statistic:	1889.
Date:	Wed, 10 Mar 2021	Prob (F-statistic):	1.56e-86
Time:	12:28:24	Log-Likelihood:	-16.645
No. Observations:	150	AIC:	35.29
Df Residuals:	149	BIC:	38.30
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	0.9628	0.022	43.467	0.000	0.919	1.007

Omnibus:	2.326	Durbin-Watson:	1.437
Prob(Omnibus):	0.313	Jarque-Bera (JB):	1.852
Skew:	0.210	Prob(JB):	0.396
Kurtosis:	3.347	Cond. No.	1.00

Statistical Tests on Coefficients

Single coefficients:

$$H_0 : w_j = 0 \qquad \text{vs.} \qquad H_1 : w_j \neq 0$$

$$t_{stat} = \frac{\hat{w}_j - w_j}{\hat{\sigma}\sqrt{v_j}} \sim t_{N-M-1}$$

where t_{N-M-1} is the T-Student distribution with $N - M - 1$ degrees of freedom

Overall significance of the model:

$$H_0 : w_0 = w_1 = \ldots = w_M = 0 \qquad \text{vs.} \qquad H_1 : \exists w_j \neq 0$$

$$F = \frac{dfe}{M-1} \frac{\sum_n (\hat{t} - t_n)^2 - RSS(w)}{RSS(w)} \sim F_{M-1,N-M-1}$$

where $F_{M-1,N-M-1}$ is the Fisher-Snedecor distribution with parameters $M - 1$ and $N - M - 1$

Different Implementations

`fit()`

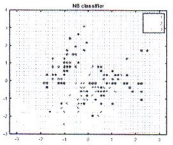
- Slow in the execution
- Many checks for consistency
- Recap

By-hand solution

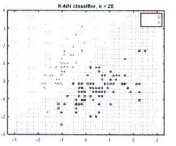
- Very fast
- No checks for consistency
- No tests on the coefficients

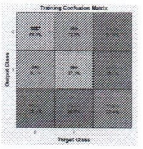
Machine Learning

Classification



Francesco Trovò





Outline

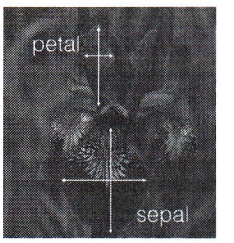
Outline

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 of a specific kind of Iris setosa by using the **petal length**? sepal length and width?

A Classification Problem

- In this specific case the target is not ordered, therefore we have a **Classification Problem**
- Initially, we solve the problem of discriminating between setosa and non-setosa flowers (binary classification problem)
- The target will be $y_n \in \{0, 1\}$
- As input $\mathbf{x}_n$ we choose sepal length and width (for visualization purposes)

Different Approaches for Classification

Three possible approaches:

- Discriminant function approach
  - model a function that maps inputs to classes
  - 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 $P(C_k|\mathbf{x}) = \dfrac{P(C_k)P(\mathbf{x}|C_k)}{P(\mathbf{x})}$

Possible Solutions

- Linear Classification
  - Perceptron
  - Logistic regression
- Naive Bayes
- K-nearest neighbour

Outline

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

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

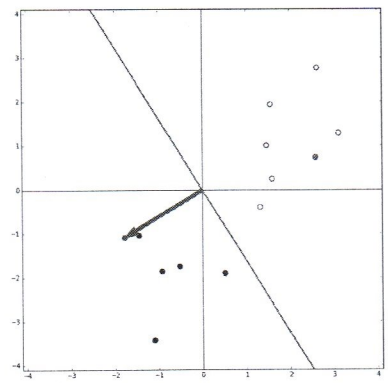
- Optimization method: Online Gradient Descent

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

Optimization in Python via:

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

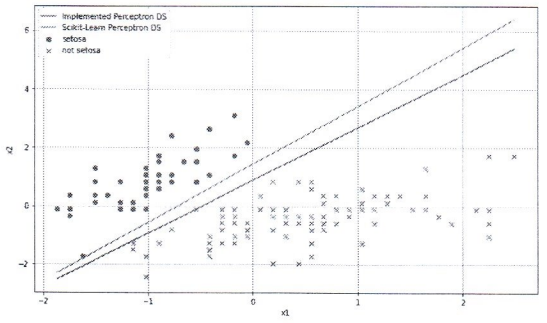
Learning Example



Plotting the results

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

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



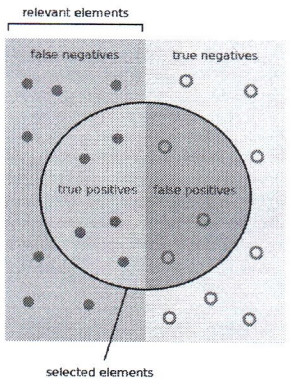
Evaluating the Results

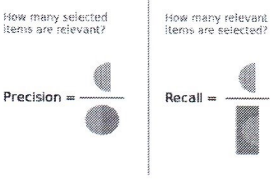
Confusion Matrix:

	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

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

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: $p(x_j|C_k) \sim \mathcal{N}(\mu_{jk}, \sigma^2_{jk})$, 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 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 $\hat{p}(C_1), \ldots, \hat{p}(C_K)$
- 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 $(x_n, t_n), \forall i \in \{1, \ldots, N\}$ and a new data point $\mathbf{x}_q$ we decide the class as follows:

$$y_q(\mathbf{x}_q) = \arg \min_{n \in \{1,\ldots,N\}} ||\mathbf{x}_q - \mathbf{x}_n||_2$$

We choose:

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

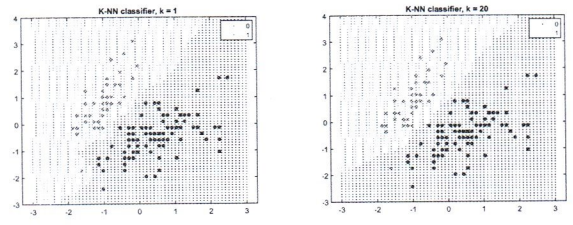
K-Nearest Neighbour (KNN)

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

Outline

Multiple Classes

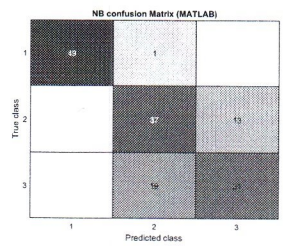
- In the case we have multiple classes we can use the same function, feeding a target with more than two labels
- It will train K different models, one for each class vs. the rest
- The parameter vector is now a matrix W

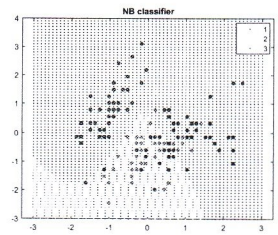
We can display the separating surfaces, but it would be a little more difficult than the case with two classes

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





Machine Learning

Bias-Variance Tradeoff

Francesco Trovò

Bias-Variance Dilemma

Known Process

To explicitly analyse the variance and the bias of a model we need to know the process generating the data:

$$t = t(x) = f(x) + \varepsilon = 1 + \frac{1}{2}x + \frac{1}{10}x^2 + \varepsilon$$

- the input are $x \in [0,\ 5]$
- the noise ε has $\mathbb{E}[\varepsilon] = 0$ and $Var(\varepsilon) = \sigma^2 = 0.7^2$

Two-Model Dilemma

Assume to approach the learning problem (we do not know the true model) using either one of the two following models:

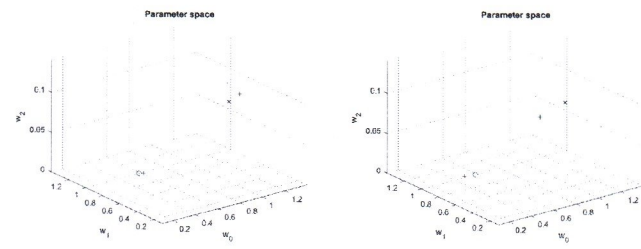
$$\mathcal{H}_1: \quad y(x) = a + bx$$
$$\mathcal{H}_2: \quad y(x) = a + bx + cx^2$$

Optimal Parameters

If the real mode is known we can compute the optimal model for the two hypothesis space:

$$\mathcal{H}_1: \quad \arg\min_{(a,b)} \int_0^5 (f(x) - a - bx)^2 dx = \left(\frac{7}{12},\ 1\right)$$
$$\mathcal{H}_2: \quad \arg\min_{(a,b,c)} \int_0^5 (f(x) - a - bx - cx^2)^2 dx = \left(1,\ \frac{1}{2},\ \frac{1}{10}\right)$$

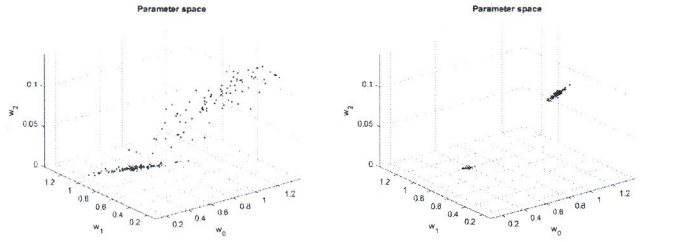
Optimal Parameters and Realized Parameters



The blue x is the best model in $\mathcal{H}_2$ and the red circle is the best model in $\mathcal{H}_1$
The pluses are the optimal parameters for two realization of the dataset ($N = 1000$)

Visualization of Bias and Variance

If we repeat the process for multiple times (generation of 100 independent dataset) with different number of samples ($N = 100$ on the left and $N = 10000$ on the right)



Computation of Bias and Variance

In this specific case we can even estimate the Bias and Variance of the two models:

$$\mathbb{E}[(t(x) - y(x))^2] = \sigma^2 + Var[y(x)] + \mathbb{E}[(f(x) - y(x))]^2$$

```
Linear error: 0.46867
Linear bias: 0.03613
Linear variance: 0.00011514
Linear sigma: 0.43242
Quadratic error: 0.42146
Quadratic bias: 1.412e-06
Quadratic variance: 0.00014674
Quadratic sigma: 0.42131
```

All the considerations holds on average, therefore there might be realizations for which the Bias and Variance of different models might not be coherent with what we saw

Bias-Variance Tradeoff

Model Selection Problem

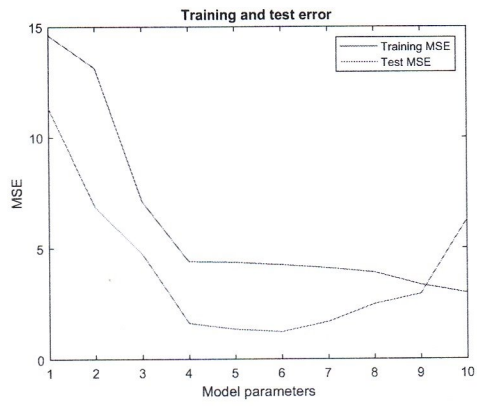
In real scenarios we do not know the real model, so we should select the correct one among a set of models.

Consider the possible solutions for a regression problem:

- Hypothesis space: $y_n = f(x_n, w) = \sum_{k=0}^{o} x_n^k w_k$
- Loss measure: $RSS(w) = \sum_n (y_n - t_n)^2$
- Optimization method: Least Square (LS)

The order o and other parameters which should be chosen before performing the training phase are usually addressed as *hyperparameters*

Limits of Using the Training error

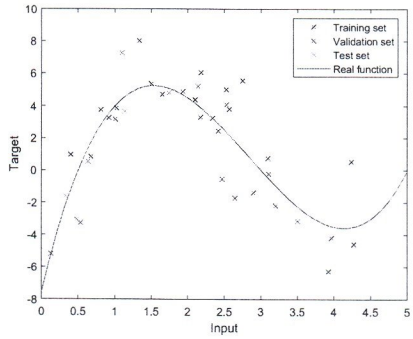


Validation

- Training set X_{train}, i.e., the data we will use to learn the model parameters
- Validation set X_{vali}, i.e., the data we will use to select the model
- Test set X_{test}, i.e., the data we will use to test the performance of our model

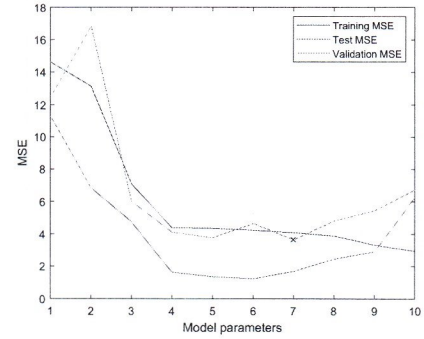
Usually, we use a split proportional to 50%-25%-25% for the three sets

Dataset Generated



Here it is clear the importance of shuffling data before performing the split

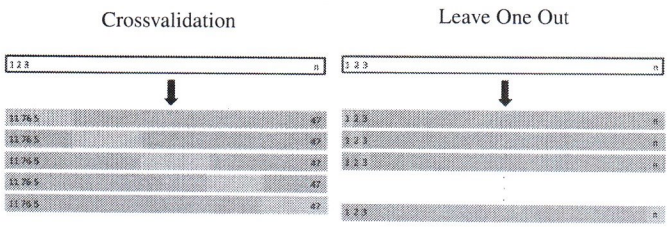
Validation Results



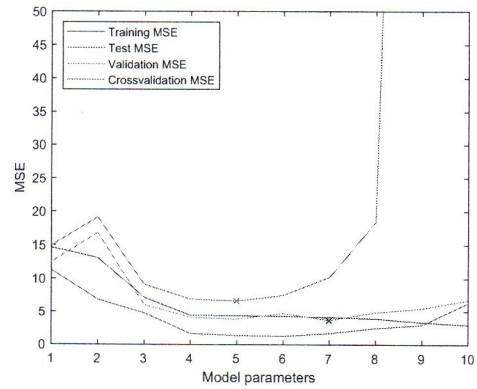
We are still biased w.r.t. the specific validation set we used

LOO and Crossvalidation

This way we reduce the amount of samples we could use for training of 33%, which could compromise the analysis since the training has been performed with a significantly smaller dataset



Crossvalidation Results ($K = 5$)



Checking the Results

The data have been generated from the following model:

$$y = (0.5 - x)(5 - x)(x - 3) + \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0, 1.5^2)$

The correct order is then $o = 3$ (4 in the graphs which considers also the constant term)

The procedure is correct on average, the realizations might return different orders than the correct one

Computational Times

Using different methods we have different time for the model selection:

```
Elapsed time is 0.016354 seconds. % Validation
Elapsed time is 0.431666 seconds. % Crossvalidation
Elapsed time is 4.308715 seconds. % LOO
```

Depending on the computational power available and the number of data we have we might choose different methods

Machine Learning

Model Selection

Francesco Trovò

Definition of different models

What to do in the case the model you are considering is not performing well even by tuning properly the parameters (cross-validation)?

We have two opposite options:

- simplify the model (model selection)
- increase its complexity (next time)

How to Select a Model

We already discussed how to evaluate a specific model (bias/variance dilemma)

- Model Selection
  - Feature selection: choose only a subset of significant features to use
  - Regularization (shrinkage): introduce some penalization for complex models in the loss function
  - Dimensionality reduction: project the features in a lower dimensional space
- Ensemble model
  - Bagging
  - Boosting

Model Selection

Model Selection

- Feature Selection
  - Filter methods
  - Embedded FS
  - Wrapper methods
    - Brute Force
    - Forward Step-wise Selection
    - **Backward Step-wise Selection**
- Feature Extraction
  - **PCA**
  - ICA
- Regularization
  - LASSO
  - Ridge
  - ...

Feature Selection: Brute Force

In principle one evaluate:

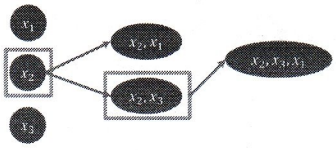
- For each feature x_k with $k \in \{0, \dots, M\}$
  - Learn all the possible $\binom{M}{k}$ possible models with k inputs
  - Select the model with the smallest loss
- Select the k providing the model with the smallest loss

Problem: if M is large enough the computation of all the models is unfeasible

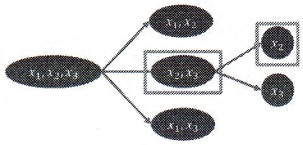
Wrapper Methods

We evaluate only a subset of the possible models

Forward Feature Selection



Backward Feature Selection



we always take the best and we don't reconsider (among x_1, x_2, x_3 → x_2 is the best, then x_2 remains always)

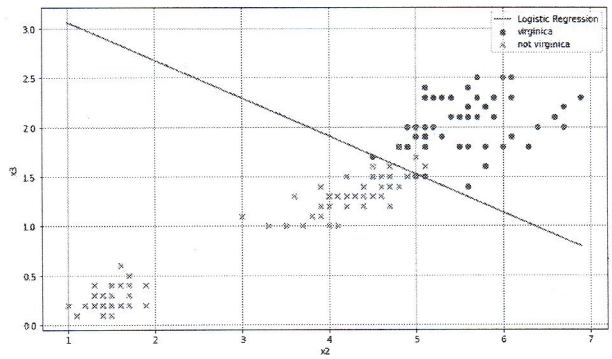
Backward Feature selection on the Iris Dataset (1)

- Assume the problem is to discriminate between Virginica and Non-Virginica iris
- We select a performance metric: validation accuracy on 20% of the data
- Train a model on the full data (x_1, x_2, x_3, x_4): Logistic regression
- Remove one of the features and check the error:
  - Model with (x_1, x_2, x_3): accuracy 1
  - Model with (x_1, x_3, x_4): accuracy 1
  - Model with (x_1, x_2, x_4): accuracy 1
  - Model with (x_2, x_3, x_4): accuracy 1
- Removing a single feature does not change the method performance

Backward Feature selection on the Iris Dataset (2)

- Let us remove one of the features at random x_4
- Remove another feature and check the error:
  - Model with (x_1, x_2): accuracy 0.96
  - Model with (x_1, x_3): accuracy 0.96
  - Model with (x_2, x_3): accuracy 1
- The model with (x_2, x_3) is performing better than the others
- Iterate *one more time*

Results on the Iris Dataset



Principal Component Analysis

Idea

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction technique, i.e., which extract some low dimensional features from a dataset

We would like to perform a linear transformation of the original data X s.t. the largest variance lies on the first transformed feature, the second largest variance on the second transformed feature, ...

At last we only keep some of the features we extract

Procedure

- Translate the original data X to $\tilde{X}$ s.t. they have zero mean
- Compute the covariance matrix of $\tilde{X}$, $C = \tilde{X}^T \tilde{X}$
- The eigenvector e_1 corresponding to the largest eigenvalue λ_1 is the first principal component
- The eigenvector e_2 corresponding to the second largest eigenvalue λ_2 is the second principal component
- ...

Given a sample vector $\tilde{\mathbf{x}}$, its transformed version $\mathbf{t}$ can be computed using:

$$\mathbf{t} = \tilde{\mathbf{x}}\, W$$

where t_i is the i-th principal component

- loadings: W matrix of the weights
- scores: T transformation of the input dataset $\tilde{X}$
- variance: $(\lambda_1, \ldots, \lambda_M)$ vector of the variance of principal components

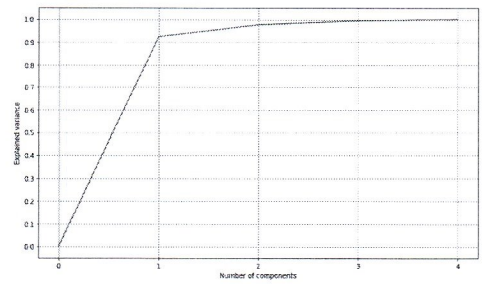
How Many Features

There are a few different methods to determine how many feature to choose

- Keep all the principal components until we have a cumulative variance of 90%-95%
- Keep all the principal components which have more than 5% of variance (discard only those which have low variance)
- Find the elbow in the cumulative variance

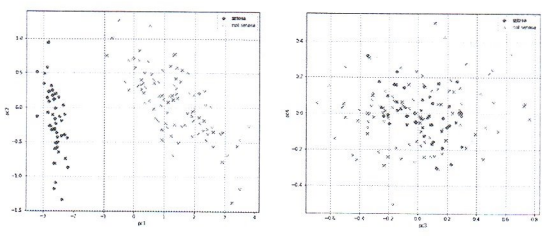
Cumulated Variance Plot

Using the Iris dataset inputs:



Principal Components

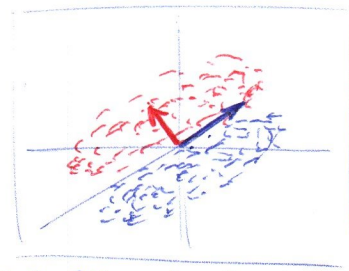
If we separate the first two components from the second twos:



Is PCA always good?

Simpson's Paradox:

Suppose to have this dataset:



we apply PCA and as first PC we obtain: →
and as second: →

If we keep only the 1st PC we project the data on →, the problem here is that the direction → explains most of the overall variance, however it not informative! Knowing the labels we can see that → makes more sense (works better).

However PCA is unsupervised ⇒ NO LABELS.

PCA Different Purposes

- Feature Extraction: reduce the dimensionality of the dataset by selecting only the number of principal components retaining information about the problem
- Compression: the linear transformation W minimizes among the ones with k dimension $\min_{W_{red}} ||W_{red}\tilde{X}(k) - \tilde{X}||_2^2$, i.e., is the linear transformation minimizing the reconstruction error
- Data visualization: reduce the dimensionality of the input dataset to 2 or 3 to be able to visualize the data

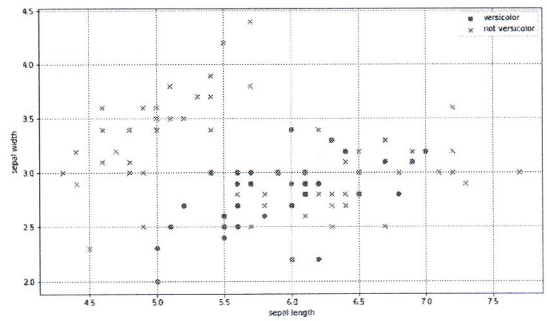
Regularization

Regularization

Already known regularization procedure:

- Ridge:
$$L(\mathbf{w}) = \frac{1}{2}RSS(\mathbf{w}) + \frac{\lambda}{2}||\mathbf{w}||_2^2$$
- Lasso:
$$L(\mathbf{w}) = \frac{1}{2}RSS(\mathbf{w}) + \frac{\lambda}{2}||\mathbf{w}||_1$$
- Elastic net:
$$L(\mathbf{w}) = \frac{1}{2}RSS(\mathbf{w}) + \frac{\lambda_1}{2}||\mathbf{w}||_2^2 + \frac{\lambda_2}{2}||\mathbf{w}||_1$$
- They can be applied to the linear regression technique, it can be extended for other methods
- For classification we will see some specific methods

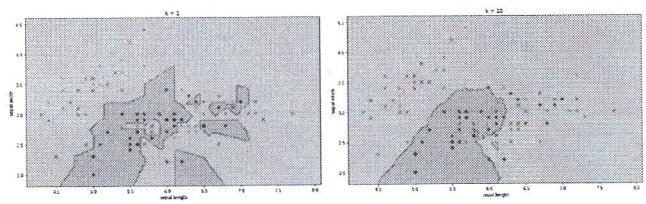
A hard problem



This problem cannot be solved with a linear classification technique

K-Nearest Neighbour

Different values of the K parameter



The larger the value of K, the more the model is regularized ($1/K$ acts as a regularization hyperparameter)

Machine Learning

Kernel Methods

Francesco Trovò

Definition of Different Models

What to do in the case the model you are considering is not performing well even by tuning properly the parameters (cross-validation)?

We have two opposite options:

- simplify the model
- **increase its complexity** — increase the dimensionality of the features without actually computing the features

In the second hypothesis one might see the problem in a more complex space:

- use handcrafted features
- look at the problem in the kernel space

Gaussian Process

kernel method for regression

GP Definition

- A Gaussian process is defined as a probability distribution over functions $\mathbf{y}(\mathbf{x})$ such that the set of values of $\mathbf{y}(\mathbf{x})$ evaluated at an arbitrary set of points $\mathbf{x}_1, \dots, \mathbf{x}_N$ jointly have a Gaussian distribution
- This distribution is completely specified by the mean and the covariance:
  - usually, we do not have any prior information about the mean of $\mathbf{y}(\mathbf{x})$, so we take it to be zero
  - the covariance is given by the kernel function $\mathbb{E}[t(x_i)\, t(x_j)] = K(x_i, x_j)$
- With this formulation, Gaussian Process (GP) are kernel methods that can be applied to solve regression problems

Output Modeling

- Assume to have a target $t_n = y(\mathbf{x}_n) + \varepsilon_n$, where ε_n is a measurement noise which is not dependent on the specific point $\mathbf{x}_n$
- The joint distributions of the targets $\mathbf{t}$ of dimensions N is:

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \sigma^2 I_N)$$

- since $p(\mathbf{y}) = \mathcal{N}(\mathbf{0}, K)$, we have:

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y}) p(\mathbf{y})\, d\mathbf{y} = \mathcal{N}(\mathbf{t}|\mathbf{0}, C)$$

where $C(\mathbf{x}_n, \mathbf{x}_m) = K(\mathbf{x}_n, \mathbf{x}_m) + \delta_{nm}\sigma^2$ and δ_{nm} is the Dirac delta, i.e., $\delta_{nm} = 1 \iff n = m$

Making Predictions

We would like to predict the target t_{N+1} corresponding to a specific unseen input $\mathbf{x}_{N+1}$

From the definition we have:

$$p(\mathbf{t}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1}|\mathbf{0}, C_{N+1}),$$

where $C_{N+1} = \begin{pmatrix} C_N & \mathbf{k} \\ \mathbf{k}^\top & c \end{pmatrix}$, $\mathbf{k}$ is the vector of $K(\mathbf{x}_i, \mathbf{x}_{N+1})$, for each $i \in \{1, \dots, N\}$ and c is the covariance $C(\mathbf{x}_{N+1}, \mathbf{x}_{N+1})$

We need to compute: $p(t_{N+1}|\mathbf{t}_N, \mathbf{x}_1, \dots, \mathbf{x}_N)$

- Mean: $m(\mathbf{x}_{N+1}) = \mathbf{k}^\top C_N^{-1} \mathbf{t}$
- Variance: $\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^\top C_N^{-1} \mathbf{k}$

GPs in Python

Model the relationship between petal length and width as a GP:

- Load the data and normalize them
- Select the value for the GP:
  - noise variance $\sigma^2 = Var[\varepsilon_n] = 0.2$
  - constant $k = 1$
  - lengthscale $l = 0.8$
- Initialize a GP regression model (`GaussianProcessRegressor`)
- Predict new values

Kernel:

$$\mathbf{K}_{ij} = k \exp\left\{-\frac{||x_i - x_j||_2}{2l^2}\right\}$$

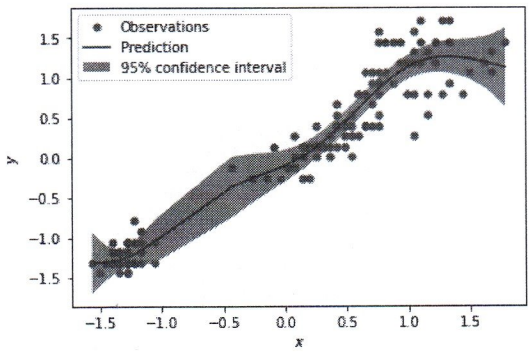
Hyperparameters

While GPs are a non-parametric methods, the parameters of the kernel has to be estimated or set:

- using a priori information on the problem we are analysing
- maximizing their loglikelihood on an independent dataset
- possibly improved as new data are collected

Caveat: most of the time you will see that they are estimated using the same data used for the prediction. This is clearly not a good ML practice (equivalent to overfitting)

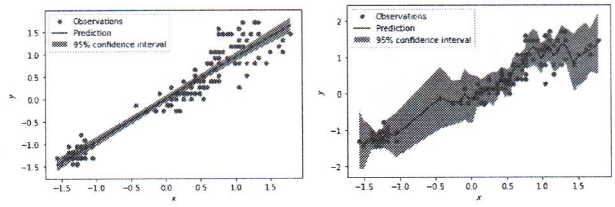
Results on the Iris Dataset



Parameters: $k = 3$, $l = 0.8$, and $\sigma^2 = 0.2$

Modify the Lengthscale

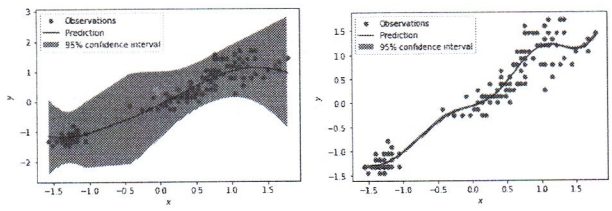
- Left: $k = 3$, $l = 8$, and $\sigma^2 = 0.2$
- Right: $k = 3$, $l = 0.08$, and $\sigma^2 = 0.2$



Controls the smoothness of the GP

Modify the Noise

- Left: $k = 3$, $l = 0.8$, and $\sigma^2 = 10$
- Right: $k = 3$, $l = 0.8$, and $\sigma^2 = 0.002$



Controls the point noise of the GP

SVM

Support Vector Machines

- Flexible and theoretically supported method
- Initially only applied to classification, over the years it has been extended to deal with regression, clustering and anomaly detection problems
- Idea: find the hyperplane maximizing the margins (distance between the boundary and the points)

- Hypothesis space: $y_n = f(\mathbf{x}_n, \mathbf{w}) = sign\left(\mathbf{w}^T\mathbf{x}_n + b\right)$ equation of the boundary
- Loss measure: $||\mathbf{w}||^2 + C\sum_i \zeta_i$ s.t. $t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \zeta_i \; \forall n$ penalization of the missclassified
- Optimization method: quadratic optimization

Linear SVM in Python

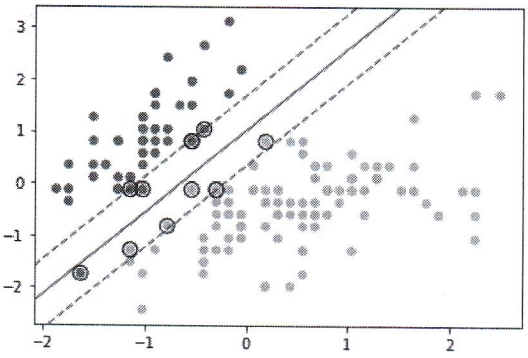
To train a linear classification SVM:

- Define an SVM: `SVM_model.svm.SVC(kernel='linear')`
- Train the SVM: `SVM_model.fit(input, target)`

We are interested to determine:

- Boundary $\mathbf{w}^T\mathbf{x}_n + b = 0$
- Margins $\mathbf{w}^T\mathbf{x}_n + b = \pm 1$
- Support vectors (`SVM_model.support_vectors_`)

Results on the Iris Dataset



Adding a Kernel

The use of kernels in the SVM is almost native (non-parametric method):

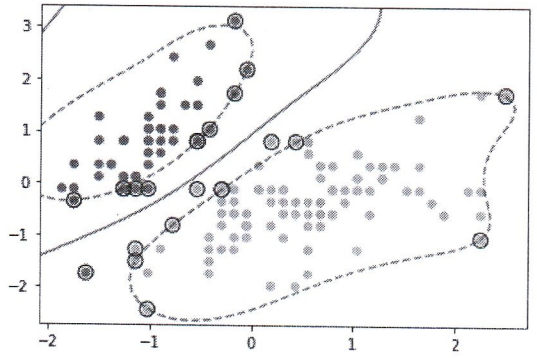
- Hypothesis space: $y_n = f(\mathbf{x}_n, \mathbf{w}) = sign\left(\sum_n \alpha_i t_i K(\mathbf{x}_i, \mathbf{x}_n) + b\right)$
- Loss measure: loss function in the dual formulation
- Optimization method: quadratic optimization

In `Python`:

- Define an SVM: `SVM_model.svm.SVC()`
- Train the SVM: `SVM_model.fit(input, target)`

We do not have an explicit formula for the boundary and the margins anymore

Results on the Iris Dataset



Machine Learning

Markov Decision Processes

Francesco Trovò

Markov Decision Process

└ general models used to formalize sequential problems (sequential decision making)

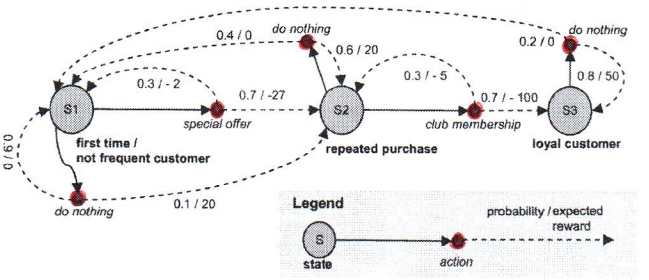
Two different problems

We would like to model the dynamics of a process and the possibility to choose among different actions in each situation

Two different problems:

- Prediction: given a specific behaviour (policy) in each situation, *estimate the expected long-term reward* starting from a specific state
- Control: learn the optimal behaviour to follow in order to *maximize the expected long-term reward* provided by the underlying process

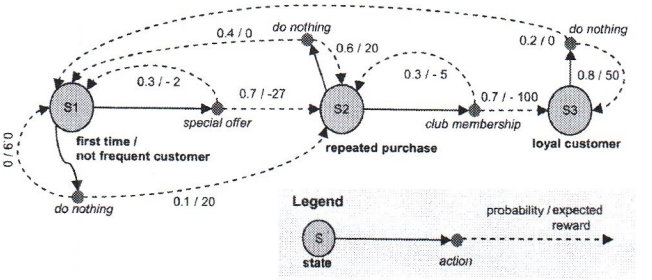
Example: Advertising Problem



- Given the actions in each state ($S1$, $S2$, $S3$) compute the value of a state
- Determine the best action in each state

Prediction

Prediction on the Advertising Problem



Given the policy $(do\ nothing, do\ nothing, do\ nothing)$, compute the value of each state

Modeling the MDP

First, we model the MDP $\mathcal{M} := (\mathcal{S}, \mathcal{A}, P, R, \mu, \gamma)$ for the given problem:

- States: $\mathcal{S} = \{\text{first time, repeated purchaser, loyal customer}\}$
- Actions: $\mathcal{A} = \{\text{do nothing, special offer, club membership}\}$
- Transition model: $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, we need $\dim(P) = |\mathcal{S}||\mathcal{A}||\mathcal{S}|$ numbers to store it
- Reward function: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we need $\dim(R) = |\mathcal{S}||\mathcal{A}|$ numbers to store it
- Initial distribution $\mu \in \Delta(\mathcal{S})$
- Discount factor: $\gamma \in (0, 1]$

where $\Delta(\cdot)$ represents the simplex of a set

We assume that all the customer are first timers $\mu = (1, 0, 0)$ and use $\gamma = 0.9$

Modeling the MDP in Python

Since we know the policy π already, which is defined as

$$\pi : \mathcal{S} \to \Delta(\mathcal{A})$$

we can directly represent P^π and R^π, which are defined as:

$$P^\pi(s'|s) = \sum_a \pi(a|s) P(s'|s, a) \qquad \dim(P^\pi) = |\mathcal{S}||\mathcal{S}|$$

$$R^\pi(s) = \sum_a \pi(a|s) R(s, a), \qquad \dim(R^\pi) = |\mathcal{S}|$$

Computing the Value of the States

We have the Bellman expectation equation:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^\pi(s') \right]$$

which we can rewrite in matrix form as:

$$V^\pi = R^\pi + \gamma P^\pi V^\pi \qquad \dim(V^\pi) = |\mathcal{S}|$$

Closed-Form Solution

Thanks to the Bellman expectation equation:

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

Since P^π is a stochastic matrix, we have that the eigenvalues of $(I - \gamma P^\pi)$ are in $(0, 1)$ for $\gamma \in (0, 1)$ and the matrix is invertible

Recursive Solution

In the case we are not able to invert the matrix (the state space is too large) let us consider the recursive version of the Bellman expectation equation:

$$V^\pi = R^\pi + \gamma P^\pi V^\pi$$

```
V_old = np.zeros(nS)
tol = 0.0001
V = pi @ R_sa
while np.any(np.abs(V_old - V) > tol):
  V_old = V
  V = pi @ (R_sa + gamma * P_sas @ V)
```


Evaluating Different Policies

By changing the policy, which in matrix form is

$$\pi(a|s) = \Pi(s, a|s) \qquad \dim(\Pi) = |\mathcal{S}||\mathcal{S}||\mathcal{A}|$$

we are able to compute the values of the states with different strategies:

- **myopic:** we do not want to spend any money in marketing

```
pi_myo = np.array([[1., 0., 0., 0., 0.],
                   [0., 0., 1., 0., 0.],
                   [0., 0., 0., 0., 1.]])
```

- **far-sighted:** we want to spend some money in marketing for the customer in both cases if she is a new customer or if she repeatedly purchased

```
pi_far = np.array([[0., 1., 0., 0., 0.],
                   [0., 0., 0., 1., 0.],
                   [0., 0., 0., 0., 1.]])
```

Results with Different Discounts

$\gamma = 0.5$		$\gamma = 0.9$		$\gamma = 0.99$	
m	f	m	f	m	f
5.3333	-47.6202	36.3636	-9.2889	396.0396	785.3831
18.6667	-59.9347	54.5455	20.1890	415.8416	824.8548
67.5556	58.7300	166.2338	136.8857	569.3069	939.9320

- For $\gamma = 0.5$ the myopic policy evidently outperforms the far-sighted one
- For $\gamma = 0.9$ the two policies are getting close
- For $\gamma = 0.99$ the far-sighted policy becomes the most rewarding one

Control

Select the Policy

- Brute force: enumerate all the possible policies, evaluate their values and consider the one having the maximum values, generally requires $|\mathcal{S}|^{|\mathcal{A}|}$ evaluation steps
- Policy Iteration: iteratively evaluate the current policy and update it in the greedy direction
- Value Iteration: iteratively apply the Bellman optimality equation in its recursive form

In this case we do not have the option to solve the Bellman optimality equation in a closed form since the max operator is not linear

Policy Iteration

We want to solve the following problem:

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|a,s) V^*(s') \right\}$$

Decouple the process into:

- *policy evaluation*, where we compute the value V^π of the given policy
- *policy improvement*, where we change the policy from π to π' according to the newly estimated values (greedy improvement)

$$\begin{aligned} a'(s) &= \arg\max_{a \in \mathcal{A}} Q^\pi(s,a) \\ &= \arg\max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|a,s) V^\pi(s') \right\} \quad \forall s \in \mathcal{S} \end{aligned}$$

Value Iteration

Instead of iterating between policy evaluations and improvements, let us try to evaluate the optimal policy directly (i.e., to compute V^*), by repeatedly applying the Bellman optimality equation on the current value function V_k:

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_k(s') \right\}$$

This procedure is guaranteed to V^* eventually (because the Bellman optimality equation induces a contraction)
Once we have V^*, we can easily recover the optimal policy, i.e., the greedy one w.r.t. V^*

Machine Learning

Reinforcement Learning

Francesco Trovò

Reinforcement Learning For Prediction

Possible Options for Prediction

When we want to perform prediction and we do not know the environment dynamics or modeling the environment is too complex:

- Monte Carlo (first and every visit)

$$V(s_t) \leftarrow V(s_t) + \alpha(v_t - V(s_t))$$

- Temporal Difference

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

- $TD(\lambda)$ (eligibility traces)

$$V(s_t) \leftarrow V(s_t) + \alpha(v_t^\lambda - V(s_t))$$

with $v_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} v_t^{(n)}$

Exercise 8.5

Evaluate the value for the MDP with three states $\mathcal{S} = \{A, B, C\}$ (C is terminal), two actions $\mathcal{A} = \{h, l\}$ given the policy π, given the following trajectories:

$$(A, h, 3) \rightarrow (B, r, 2) \rightarrow (B, h, 1) \rightarrow (C)$$
$$(A, h, 2) \rightarrow (A, h, 1) \rightarrow (C)$$
$$(B, r, 1) \rightarrow (A, h, 1) \rightarrow (C)$$

- Can you tell without computing anything if by resorting to MC with every-visit and first-visit approach you will have different results?
- Compute the values with the two aforementioned methods
- Assume to consider a discount factor $\gamma = 1$ and compute the values by resorting to TD? Assume to start from zero values for each state and $\alpha = 0.1$

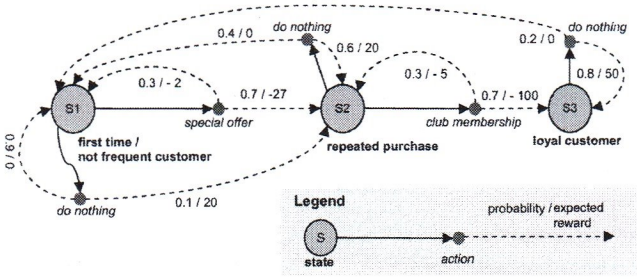
Reinforcement Learning For Control

	Bellman expectation equation	Bellman optimality equation
dynamic programming	policy iteration	value iteration
empirical version	• Monte Carlo control • SARSA	• Q-learning

Possible Options for Control

- Monte Carlo Control: Monte Carlo Estimation combined with ε-greedy policy improvement
- SARSA: Temporal Difference Estimation combined with ε-greedy policy improvement
- Q-learning: empirical version of Value Iteration

Example: Advertising Problem



RL Basic Elements

The elements needed to apply RL algorithms are:

- Dataset or model generating data
- Policy improvement step
- Evaluation (update) step

Transition Model

Let us model the transition model of the advertising problem from which we will get episodes used in the RL algorithms:

$$r : S \times A \to \mathbb{R}$$
$$P : S \to S$$

Especially, we need to define the generative process:

```
class Environment(object):
    ...
    def transition_model(self, a):
        ...
        return s_prime, inst_rew
```

it gets an action and returns the next state and the reward

Policy Improvement Step

The ε-greedy policy is:

```
def eps_greedy(s, Q, eps, allowed_actions):
    if np.random.rand() <= eps:
        a = % take a random action
    else:
        Q_s = Q[s, :].copy()
        Q_s[allowed_actions == 0] = - np.inf
        a = np.argmax(Q_s)
    return a
```

NB: we need to manage also the case in which the Q-values of more than one action have the same value in a state

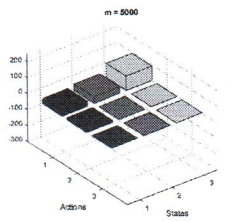
SARSA

The SARSA algorithm iterates between:

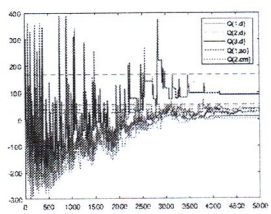
- An environment step, with the transition model
- A policy improvement step, with the ϵ-greedy policy
- An evaluation step, with the TD update of the Q function:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$$

SARSA - Results



final Q approximation



Solutions Comparison

Is it a good solution?

SARSA			Exact		
40.2274	8.5816	0	36.3636	24.6818	0
67.3932	0	6.0867	54.5455	0	47.9545
79.7005	0	0	166.2338	0	0

Depending on the task we are interested in, we have a good or a poor one

If we just have to select the optimal policy then it's okay. If we also have to have an estimate (approximate but good) then it's not too good.

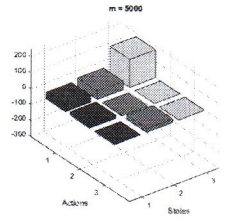
Q-learning

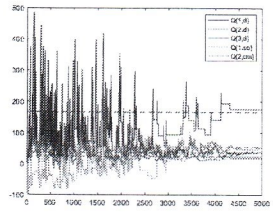
The Q-learning algorithm iterates over:

- An environment step, with the transition model
- A policy improvement step, with the ϵ-greedy policy
- An update with the Bellman optimality equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \max_{\tilde{a}\in\mathcal{A}} Q(s',\tilde{a}) - Q(s,a)\big)$$

Q-learning - Results





Solutions Comparison

SARSA			Q-learning			Exact		
40.22	8.58	0	41.15	25.13	0	36.36	24.68	0
67.39	0	6.08	68.68	0	28.26	54.54	0	47.95
79.70	0	0	127.83	0	0	166.23	0	0

On-policy vs Off-policy

With Q-learning I could have had a dataset to use for learning, with SARSA I need to execute the ε-greedy policy at each time point

Possible Solution: use importance sampling

Final Considerations

What if I need to implement the previous two methods on a different environment?
Just replace `transition_model(s, a)` with the one corresponding to the new environment

What other could I change in the learning process?

- M time horizon (#samples we use for prediction: the higher the better)
- α learning rate
- ε exploration incentive