Università degli Studi di Ferrara

Università degli Studi di Ferrara Corso di

Laurea in Informatica

Integrazione di una applicazione web con strumenti per la statistica

Relatore:

Prof. Fabrizio Riguzzi

Laureando:

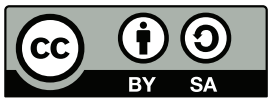
Franco Masotti

Anno Accademico 2017 – 2018

Indice

Elenco delle figure

Elenco dei listati



Capitolo 1

Introduzione

1.1 Obbiettivi

L'obbiettivo di questa tesi è quello di permettere ad un ambiente web di programmazione logica chiamato SWISH di usare un ambiente per la statistica.

Verrà usata una particolare versione di SWISH, chiamata Cplint on SWISH, che oltre a mettere a disposizione l'interfaccia web, comprende anche strumenti basati sull'intelligenza artificiale. Saranno messi in evidenza alcuni strumenti e modalità per generare grafici correlati ai risultati ottenuti da programmi di Cplint on SWISH. Questo sarà possibile grazie all'ambiente di statistica e visualizzazione grafica chiamato R. In questo senso verranno anche illustrati gli esempi più rilevanti.

Verranno spiegate anche le modalità di installazione di tutti i software necessari, e, vista la complessità di queste operazioni, alcuni modi per semplificarle.

1.2 Struttura

Nel capitolo 2 ci si è soffermati sulla descrizione di tutti gli strumenti necessari. Nel capitolo 3 ci si è concetrati sulla loro installazione e gestione. Nel capitolo 4 si sono descritte le modalità di disegno dei grafici e della descrizione della libreria che permette questo. Nel capitolo 5 si sono descritti alcuni esempi particolarmente complessi che non fanno uso della libreria appena citata.

Capitolo 2

Strumenti e architettura

2.1 R

R è un ambiente software per il calcolo statistico e la visualizzazione grafica. Con il termine R si definisce l'interprete dei comandi mentre con S il linguaggio di programmazione. R comprende anche un gestore di pacchetti attraverso il quale si possono installare facilmente funzioni utili non comprese in R stesso [1] [1]

2.1.1 Comandi e operatori fondamentali

Come la maggior parte dei linguaggi di programmazione, R mette a disposizione vari tipi di operatori tra cui quelli aritmetici, logici e di assegnazione.

Esistono varie possibilità per assegnare un valore ad una variabile. Di solito però viene preferita la notazione `<-` piuttosto che il classico `=`. Nel primo caso infatti non bisogna preoccuparsi di alcuni dettagli funzionali [2] [3].

Una delle operazioni fondamentali di R è la funzione `c()`. Questa serve per concatenare un insieme arbitrario di valori, in modo che questi possano essere salvati all'interno di un'unica variabile [3] [4].

2.1.2 Oggetti

R mette a disposizione strutture dati chiamate oggetti. Questi possono essere array di numeri, stringhe, funzioni, liste, *data frame*, ed altre entità [5].

In R le liste sono una collezione ordinata di oggetti, chiamati anche componenti, che possono essere di tipo diverso tra di loro. Con le liste si ha la possibilità di accedere direttamente alle componenti sia attraverso il principio dell'indicizzazione, sia attraverso l'uso della notazione per nome, del tipo `lista$componente`. [6]

[1] Da questo momento in poi, per semplicità, si userà il termine R per indicare anche il linguaggio.

I *data frame* sono un tipo particolare di oggetti che permettono di organizzare i dati sotto forma matriciale e sono definiti come specializzazione della classe lista [7]. Rispetto ad una lista, infatti, esistono delle restrizioni tra le quali le più importanti sono che le componenti devono essere vettori della stessa lunghezza. Un data frame può essere considerato per molti aspetti una matrice anche perché si possono usare le stesse convenzioni per visualizzare e manipolare i dati. Tuttavia, a differenza delle matrici, i vettori che lo compongono possono avere valori di tipo diverso, come avviene nelle tabelle [8].

Sia le liste sia i data frame verranno usati successivamente per gestire alcuni dati.

2.1.3 Funzioni grafiche di R

R dispone di un semplice meccanismo interno per il disegno di grafici nel quale l'output grafico dell'interprete è aggiunto direttamente al *plotting device* piuttosto che separatamente per ogni elemento distinto del plot. Questo sistema non permette ad esempio di poter trasformare i dati secondo scale e livelli diversi [9].

Per andare oltre queste limitazioni alcuni sviluppatori hanno scritto nuove librerie grafiche. Ad esempio, il package R *ggplot2* [10] permette di disegnare grafici usando la cosiddetta *grammatica dei grafici* [11]. Questa consiste nella divisione per contesti dei vari aspetti di un grafico come per esempio la geometria, la scala, la rappresentazione estetica, ed altri aspetti [12]:

> *ggplot2 è un sistema di plotting per R, basato sulla grammatica dei grafici, che cerca di utilizzare le parti migliori dei grafici di base e lattice, e di escludere le parti peggiori. Questo sistema si occupa della maggior parte dei dettagli poco pratici che rendono il plotting problematico (come per esempio il disegno di legende), oltre che fornire un potente modello di grafica che rende semplice produrre complessi plot con molteplici strati.*

2.2 SWI Prolog e SWISH

Prolog è un linguaggio di programmazione dichiarativo usato per esprimere relazioni tra elementi attraverso fatti e regole logiche [13].

SWI Prolog è una particolare implementazione del linguaggio Prolog. Si tratta di un ambiente di programmazione Prolog, libero e gratuito [14] [15].

SWISH è un'interfaccia web di SWI Prolog ed ha lo scopo principale di far collaborare gli utenti per scrivere programmi Prolog ed analizzarne i risultati [16] [17] [18].

2.2.1 Funzioni grafiche di SWISH

SWISH offre due tipi di funzionalità grafiche, una attraverso C3.js e l'altra attraverso R. C3.js è una libreria Javascript che a sua volta deriva da D3.js, anche chiamata *Data-Driven Documents*. Questa permette

di visualizzare dei dati utilizzando la Scalable Vector Graphics (*SVG*) all'interno dell'elemento *Canvas* che è un'area all'interno di un pagina web nella quale è possibile disegnare in due dimensioni [19] [20]. Questi grafici, inoltre, sono interattivi a differenza di quelli di quelli generabili da R.

2.3 Cplint on SWISH

Cplint è una suite di programmi di logica probabilistica e comprende sia programmi per fare inferenze sia per l'apprendimento [21]. Questa libreria è inclusa all'interno della versione di SWISH chiamata *Cplint on SWISH* [22].

2.4 rserve_client

rserve_client è una libreria Prolog che rende possibile l'accesso ad un ambiente R da parte di SWISH [23] [24]. Questa libreria include anche delle funzionalità per creare e gestire i data frame R direttamente da SWI Prolog [25].

2.4.1 Motivazione

L'utilizzo di R rispetto a Prolog comporta in molti casi un netto miglioramento della velocità di calcolo [26]:

> *È risultato che R è più veloce nel lavorare con una grande quantità di numeri interi rispetto a Prolog. Perché? perché una volta trasformato in R, il vettore è un semplice array C di interi, e quindi molto veloce nell'addizione. La versione Prolog invece somma i numeri uno alla volta oltre che controllare eventuali overflow. (..)*

Un altro vantaggio nell'utilizzare R è la possibilità di accedere a tutte le funzionalità di computazione che non sono presenti in SWI Prolog, come ad esempio le funzioni per studi statistici.

2.4.2 Scambiare dati fra R e Prolog

Esistono due modalità per usare i comandi di R all'interno di SWI Prolog:

- come *Prolog term*, usando l'operatore `<-` seguito dal comando R,
- oppure con una notazione chiamata *quasi-quotation*

La documentazione di SWISH contiene un esempio per entrambi i casi[27]:

> *- Le quasi quotations possono essere usate per fare copia incolla del codice R nel proprio programma Prolog senza necessità di cambiarlo (tranne quando contiene }).*

- *Il codice Prolog protrebbe aver bisogno di piccole modifiche. Nell'esempio,* `I(.5)` *deve essere cambiato in* `'I'(0.5)` *perché i floating point in prolog non possono iniziare con* `.` *e gli argomenti (usati come simboli di funzioni) non possono iniziare con una lettera maiuscola. L'utilizzo del prolog term tuttavia*

- *È più compatibile.*

- *Giova del controllo della sintassi Prolog.*

- *Rende più facile comporre espressioni R da elementi più piccoli.*

Per l'esempio in Prolog term si veda il listato 1 mentre per l'esempio quasi quotation si veda il listato 2. Come si vede nel primo caso è necessario fare qualche piccola modifica rispetto all'originale R. Nel secondo caso invece è possibile gestire alcuni casi particolari che genererebbero errori oppure che non sarebbe possibile trasformare nel primo caso.

Listato 1: Esempio notazione Prolog term

```
<- qplot(mpg, data=mtcars, geom="density", fill=gear, alpha='I'(0.5), main="Distribution of Gas
↪  Milage", xlab="Miles Per Gallon", ylab="Density").
```

Listato 2: Esempio notazione quasi quotation

```
<- {|r||qplot(mpg, data=mtcars, geom="density", fill=gear, alpha=I(.5), main="Distribution of Gas
↪  Milage", xlab="Miles Per Gallon", ylab="Density")|}.
```

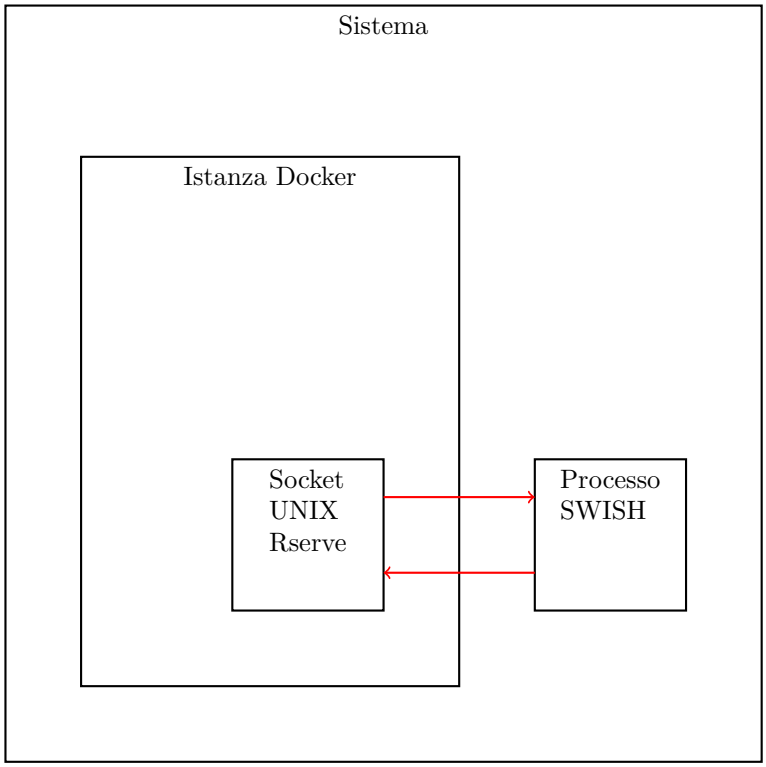
2.5 Rserve ed rserve-sandbox

Rserve è un server che permette ad altri programmi di utilizzare gli strumenti messi a disposizione da R attraverso diversi linguaggi di programmazione, senza dover inizializzare R o linkarne la libreria [28]. Rserve è il software attraverso il quale rserve_client può interfacciarsi con R.

Per migliorare la sicurezza Rserve non viene mai eseguito direttamente, ma all'interno di un sistema di virtualizzazione chiamato Docker. Inoltre non viene mai esposto un socket *TCP/IP* ma solo un socket di tipo *UNIX*. A differenza di socket TCP/IP, in un socket UNIX tutto lo scambio di dati avviene all'interno del *kernel* del sistema operativo [29]. L'applicazione di quest'ultimo aspetto si può vedere nel listato 3 che contiene la configurazione di Rserve.

Le modalità di funzionamento di reserve-sandbox sono spiegate nello schema in figura 2.1,

Figura 2.1: Schema funzionamento di rserve-sandbox



Listato 3: File di configurazione di Rserve

```
# Copyright (c) 2016, Jan Wielemaker, Franco Masotti.
# See LICENSE file for details.

daemon disable
deamon disable
socket /home/rserve/socket
sockmod 0660
remote enable
auth disable
fileio enable
encoding utf8
interactive disable
workdir.clean enable
```

2.6 Docker

Docker è un implementazione di una virtualizzazione a livello di sistema operativo, cioé permette l'esistenza di molteplici istanze di ambienti isolati che operano in modalità utente [30]. Con Docker, in particolare, questi ambienti possono essere eseguiti su una qualunque macchina in modo indipendente dalla configurazione del sistema operativo e sono eseguiti dal singolo kernel del sistema operativo ospite.

In questo modo il software che è eseguito al loro interno è garantito funzionare in modo affidabile e come lo sviluppatore aveva inteso che funzionasse [31].

2.6.1 Immagini

Attraverso un documento di testo chiamato *Dockerfile* [32], Docker viene usato per costruire un'*immagine* che comprende un'insieme di programmi, file, utenti e tutto ciò che è necessario allo svolgimento di un specifico compito. Le istruzioni nel documento di testo sono nella forma `ISTRUZIONE_DOCKER argomento`. Per aggiungere utenti, gruppi ed altre operazioni viene usata l'istruzione **RUN** seguita dal percorso di una shell disponibile e infine dal comando che deve essere eseguito. Per semplicità si può anche omettere la shell ne verrà usata una di default. Per esempio

```
RUN /bin/bash -c groupadd -g newgroupid groupname
```

Altri comandi importanti sono:

- **CMD**: stabilisce l'istruzione che deve essere eseguita ad ogni avvio dell'immagine
- **ENV**: imposta variabili d'ambiente
- **USER**: effettua un cambio di utente per tutti i comandi che lo seguono
- **WORKDIR**: cambia directory
- **ADD**: copia file nell'immagine

Dopo aver costruito l'immagine con

```
docker build <nome immagine> .
```

la si può eseguire con

```
docker run <nome immagine>
```

L'istanza di un immagine è chiamata *container*.

Utilizzando i dockerfile la comunità di utilizzatori di Docker continua a creare e a distribuire nuove immagini precompilate ognuna con un particolare utilizzo. Il vantaggio di questo metodo è che l'utente finale è in grado di utilizzare subito l'immagine, senza che possano sorgere problemi nella fase di build. Lo svantaggio tuttavia è che l'immagine può essere anche molto grande (nell'ordine del Gigabyte) e quindi difficile da distribuire con i mezzi di rete più noti.

2.6.2 Configurazioni di Docker per utilizzare rserve-sandbox

Per l'utilizzo dell'ambiente R all'interno di rserve-sandbox è necessario che Rserve sia integrato opportunamente con Docker. Il dockerfile che si è utilizzato contiene infatti le instruzioni per installare e avviare Rserve. L'istruzione presente nel listato 4 consente di installare il pacchetto R ggplot2.

Listato 4: Istruzione installazione pacchetto R

```
1 RUN echo 'install.packages(c("ggplot2"), repos="http://cloud.r-project.org/", dependencies=TRUE)' >
  ↪ /tmp/packages.R \
2     && Rscript /tmp/packages.R
```

Con il comando `CMD /bin/bash Rserv.sh`, presente nel Dockerfile, Docker chiama Rserve con la configurazione presente nel listato 3 presente a pagina 6.

Dopo aver generato l'immagine si può avviare il container. Le opzioni che serve passare al comando `docker` sono:

- `--net=none`, in quanto non ci serve uno stack di rete
- `--rm`, perché non ci interessa salvare il container per un futuro utilizzo. Infatti ogni volta ne verrà creato uno nuovo. Senza questa opzione la memoria di massa si riempirebbe dopo vari avvii.
- `-v /home/rserve:/home/rserve`, che serve a montare la directory `/home/rserve` del sistema *host* in quello *guest*. In questa directory sarà infatti presente il socket di Rserve, in modo da essere accessibile al daemon di SWISH.

Nel listato 5 si trova il comando completo.

Listato 5: Comando per avviare il container Docker

```
1 docker run --net=none --rm -v /home/rserve:/home/rserve rserve
```

2.7 QVM

Per completare tutto il lavoro e testare le modifiche sui programmi man mano, è stato necessario installare l'intera piattaforma sul computer. Questo è stato possibile attraverso l'utilizzo di macchine virtuali per ragioni di comodità.

2.7.1 Macchine virtuali

Per agevolare l'installazione ed il mantenimento della piattaforma web su una istanza locale si sono utilizzate macchine virtuali con immagini di distribuzioni GNU/Linux diverse quali: Antergos, Parabola GNU/Linux-libre, Debian e Trisquel.

Qemu Virtual Machine

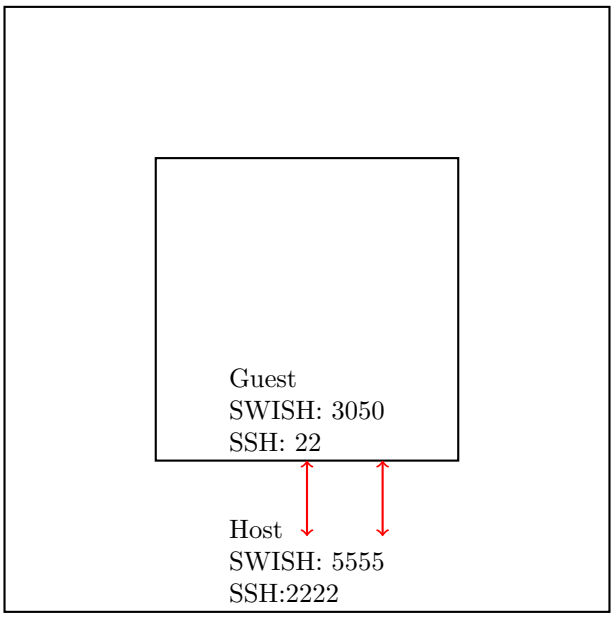
Utilizzando il programma *QEMU* [33] si è potuto scrivere uno script shell, *QVM*, in modo da gestire facilmente tutti i casi d'uso che si sono presentati [34]. Questo si è rivelato molto utile per esempio per gestire i backup dei dischi rigidi virtuali. Infatti per testare nuovi pacchetti e programmi è consigliabile partire sempre da un sistema pulito per evitare che precedenti installazioni possano alterare il funzionamento. Con QVM sono infatti sufficienti un paio di comandi per ripristinare il backup e correggere l'errore.

Lo script dà anche la possibilità di condividere file fra la macchina host e guest attraverso una directory.

Poiché l'interfaccia web di SWISH è accessibile attraverso un *port* ben definito (di default *3050*), si è fatto in modo che tale port fosse accessibile dalla macchina host con il port *5555*. Infatti, a meno che non venga fatto un *bridge* di rete la macchina virtuale ed il computer che la ospita lavorano su due sottoreti diverse. Bisogna quindi fare un *port forwarding* esplicito delle porte interessate.

Uno schema della relazione fra la macchina virtuale e l'host è disponibile nella figura 2.2 mentre nel listato 6 è riportata la pagina di aiuto.

Figura 2.2: Schema della relazione tra i port della macchina host e guest di QVM



Listato 6: Pagina di aiuto di QVM

```
Usage: qvm [OPTION]
Trivial management of 64 bit virtual machines with qemu.

Options:
    -a, --attach                connect to SSH locally
        --attach-remote         connect to SSH remotely
    -b, --backup                backup vhd
    -c, --create                create new vhd
    -d, --delete                delete vhd backup
        --delete-orig           delete original vhd
    -h, --help                  print this help
    -i, --install               install img on vhd
        --install-vnc           install img on vhd via vnc
    -n, --run-nox               run vm without opening a graphical window
                                (useful for background jobs like SSH)
        --run-nox-orig          run-orig and run-nox combined
    -s, --mkdir-shared          create shared directory
    -r, --remote                connect to a vnc instance via ssh
    -x, --run                   run vm
        --run-vnc               run vm with vnc
        --run-orig              run from original vhd
        --run-orig-vnc          run from original vhd with vnc


Only a single option is accepted.
By default, the backup vhd is run.

CC0
Written in 2016 by Franco Masotti/frnmst <franco.masotti@student.unife.it>
```

Installazione e avvio di una nuova macchina virtuale con qvm

Per l'installazione di una nouva macchina virtuale la prima cosa da fare è quella di ottenere l'immagine *ISO* della distribuzione da installare.

Successivamente si deve modificare il file di configurazione `configvmrc` inserendo il nome del file ISO.

Poi è necessario eseguire

```
$ ./qvm -c
```

per creare un nuovo disco rigido virtuale, e successivamente

```
$ ./qvm -i
```

per l'installazione.

Una volta terminata l'installazione si può installare SSH sulla macchina guest. Per ottenere una directory condivisa fra la macchina virtuale e la macchina ospite è sufficiente aggiungere un'istruzione nel file `/etc/fstab` della macchina guest. Questo si può vedere nel listato 7.

Listato 7: Comando fstab per il mount della directory condivisa

```
host_share /home/<shared_directory_path> 9p noauto,x-systemd.automount,trans=virtio,version=9p2000.L
↪  0 0
```

Ora si può creare un backup dell'hard disk virtuale con l'opzione `-b` e nel caso fosse necessario rimuoverlo basterà usare l'opzione `-d`

Successivamente si avvia la macchina virtuale con

```
$ ./qvm -x
```

oppure con

```
$ ./qvm -n
```

rispettivamente a seconda che si utilizzi o non si utilizzi l'interfaccia grafica sulla macchina guest.

Infine, per collegarsi via SSH si deve usare l'opzione `-a`.

Nel listato 8 è riportata la funzione che viene chiamata nel momento di installazione della macchina virtuale.

Il codice sorgente e la documentazione di QVM si trovano su Github [34].

Listato 8: Funzione di installazione di una macchina virtuale in QVM

```
installs()
{
    local argc1="$1"
    local enable_vnc=""

    if [ "$argc1" = "vnc" ]; then
        enable_vnc="-monitor pty -vnc 127.0.0.1:0"
    fi

    qemu-system-x86_64 -m "$vm_memory" \
-device e1000,netdev=user.0 \
-netdev user,\
id=user.0,hostfwd=tcp::"$ssh_host_port"-:"$ssh_guest_port" \
-enable-kvm \
$enable_vnc \
-cdrom "$img_name" \
-boot order=d \
"$vhd_name" &
}
```

Capitolo 3

Installazione e gestione degli strumenti

3.1 Introduzione

Avendo a che fare con un insieme di programmi diversi esiste il problema della loro gestione e installazione. Per risolvere questi fattori alcuni di questi software hanno disponibili un insieme di istruzioni chiamate *pacchetti software*. Quando non si hanno a disposizione i pacchetti è necessario:

- cercare tutte le dipendenze software,
- capire come queste si installano,
- e infine sapere quali altre azioni sono necessarie

3.2 Installazione manuale

Per prima cosa bisogna installare SWI Prolog che di solito è già disponibile come pacchetto per le principali distribuzioni GNU/Linux.

Successivamente bisogna installare SWISH manualmente. Questa operazione può essere fatta sia compilando le dipendenze, sia scaricando un file zip presente nel sito ufficiale di SWI Prolog. Nel primo caso bisogna installare Bower [35] ed eseguire un comando `make`, mentre nel secondo è sufficiente l'estrazione del file all'interno della root directory dell'installazione di SWISH [36].

A questo punto è necessario installare Docker e costruire l'immagine nella quale verrà eseguito il daemon Rserve.

Bisogna poi installare i pacchetti SWI Prolog chiamati `real`, `aleph` e `cplint_r`. Questo viene fatto con l'interfaccia testuale di SWI.

A complicare ulteriormente le cose, tutti questi passaggi necessitano della creazione degli opportuni utenti e gruppi di sistema.

3.3 Andare oltre l'installazione manuale

Per affrontare il problema dell'installazione di tutti i componenti di cui si è parlato fino ad ora, si è resa quindi necessaria la creazione di vari pacchetti software per:

- l'interfaccia web SWISH,
- il daemon Rserve,
- tutte le dipendenze necessarie di Prolog ed R.

Tutto questo materiale e documentazione è disponibile nel repository swish-installer [37].
Verrà ora spiegata la terminologia ed il procedimento.

3.4 Pacchetto e gestore di pacchetti

Un pacchetto software è una raccolta di istruzioni che permettono di installare un programma in modo uniforme rispetto a tutti gli altri programmi di un sistema. Questo è possibile grazie ad un *gestore di pacchetti*. Questi gestori hanno contribuito al successo delle distribuzioni GNU/Linux (ed altre) perché consentono a chiunque di installare un programma senza aver conoscenza di metodi di compilazione. In questo modo l'installazione risulta più veloce ed essendo automatica anche meno prona a generare errori.

In generale, per costruire un pacchetto software è necessario delineare prima alcune caratteristiche:

- Quali sono le dipendenze e come si relazionano tra di loro.
- In che modo è necessario adattare il pacchetto in modo che funzioni con la specifica distribuzione.
- Il software funziona così com'è oppure bisogna creare nuovi utenti, file, etc...

Poiché l'obbiettivo è stato quello di creare pacchetti per varie distribuzioni è stato necessario un'ulteriore passaggio: separare le parti in comune che esistono tra le distribuzioni con le loro parti specifiche. Nel caso in questione le parti in comune comprendono:

- Gli script per avviare e fermare i daemon.
- Le istruzioni per installare i pacchetti Prolog.
- Le configurazioni per i sistemi di *init*, cioè quei sistemi che gestiscono l'avvio e l'arresto dei servizi. In questo caso in è presente solo *Systemd* ma si potrebbe generalizzare per qualunque sistema.

- Le azioni da fare prima, durante e dopo l'intallazione e la rimozione del pacchetto. Queste vengono date in forma generica e quindi vanno adattate con qualche modifica al caso in uso.

Una volta definite le caratteristiche di base, si è passato alla scrittura dei pacchetti veri e propri. Si è deciso di iniziare con la distribuzione *Arch Linux* per poi passare a *Debian*.

3.5 Arch Linux

Arch Linux è una distribuzione *bleeding edge* e *rolling release*. Questo significa che i pacchetti sono sempre all'ultima versione stabile e che la distribuzione non viene rilasciata come versione fissata, cosa che accade per la maggior parte delle distribuzioni. Tutto questo è spiegato nella wiki ufficiale[38]:

> *Arch Linux è una distribuzione indipendente sviluppata per i686/x86-64, abbastanza versatile ed adatta ad ogni ruolo. Il suo sviluppo si concentra sulla semplicità, il minimalismo e l'eleganza del codice. Arch è installato come sistema di base minimale (configurato dall'utente) sul quale viene costruito il proprio ambiente ideale, installando solo ciò che è necessario in base alle proprie necessità. Le GUI delle utility di configurazione non sono fornite ufficialmente e la maggior parte della configurazione del sistema viene eseguita dalla Shell e da un editor di testo. Basata sul modello rolling-release, Arch si sforza di rimanere all'avanguardia, e di solito offre le ultime versioni stabili della maggior parte dei software.*

Lo svantaggio principale di questo sistema è che il software distribuito può avere *bug* in quanto non è stato testato pienamente.

3.5.1 Struttura

Arch Linux adotta un sistema di pacchettizzazione relativamente semplice che è simile al sistema *ports* di alcuni sistemi BSD[39]:

> *Ports è il sistema usato da FreeBSD per l'automazione della generazione di pacchetti a partire dal codice sorgente. Il sistema usa un port per scaricare, scompattare, patchare, compilare e installare il software desiderato. Un port è solo una piccola directory nel computer dell'utente, chiamata come il corrispondente software che vi verrà installato, che contiene un po' di file con le istruzioni per scaricare ed installare il pacchetto dai sorgenti. Questo rende possibile la generazione del pacchetto con un semplice* `make` *o* `make install clean` *all'interno della directory port.*

Nei casi più semplici è sufficiente definire un file chiamato `PKGBUILD`. Questo contiene le istruzioni solo per fare il build del pacchetto, delle modalità di copia del contenuto nel *filesystem* e di come recuperare il software. Nel caso in questione è stato anche necessario aggiungere il file `.install` che si occupa degli

hook, cioè delle azioni pre-post installazione e rimozione. Infine sono stati aggiunti i file per *Systemd* e gli script per avviare i daemon.

3.6 Debian e Ubuntu

Debian GNU/Linux è tra le prime distribuzioni mai create[40] [41].

> *Il Progetto Debian è una associazione di persone che ha come scopo comune la creazione di un sistema operativo libero. Il sistema operativo che abbiamo creato si chiama Debian.*
>
> *[..]*
>
> *Certo, la cosa che la gente vuole è il software applicativo. Strumenti che permettano loro di ottenere che sia fatto ciò che vogliono, dallo scrivere documenti a gestire il lavoro ad eseguire giochi a scrivere altro software. Debian si presenta con più di 43000 pacchetti (software pre-compilato messo insieme in un formato adatto ad essere installato in maniera semplice sulla propria macchina), un gestore di pacchetti (APT) e altri programmi di utilità che permettono di gestire migliaia di pacchetti su migliaia di computer con la stessa semplicità con cui si installa un'applicazione. Tutto completamente free.*

Ubuntu è un'altra distribuzione GNU/Linux derivata da Debian, e basata sul ramo *unstable* di quest'ultima. Per questo motivo i pacchetti che ha disponibili sono più recenti ma meno testati [42].

3.6.1 Struttura

Il procedimento descritto nella sezione 3.4 è stato adattato anche per Debian e Ubuntu. Questo consiste nel creare la directory `debian` nella directory principale del codice sorgente all'interno della quale verranno posti tutti file necessari come `rules`, `control`. Il file `rules` è un makefile che contiene istruzioni per quanto riguarda la compilazione del pacchetto. Si può per esempio specificare di incudere i file per Systemd poiché non viene fatto di default. Il file `control` invece contiene informazioni sulle dipendenze, maintainer, descrizioni, etc... [43].

3.7 Descrizione dei pacchetti e delle funzionalità

Come accennato all'inizio di questo capitolo i pacchetti software creati comprendono SWISH, l'ambiente R, e tutto il necessario.

3.7.1 Componenti comuni

I componenti in comune a tutti i sistemi di pacchettizzazione sono:

- `run.sh`, uno script che contiene le funzioni per avviare ed arrestare il daemon. Questo file è presente in ognuno dei pacchetti creati.
- `run.pl`, è una versione modificata dell'omonimo file presente nel repository SWISH. Questo file è stato specificatamente adattato in modo che il daemon SWISH possa essere messo in *background* oltre che accessibile ad altri computer.
- `install_web_iface_deps.pl`, è uno script prolog che contiene le istruzioni necessarie per installare le dipendenze Prolog di Cplint on SWISH con supporto per R.
- I file `.service` per Systemd sono presenti per ogni pacchetto.

3.7.2 Pacchetti

Per Arch Linux sono stati scritti il maggior numero di pacchetti:

- `swish`: installa la versione ufficiale di SWISH, che non contiene modifiche
- `swish-cplint`: è una versione modificata che comprende Cplint on SWISH con supporto per R.
- `rserve-sandbox-docker`: installa l'ambiente R che permette di creare il socket gestito da Rserve. Questa è una dipendenza fondamentale di `swish-cplint`.

  Di particolare rilevanza è la creazione dell'utente `rsd`, acronimo di *rserve sandbox docker*. Questo è stato creato esclusivamente per l'interazione tra il daemon di Docker e il sistema per evitare problemi di sicurezza come il *privilege escalation*, cioè la possibilità di accedere come utente root pur non avendone i permessi [44] [45].
- `swish-cplint-bin` e `rserve-sandbox-docker-bin`: questi sono analoghi alle versioni non *bin* per quanto riguarda le funzionalità. Quello che cambia è che i componenti più importanti sono precompilati. Questo è stato necessario poiché sia `swish-cplint`, sia `rserve-sandbox-docker` sono basati su componenti che devono essere scaricati da Internet nel momento della creazione dei pacchetti quindi può succedere che qualche componente non sia raggiungibile. Se questo è il caso allora l'installazione o qualche altra funzionalità fallirebbe. In particolare, `rserve-sandbox-docker-bin` fa riferimento ad un immagine Docker precompilata e poi resa disponibile attraverso un repository git. `swish-cplint-bin` invece contiene tante dipendenze Javascript [46] che sono sparse su molti repository diversi. Come alternativa, gli sviluppatori di SWISH hanno messo a disposizione un file *zip* [47] con tutti gli strumenti precompilati, e quello che fa `swish-cplint-bin` è sfruttare questi file:

  > *Poiché installare node e bower non è una cosa immediata da fare con tutti i sistemi operativi, si possono anche scaricare le dipendenze come singolo file zip [..] È sufficiente*

estrarre il file zip, mantenendo la struttura della directory, rimanendo all'interno della directory di root di swish. In questo modo verrà creata la directory web/bower_components (..)

Per Debian/Ubuntu, viste alcune difficoltà spiegate nella sezione 3.10, si è provveduto soltanto a scrivere `rserve-sandbox-docker` e `swish-cplint`.

3.8 Build e installazione dei pacchetti

Per gestire questa situazione complessa ogni distribuzione ha un `Makefile` ed uno script shell chiamato `build_pkg.sh`. Quest'ultimo viene chiamato dal `Makefile` con parameteri diversi volta per volta. Entrambi si occupano di costruire automaticamente i pacchetti senza che l'utente abbia alcuna conoscenza dei meccanismi interni. È sufficiente infatti eseguire

```
$ make
```

per costruire tutti i pacchetti disponibili, mentre l'installazione avviene con

```
$ makepkg -sri
```

per Arch e

```
$ debuild -us -uc && dpkg -i
```

per Debian.

3.9 Distribuzione dei pacchetti

Alcune distribuzioni danno la possibilità agli utenti di caricare e condividere i pacchetti software. Nel caso in questione si veda la documentazione di swish-installer per capire quali sono e come fare per scaricare e installare i pacchetti.

3.10 Problemi riscontrati

Uno dei problemi riscontrati è che Debian ha a disposizione software che non è abbastanza recente per l'installazione dei pacchetti in questione. Questo è vero sia che si utilizzi la versione *stable* sia *unstable*.

Per quanto riguarda SWISH è infatti necessario avere sempre l'ultima versione di SWI Prolog, come è descritto nella documentazione ufficiale [48]:

È necessario installare l'ultimissima versione di sviluppo di SWI Prolog. Poiché SWISH è costantemente aggiornato e dipende da recenti librerie di sandboxing e Prolog engines, è

piuttosto comune che sia necessaria la nightly build (Windows). In alternativa è necessario fare il build del sistema dalla versione di sviluppo del repository git swipl-devel.git.

Questo sarebbe possibile soltanto ricompilando il software. Ma anche questo non è fattibile poiché mancano ancora alcune librerie fondamentali. L'unico software che sembra funzionare correttamente è Rserve sandbox.

Viste queste difficoltà si è deciso utilizzare una distribuzione derivata da Ubuntu, chiamata *Trisquel* [49]. In questo caso, abilitando i PPA di SWI Prolog [50], si è riusciti ad installare tutto con minime difficoltà. Tuttavia è presente qualche problema che inibisce l'accesso ad Rserve da SWISH.

Capitolo 4

Disegnare grafici R con Cplint on SWISH

cplint_r è una libreria per SWI Prolog che fornisce un'interfaccia fra l'interprete Prolog ed R, in modo da poter graficare i risultati generati dalla libreria cplint. Questo significa che cplint_r si occupa esclusivamente del disegno dei grafici grazie ad una libreria R chiamata ggplot2.

4.1 Tipi di grafici

cplint_r è in grado di disegnare:

- istogrammi,
- grafici a barre,
- e curve che rappresentano funzioni di densità.

4.2 Struttura

cplint_r è composta da tre diversi tipi di funzioni che in Prolog vengono chiamate *predicati*.

4.2.1 Interfaccia

L'interfaccia comprende una serie di funzioni accessibili allo sviluppatore che vuole scrivere programmi usando cplint_r. Questi predicati chiamano al loro interno, se necessario, le funzioni di cplint a cui si riferiscono. Una volta ottenuti i dati dalle funzioni di cplint, vengono chiamati dei predicati all'interno di cplint_r per gestire la parte grafica.

Per mantenere coerenza tra cplint e cplint_r, i nomi delle funzioni che fanno parte dell'interfaccia sono gli stessi, con la differenza che in cplint_r hanno il suffisso `_r`.

Uno schema della struttura è presente nel listato 9.

Listato 9: Struttura dei predicati dell'interfaccia di cplint_r

```
<predicato_interfaccia_r>(input):-
    load_r_libraries,
    <operazioni sull'input>,
    geom_<qualcosa>(<nuovo input di solito liste>),
    finalize_r_graph.
```

Nel listato 10 è riportato un predicato tratto da cplint_r.

Listato 10: Esempio di un predicato dell'interfaccia di cplint_r

```
mc_mh_sample_arg_bar_r(M:Goal,M:Ev,S,Mix,L,Arg):-
    load_r_libraries,
    mc_mh_sample_arg(M:Goal,M:Ev,S,Mix,L,Arg,ValList0),
    maplist(to_atom,ValList0,ValList),
    geom_mc_mh_sample_arg_bar(ValList),
    finalize_r_graph.
```

Le funzioni di interfaccia sono specificate all'inizio della libreria e la descrizione completa è disponibile nella documentazione online di cplint_r [51]

4.2.2 Funzioni grafiche

Le funzioni grafiche trasformano i dati in input in data frame leggibili da R e, successivamente, utilizzano ggplot2 per fare il disegno dei grafici. Tutti questi predicati hanno il prefisso `geom_` come si vede nei listati 11 e 12.

Listato 11: Struttura dei predicati per il disegno dei grafici di cplint_r

```
geom_<qualcosa>(<Liste e/o altro input>) :-
    <gestisci le liste>,
    <crea uno o più data frame con i dati ricavati dalle liste>,
    <rinomina le colonne dei data frame per evitare di usare i nomi di default>,
    <- ggplot <qualcosa>
```

Listato 12: Esempio di un predicato per il disegno dei grafici di cplint_r

```
geom_mc_sample_arg_first_bar(L) :-
    get_set_from_xy_list(L,R),
    r_data_frame_from_rows(df1, R),
    colnames(df1) <- c("names", "prob"),
    df <- data.frame(
        ids=as.character(df1$names),
        probabilities=c(df1$prob)
    ),
    <- ggplot(
        data=df,
        aes(
            x=reorder(
                ids,
                probabilities
            ),
            y=probabilities
        )
    ) + geom_bar(
        stat="identity",
        width=0.5
    )
    + coord_flip() + theme(
        axis.title.y=element_blank()
    ).
```

4.2.3 Funzioni di supporto

Le funzioni di supporto, chiamate anche *helpers* in cplint_r, servono sia per inizializzare le librerie R necessarie, sia per trasformare i dati in un formato accessibile a ggplot2, cioè in data frames. Una delle funzioni di supporto di trova nel listato 13.

Listato 13: Una delle funzioni helper di cplint_r

```
/**
 * build_xy_list(+X:list,+Y:list,-Out:list) is det
 *
 * Given to lists X and Y build an output list Out
 * in the form [X1-Y1,X2-Y2,...,XN-YN].
 */
build_xy_list([], [], []).

build_xy_list([XH|XT], [YH|YT], [XH-YH|Out]) :-
        build_xy_list(XT, YT, Out).
```

4.3 La libreria come pacchetto

Come succede con altre librerie SWI Prolog, cplint_r è distribuita sotto forma di *pack*, cioè pacchetto. Esiste infatti un gestore di pacchetti in modo che l'installazione di librerie risulti semplice ed uniforme:

```
pack_install(cplint_r).
```

In questo modo è possibile avere a disposizione la libreria con

```
use_module(library(cplint_r)).
```

Nel file di configurazione `pack.pl` che si trova all'interno del pack sono elencate tutte le dipendenze, la versione della libreria, descrizioni, ed altre informazioni di cplint_r. Il gestore di pacchetti andrà a leggere questo file per installare il tutto in una directory all'interno della *home* dell'utente.

Al momento dell'installazione cplint_r esegue dei semplici test per verificare che la libreria stessa funzioni correttamente.

4.4 Trasformazione degli esempi da C3.js ad R

Per applicare la nuova libreria cplint_r a casi reali si sono trasformati esempi da C3.js ad R.

Dopo aver definito le basi della libreria cplint_r si è provveduto a trasformare qualche esempio più semplice da C3.js ad R. Infatti si è dovuto cambiare una minima parte del codici, tranne in casi particolari, che non hanno permesso di sfruttare completamente la libreria cplint_r. Per queste occasioni si sono dovute creare funzioni specifiche per risolvere il problema. Si veda il capitolo 5 per gli esempi.

Capitolo 5

Esempi

5.1 Uso diretto di cplint_r

Nei casi di utilizzo diretto di cplint_r il codice Prolog rimane fondamentalmente invariato rispetto alla versione C3.js. Per prima cosa è neccesario includere la libreria. Bisogna poi chiamare la corrispondente funzione di cplint_r con gli stessi parameteri ad esclusione dell'*oggetto grafico* (`Prob` nell'esempio) poiché non necessario. Il listato 14 è il risultato del comando `$ diff coinmsw.pl coinmsw_R.pl` nel quale vengono evidenziate le differenze fra la versione C3.js [52] ed R [53] dell'esempio `coinmsw.pl`.

Listato 14: Differenze fra coinmsw.pl e coinmsw_R.pl

```
9,12c9
<
< :- if(current_predicate(use_rendering/1)).
< :- use_rendering(c3).
< :- endif.
---
> :- use_module(library(cplint_r)).
35c32
< ?- prob_bar(res(coin,heads),Prob).  % what is the probability that coin lands heads?
---
> ?- prob_bar_r(res(coin,heads)).  % what is the probability that coin lands heads?
37c34
< ?- prob_bar(res(coin,tails),Prob).  % what is the probability that coin lands tails?
---
> ?- prob_bar_r(res(coin,tails)).  % what is the probability that coin lands tails?
```

5.2 Casi particolari

Come detto precedentemente, sono state create alcune funzioni ad-hoc nei casi in cui non è stato possibile generalizzare.

5.2.1 gpr_R

Introduzione

Lo scopo di questo esempio è quello di prevedere il valore di alcuni punti conoscendone altri. Questo è possibile grazie al *Gaussian Process* che definisce una serie di funzioni, anche chiamate *kernel*, per fare questo tipo di analisi.

Listato 15: Chiamata di esempio di gpr_R.pl

```
?- draw_fun_pred_r(sq_exp_p).
% Given the three points
% XT=[2.5,6.5,8.5]
% YT=[1,-0.8,0.6]
% draws 5 functions predicting points with X=[0,...,10] with a
% squared exponential kernel. The graphs shows as dots the given points.
```

Varianza

A differenza della versione scritta in C3.js, nel quale è presente il calcolo della media, nella versione R è stato aggiunto anche il calcolo della varianza ottenuta seguendo i procedimenti spiegati nell'articolo ≪Gaussian Processes: A Quick Introduction≫ [54]. In breve, sono necessari i seguenti passaggi:

1. Calcolo della matrice covarianza K dei punti noti.
2. Calcolo dell'inversa di K, cioè K^{-1}
3. Calcolo di K_*, che rappresenta la covarianza tra le ascisse del punto (x_*, y_*) (con y_* incognita) e le ascisse degli altri punti (x_i, y_i) noti.
4. Trasposta di K_*, cioè K_*^T
5. La matrice K_{**}, di un solo elemento che corrisponde alla covarianza tra x_* e x_*.
6. Il valore della varianza nel punto x_* è quindi ottenuto con una serie di operazioni sulle matrici ricavate nei punti precedenti.

   Quindi: $var(y_*) = K_{**} - (K_* K^{-1} K_*^T)$

Con queste istruzioni si è in grado di trovare la varianza per uno solo dei punti, quindi bisogna eseguire le operazioni dalla 3 alla 6 con i restanti punti.

Lo pseudocodice appena visto è stato trasformato in Prolog ed è riportato nel listato 16.

Listato 16: Calcolo della varianza in gpr_R.pl

```
compute_k([],_,_,[]).

compute_k([XH|XT],X,Ker,[HK|TK]):-
  call(Ker,XH,X,HK),
  compute_k(XT,X,Ker,TK).

compute_KStar(XP,XT,Kernel,[KStar]) :-
    compute_k(XT,XP,Kernel,KStar).

compute_KStarStar(XP,Kernel,KStarStar) :-
    call(Kernel,XP,XP,KStarStar).

strip_variance_list([[V]],V).

gp_predict_variance_handler(_,[],_,_,_,[]).

gp_predict_variance_handler(XT,[XPH|XPT],K_1,Kernel,Sigma,[VarianceH|VarianceT]):-
    compute_KStar(XPH,XT,Kernel,KStar),
    transpose(KStar,KStar_T),
    compute_KStarStar(XPH,Kernel,KStarStar),
    matrix_multiply(KStar,K_1,M),
    matrix_multiply(M,KStar_T,N),
    matrix_diff([[KStarStar]],N,VarianceHTmp),
    strip_variance_list(VarianceHTmp,VarianceH),
    gp_predict_variance_handler(XT,XPT,K_1,Kernel,Sigma,VarianceT).

gp_predict_variance(XP,XT,Kernel,Sigma,Variance) :-
    compute_cov(XT,Kernel,Sigma,K),
    matrix_inversion(K,K_1),
    gp_predict_variance_handler(XT,XP,K_1,Kernel,Sigma,Variance).
```

Disegno del grafico

Nell'esempio vengono disegnate tre funzioni, ognuna rappresentate una delle possibili interpolazioni dei valori della media dei punti previsti. Queste interpolazioni sono ottenute grazie ad un polinomio di nono grado. Con l'opzione `se` viene inoltre disabilitato l'intervallo di confidenza calcolato automaticamente da ggplot2 ma non pertinente al problema.

Listato 17: Disegno delle funzioni di interpolazione di gpr_R.pl

```
formula = "y ~ poly(x,degree=9)",
method = "glm",
se = 'FALSE'
```

Per rappresentare l'intervallo di confidenza si è utilizzata l'istruzione `geom_ribbon` di ggplot2 che permette di colorare l'area del grafico interessata. L'intervallo di confidenza del 95% è dato da $\bar{y}_* \pm 1.96\sqrt{var(y_*)}$

Listato 18: Disegno dell'intervallo di confidenza di gpr_R.pl

```
geom_ribbon(
    aes(
        ymin=c(
            df1$y-(1.96*sqrt(df4$y)),
            df2$y-(1.96*sqrt(df5$y)),
            df3$y-(1.96*sqrt(df6$y))
        ),
        ymax=c(
            df1$y+(1.96*sqrt(df4$y)),
            df2$y+(1.96*sqrt(df5$y)),
            df3$y+(1.96*sqrt(df6$y))
        ),
        color=group,
        fill=group
    ),
    alpha=0.2
)
```

Per evidenziare l'errore sui dati si sono usate delle error bar definite in un intervallo di $\pm\sigma$. Questo è possibile grazie alla funzione `geom_errorbar`.

Listato 19: Disegno delle error bar di gpr_R.pl

```
Sigma is 0.3,
sigma <- Sigma,
geom_errorbar(
    aes(
        ymin=c(
            df1$y-sigma,
            df2$y-sigma,
            df3$y-sigma
        ),
        ymax=c(
            df1$y+sigma,
            df2$y+sigma,
            df3$y+sigma
        ),
        color=group
    ),
    width=0.2
)
```

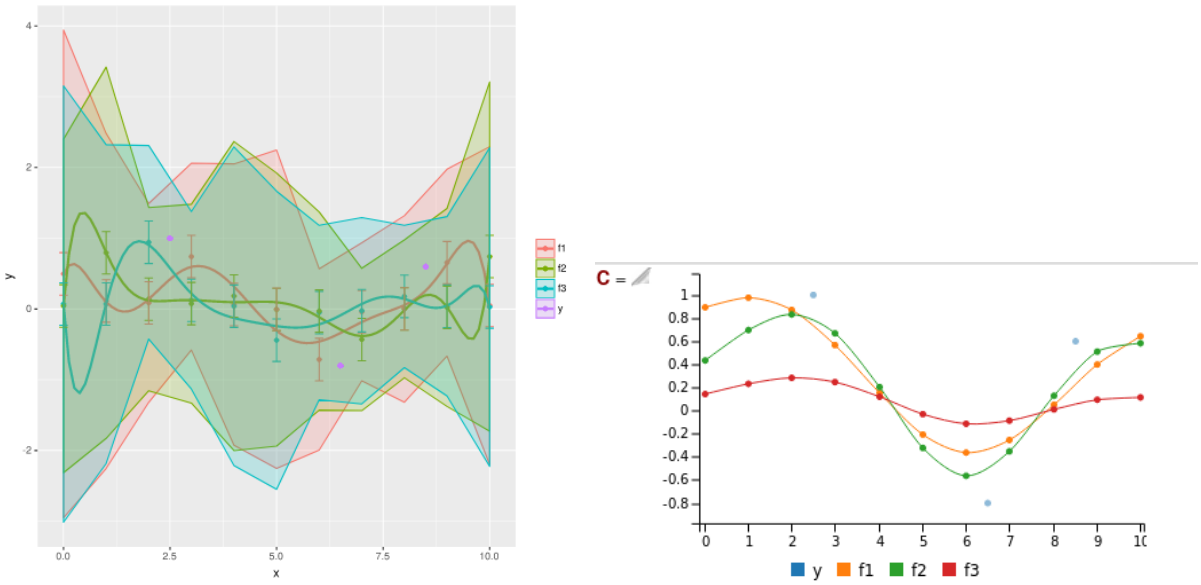
Come ultima nota sono stati anche disegnati i tre punti di training.

Listato 20: Disegno punti di training di gpr_R.pl

```
geom_point(
    aes(
        x=xN,
        y=yN,
        color="y",
        fill="y"
    )
)
```

Il risultato dell'esecuzione di gpr_R si trova nella figura 5.1 mentre il codice sorgente completo di gpr_R.pl si trova nel repository di Cplint on SWISH [55].

Figura 5.1: Risultato di `draw_fun_pred(sq_exp_p)` di gpr.pl



(a) Versione R (b) Versione C3.js

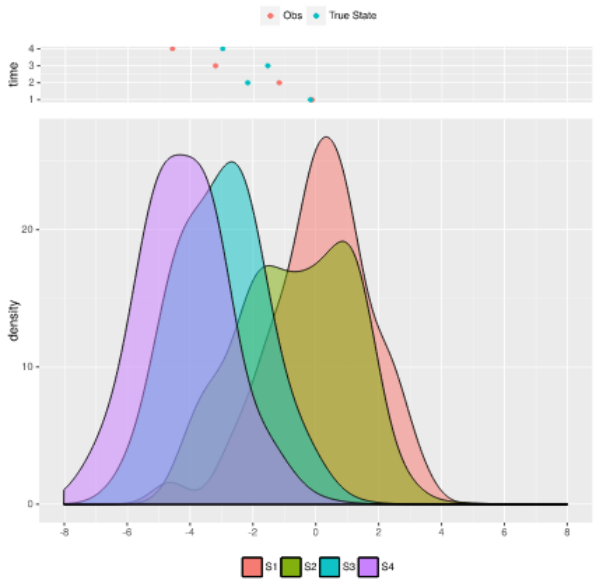
5.2.2 kalman_filter_R

Il predicato `geom_densities/7` disegna contemporaneamente due grafici in una sola figura grazie al pacchetto *gridExtra* [56]. Nella versione C3.js, invece, tutti i dati sono rappresentati sullo stesso grafico.

Il grafico superiore si riferisce a un insieme di osservazioni e di stati, mentre quello inferiore rappresenta 4 distribuzioni di densità degli stati. L'output del programma è visibile nella figura 5.2

Il codice sorgente di questo esempio si trova nel repository [57].

Figura 5.2: Risultato dell'esecuzione di `filter_sampled_par(100)` di kalman_filter_R.pl

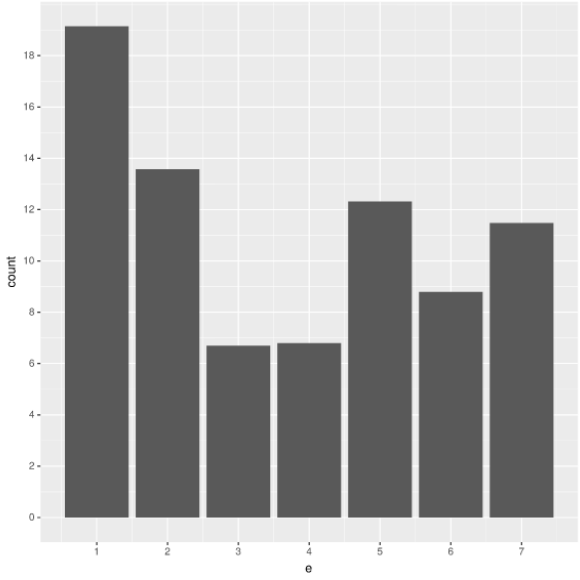


5.2.3 seven_scientists_R

Il predicato `geom_bar/1` genera un diagramma a barre della media del valore dei campioni per ogni misurazione. Poiché abbiamo 7 elementi fissi sulle ascisse dobbiamo usare `numlist(1,7,X)`, una lista da 7 elementi e `breaks=seq(0,7,1)` per evidenziare i 7 punti sul grafico. Si è inoltre usato `breaks=seq(0,max(df$val),2)` per imitare il comportamento della versione C3.js. L'output del programma è visibile nella figura 5.3

Il codice sorgente si trova nel repository di Cplint On SWISH [58].

Figura 5.3: Risultato dell'esecuzione di `chart_lw_noise(1000,E)` di seven_scientists_R.pl



Capitolo 6

Conclusioni

Inizialmente si è partiti con il creare un ambiente nel quale poter testare liberamente i software necessari. Questo ha portato alla creazione dello script QVM per una gestione semplice delle macchine virtuali.

Vista la complessità dell'installazione di Cplint on SWISH si è deciso di scrivere i primi pacchetti software. Questo è servito per poter effettuare l'installazione tutte le volte che è stato necessario, partendo da un sistema sempre pulito.

L'integrazione con R è avvenuta sia seguendo le istruzioni riportate nel repository di rserve-sandbox, sia modificando quest'ultimo con nuove istruzioni e documentazioni necessarie. Per lo stesso motivo precedente si sono dovuti scrivere i pacchetti software.

Una volta che si è verificato che l'installazione di Cplint on SWISH con supporto R era funzionante si sono incominciati a trasformare i primi esempi più semplici da C3.js ad R.

Poiché spesso venivano chiamate le stesse funzioni grafiche si è deciso di creare una libreria per evitare ripetizioni di codice.

L'obbiettivo di integrazione dell'ambiente R con Cplint on SWISH è stato quindi raggiunto. Esistono comunque margini di miglioramento: il sistema di pacchetti creato non tiene conto infatti di nuovi aggiornamenti e deve quindi essere modificato manualmente.

Bibliografia

[1] R. The R Foundation. URL: https://www.r-project.org/ (visitato il 17/09/2018).

[2] Assignment operators. URL: https://stat.ethz.ch/R-manual/R-patched/library/base/html/assignOps.html (visitato il 18/09/2018).

[3] Vectors and assignment. The R Foundation. URL: https://cran.r-project.org/doc/manuals/R-intro.pdf (visitato il 17/09/2018). An Introduction to R, Pag. 7 (13), Par. 2.1.

[4] Concatenating lists. The R Foundation. URL: https://cran.r-project.org/doc/manuals/R-intro.pdf (visitato il 17/09/2018). An Introduction to R, Pag. 27 (33), Par. 6.2.1.

[5] Other types of objects. The R Foundation. URL: https://cran.r-project.org/doc/manuals/R-intro.pdf (visitato il 17/09/2018). An Introduction to R, Pag. 11 (17), Par. 2.8.

[6] Lists. The R Foundation. URL: https://cran.r-project.org/doc/manuals/R-intro.pdf (visitato il 17/09/2018). An Introduction to R, Pag. 26 (32), Par. 6.1.

[7] Data frames. The R Foundation. URL: https://cran.r-project.org/doc/manuals/R-intro.pdf (visitato il 17/09/2018). An Introduction to R, Pag. 27 (33), Par. 6.3.1.

[8] Matrix facilities. The R Foundation. URL: https://cran.r-project.org/doc/manuals/R-intro.pdf (visitato il 17/09/2018). An Introduction to R, Pag. 22 (28), Par. 5.7.

[9] Ggplot2. Wikimedia Foundation, Inc. URL: https://en.wikipedia.org/w/index.php?title=Ggplot2&oldid=859056595 (visitato il 18/09/2018). Comparison with base graphics and other packages.

[10] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: http://ggplot2.org.

[11] The grammar of (interactive) graphics. URL: https://mattsigal.github.io/InteractiveGraphics (visitato il 17/09/2018).

[12] Ggplot2. URL: http://had.co.nz/ggplot2/ (visitato il 17/09/2018).

[13] Prolog. Wikimedia Foundation, Inc. URL: https://en.wikipedia.org/w/index.php?title=Prolog&oldid=859122393 (visitato il 15/09/2018).

[14] Jan Wielemaker, Tom Schrijvers, Markus Triska e Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. ISSN: 1471-0684.

[15] Swi prolog. URL: http://www.swi-prolog.org/ (visitato il 15/09/2018).

[16] Jan Wielemaker, Torbjörn Lager e Fabrizio Riguzzi. SWISH: swi-prolog for sharing. *CoRR*, abs/1511.00915, 2015. arXiv: 1511.00915. URL: http://arxiv.org/abs/1511.00915.

[17] Swish. URL: https://swish.swi-prolog.org/help/about.html (visitato il 15/09/2018).

[18] Jan Wielemaker, Fabrizio Riguzzi, Bob Kowalski, Torbjörn Lager, Fariba Sadri e Miguel Calejo. Using SWISH to realise interactive web based tutorials for logic based languages. *arXiv*, arXiv:1808.08042 [cs.PL], 2018.

[19] Canvas element. Wikimedia Foundation, Inc. URL: https://en.wikipedia.org/w/index.php?title=Canvas_element&oldid=856971630 (visitato il 17/09/2018).

[20] D3.js. URL: https://github.com/d3/d3/wiki (visitato il 17/09/2018).

[21] Fabrizio Riguzzi, Giuseppe Cota, Elena Bellodi e Riccardo Zese. Causal inference in cplint. *International Journal of Approximate Reasoning*, 91(Supplement C):216–232, dicembre 2017. ISSN: 0888-613X. DOI: 10.1016/j.ijar.2017.09.007.

[22] Marco Alberti, Elena Bellodi, Giuseppe Cota, Fabrizio Riguzzi e Riccardo Zese. cplint on SWISH: probabilistic logical inference with a web browser. *Intelligenza Artificiale*, 11(1):47–64, 2017. DOI: 10.3233/IA-170106. URL: http://mcs.unife.it/~friguzzi/Papers/AlbBelCot-IA17.pdf.

[23] Franco Masotti. Rserve_client. URL: https://github.com/frnmst/rserve_client/tree/82b2328e072367f5bd082d139667b9f96069ea28 (visitato il 17/09/2018).

[24] Rserve_client. URL: https://github.com/JanWielemaker/rserve_client/tree/e761657f25828dc4d493ae3bb0c8f926cf19a243 (visitato il 17/09/2018).

[25] Dealing with r data frames. URL: https://swish.swi-prolog.org/example/Rdataframe.swinb (visitato il 18/09/2018).

[26] Performance tests for transferring data. URL: http://swish.swi-prolog.org/example/Rserve.swinb (visitato il 17/09/2018).

[27] R/swish demos. URL: http://swish.swi-prolog.org/example/Rserve.swinb (visitato il 17/09/2018).

[28] Simon Urbanek. Rserve – a fast way to provide r functionality to applications. In *PROC. OF THE 3RD INTERNATIONAL WORKSHOP ON DISTRIBUTED STATISTICAL COMPUTING (DSC 2003), ISSN 1609-395X, EDS.: KURT HORNIK, FRIEDRICH LEISCH & ACHIM ZEILEIS, 2003 (HTTP://ROSUDA.ORG/RSERVE*, 2003. URL: https://www.rforge.net/Rserve/.

[29] Unix domain socket. Wikimedia Foundation, Inc. URL: https://en.wikipedia.org/w/index.php?title=Unix_domain_socket&oldid=850913211 (visitato il 17/09/2018).

[30] Operating-system-level virtualization. Wikimedia Foundation, Inc. URL: https://en.wikipedia.org/w/index.php?title=Operating-system-level_virtualization&oldid=858773658 (visitato il 17/09/2018).

[31] Docker (software). Wikimedia Foundation, Inc. URL: https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=859927458 (visitato il 17/09/2018).

[32] Dockerfile reference. URL: https://docs.docker.com/engine/reference/builder (visitato il 17/09/2018).

[33] Qemu. URL: http://www.qemu-project.org/ (visitato il 17/09/2018).

[34] Franco Masotti. Qvm. URL: https://github.com/frnmst/qvm/tree/33d11f114dd116a66c261c2786b95a36b7b114c6 (visitato il 17/09/2018).

[35] Bower. URL: https://bower.io/ (visitato il 17/09/2018).

[36] Franco Masotti. Swish. URL: https://github.com/frnmst/swish/tree/cc2c679e3c66a9ad136ee9704ba881de9ca2f884 (visitato il 17/09/2018).

[37] Franco Masotti. Swish-installer. URL: https://github.com/frnmst/swish-installer/tree/cdc16920783b3b6d13e501846b7d55c79ed8201e (visitato il 18/09/2018).

[38] Arch linux. URL: https://wiki.archlinux.org/index.php?title=Arch_Linux_(Italiano)&oldid=463564 (visitato il 17/09/2018).

[39] Cos'è un sistema ports-like? URL: https://wiki.archlinux.org/index.php?title=Arch_Build_System_(Italiano)&oldid=540056 (visitato il 17/09/2018).

[40] A brief history of debian chapter 1 - introduction – what is the debian project? URL: https://www.debian.org/doc/manuals/project-history/ch-intro.en.html (visitato il 17/09/2018).

[41] Cosa è debian? URL: https://www.debian.org/intro/about.it.html (visitato il 17/09/2018).

[42] Ubuntu. Wikimedia Foundation, Inc. URL: https://it.wikipedia.org/wiki/Ubuntu (visitato il 18/09/2018). Il rapporto con Debian.

[43] Debian new maintainers' guide. URL: https://www.debian.org/doc/manuals/maint-guide/index.en.html (visitato il 18/09/2018).

[44] Docker security. URL: http://docs.docker.com/engine/security/security/ (visitato il 17/09/2018).

[45] Docker. URL: https://wiki.archlinux.org/index.php?title=Docker&oldid=540223 (visitato il 17/09/2018). Arch Wiki.

[46] Download as zip. URL: https://github.com/SWI-Prolog/swish/tree/824c75720de82cb8a363e2ea3cd0011070a337a7 (visitato il 17/09/2018).

[47] Dipendenze web zip swish. URL: http://www.swi-prolog.org/download/swish/ (visitato il 17/09/2018).

[48] Get the latest swi-prolog. URL: https://github.com/SWI-Prolog/swish/tree/824c75720de82cb8a363e2ea3cd0011070a337a7 (visitato il 17/09/2018).

[49] URL: https://trisquel.info/ (visitato il 22/09/2018).

[50] Installing from ppa (ubuntu personal package archive). URL: http://www.swi-prolog.org/build/PPA.txt (visitato il 17/09/2018).

[51] Franco Masotti. Cplint r manual. URL: https://frnmst.github.io/cplint_r/cplint_r.html (visitato il 18/09/2018).

[52] Fabrizio Riguzzi. Coinmsw.pl. URL: https://github.com/frnmst/swish/blob/cc2c679e3c66a9ad136ee9704ba881de9ca2f884/examples/inference/coinmsw.pl (visitato il 18/09/2018).

[53] Franco Masotti. Coinmsw_r.pl. URL: https://github.com/frnmst/swish/blob/cc2c679e3c66a9ad136ee9704ba881de9ca2f884/examples/inference/coinmsw_R.pl (visitato il 18/09/2018).

[54] M. Ebden. Gaussian Processes: A Quick Introduction. *ArXiv e-prints*, maggio 2015. arXiv: 1505.02965 [math.ST].

[55] Franco Masotti. Gpr_r.pl. URL: https://github.com/frnmst/swish/blob/cc2c679e3c66a9ad136ee9704ba881de9ca2f884/examples/inference/gpr_R.pl (visitato il 18/09/2018).

[56] Baptiste Auguie. Gridextra: miscellaneous functions for "grid" graphics. URL: https://CRAN.R-project.org/package=gridExtra (visitato il 17/09/2018).

[57] Franco Masotti. Kalman_filter_r.pl. URL: https://github.com/frnmst/swish/blob/cc2c679e3c66a9ad136ee9704ba881de9ca2f884/examples/inference/kalman_filter_R.pl (visitato il 18/09/2018).

[58] Franco Masotti. Seven_scientists_r.pl. URL: https://github.com/frnmst/swish/blob/cc2c679e3c66a9ad136ee9704ba881de9ca2f884/examples/inference/seven_scientists_R.pl (visitato il 18/09/2018).