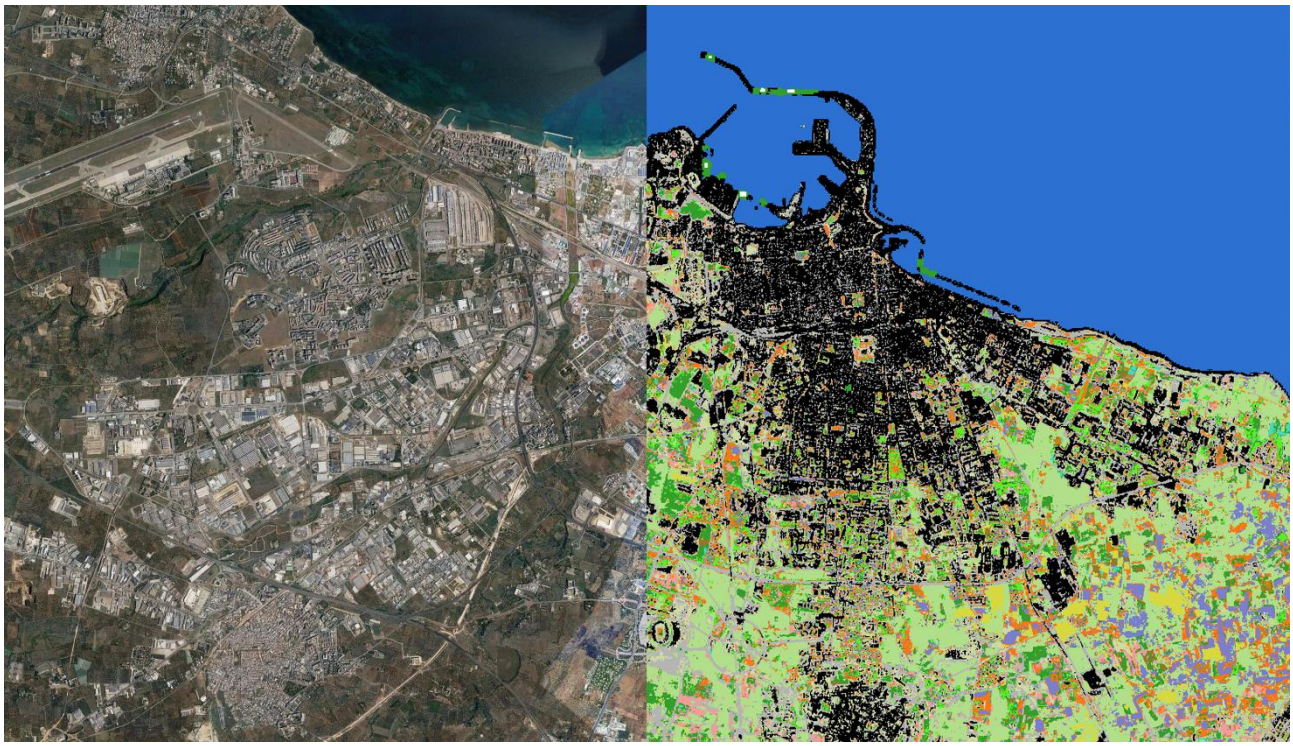
DIGITAL IMAGE PROCESSING

"IMPLEMENTAZIONE E TEST DI UN SISTEMA SVM IN R PER LA CLASSIFICAZIONE DI IMMAGINI SENTINEL-2"

A.A. 2018/2019



Prof. A. Guerriero

Dott.ssa A. Adamo

Dott.ssa C. Tarantino

Studenti:

Dott. ANIELLO Nicolò

Dott. LABORANTE Andrea

INDICE

1. INTRODUZIONE

Il tema d'anno proposto in questo lavoro è l'implementazione di un classificatore SVM in linguaggio R, al fine di classificare immagini satellitari Sentinel-2.

Inoltre, come si vedrà nei capitoli finali, sono stati effettuati una serie di test sia per comparare questo lavoro con software commerciali e sia per capire quali tipi di parametri si devono analizzare, al fine di avere risultati di classificazione migliori.

Se si vuole dare una descrizione in termini di input e output, lo script in R sviluppato, accetta in input immagini Sentinel-2, e restituisce in output un'immagine classificata, cioè un'immagine in cui, per ogni pixel, è associato un valore da {1…*k*} dove *k* è il numero di classi del classificatore SVM.

Nei paragrafi seguenti, si analizzano i componenti fondamentali che costituiscono il progetto.

1.1. LINGUAGGIO R

Il linguaggio scelto per questo lavoro è stato R. Inoltre si sono utilizzati:

- Ambiente di sviluppo *RStudio*
- Librerie:
  - *e1071*
  - *RgtSVM*
  - *Raster*
  - *Rgdal*

La descrizione sommaria e l'uso che se ne è fatto di tali librerie, si può vedere nel capitolo dedicato all'implementazione (*vedi Cap. 2*).

1.2. CLASSIFICATORE SVM

Le "*Support Vector Machines*" (*SVM*) sono un modello di apprendimento supervisionato usato nell'ambito del *Machine Learning*.

Il concetto generale è che, dato un insieme di "*samples*" (esempi) su cui si effettua un addestramento, il sistema è poi in grado di predire il valore di nuovi esempi.

Le SVM possono essere utilizzate sia per una regressione sia per una classificazione. In particolare, in questo tema d'anno, tale algoritmo è stato usato come classificatore, cioè un sistema in grado di predire a che classe appartiene un determinato "*sample*" dato in input.

Ciò che il classificatore SVM fa, è trovare la migliore "*decision boundary*" o "*hyperplane*", cioè una linea di separazione tra i vari esempi del sistema, cercando, quindi, di separare nel miglior modo possibile i vari *samples* appartenenti alle varie categorie.

La trattazione analitica di questa tecnologia è troppo complessa per essere trattata qui in dettaglio, pertanto, ci si concentra solo sugli aspetti fondamentali che riguardano questo progetto:

- **Parametri**: L'algoritmo SVM è di tipo parametrizzato, cioè non è un elemento "fisso", ma cambia a seconda dei parametri che si scelgono. In particolare:
  - **Kernel**: Funzione matematica che "trasforma" i dati di training, "proiettandoli" in un sistema a più dimensioni dove è più facile trovare la linea di separazione tra le varie classi. Esistono vari tipi di kernel, tra cui:
    - RBF: *Radial Basis Function* (**usato in questo progetto**)

        - Lineare (**usato in questo progetto**)
        - Polinomiale
        - Gaussiano
        - Etc.
    - **Parametri del relativo kernel**: Ogni kernel, a sua volta, necessita di alcuni parametri. L'esempio più esemplificativo è quello del kernel polinomiale, che, come è facile aspettarsi, ha bisogno del settaggio del parametro "d", cioè il grado del polinomio.
      Nel nostro caso, avendo utilizzato il kernel radiale, si devono impostare i parametri:
        - C (costo): Tale parametro ci permette di realizzare SVM più "rigide", cioè cercare linee di separazione che ammettono quanti meno errori di classificazione possibili (C molto grande), oppure SVM più "*soft*" cioè trovare *decision boundaries* che ammettono qualche errore (*sample misclassified*).
        - Γ (gamma): Tale parametro, parlando da un punto di vista grafico, influenza quanto distante la *decision boundary* deve trovarsi dai *samples* di sistema. Da un punto di vista *matematico*, invece, influenza quanto due *samples* del sistema devono essere considerati "simili".

      Ci si è soffermati sulla descrizione di tali parametri, perché, come si vedrà nel capitolo relativo ai test (*vedi Cap. 3*), la ricerca di quest'ultimi non è un'operazione né banale né tantomeno poco costosa.

      Il fatto che la ricerca di C e γ non sia semplice, è facilmente intuibile pensando che se tali valori si esasperano, si ottiene un sistema molto "rigido" che è molto performante sui dati di input, ma, perdendo generalità, non è abbastanza "elastico" da adattarsi ai nuovi *samples* da predire. D'altro canto, se si costruisce un sistema molto "semplice", quest'ultimo non riesce con buone performance a predire nessun tipo di nuovo *samples*.

- **Tuning dei parametri**:
  Dal momento che, come appena descritto, la ricerca dei parametri da passare ad SVM non è un'operazione banale, l'ambiente di programmazione, e, in particolare, la libreria (in linguaggio R) *e1071*, mette a disposizione delle funzioni che effettuano il lavoro in maniera automatica.
  Tale lavoro è detto "*tuning*" del modello SVM, cioè, dati i relativi *samples* in input, in maniera automatica si ricavano i parametri da passare al kernel che danno le migliori performances.
  Tali funzioni, come si può leggere nei capitoli dedicati ai test, sono molto costose dal punto di vista computazionale, pertanto, la strada di provare dei parametri "*random*" ed analizzare le rispettive performances, non è da escludere.
- **Creazione del modello**: Una volta che tutti i parametri sono disponibili, si può creare il vero e proprio modello SVM.
- **Predizione**: Con il modello SVM appena creato, si effettua la predizione sui *samples* del sistema di cui si vuole capire a quale classe essi appartengano.

1.3. SENTINEL-2

In questo progetto, come "materia prima" si sono utilizzate delle immagini Sentinel-2. Con questo si intende che tali immagini sono state utilizzate come "input" per il classificatore SVM.

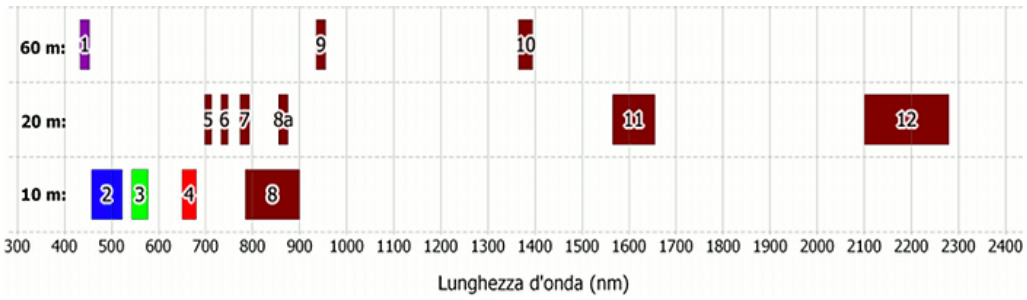
Anche per questo argomento, essendo impossibile una descrizione esaustiva, si tratteranno in maniera sommaria solo gli elementi di interesse per il tema d'anno.

1.3.1. IMMAGINI SENTINEL-2

Le immagini Sentinel-2 sono immagini prelevate da due satelliti (Sentinel-2A e Sentinel-2B) che fanno parte di un progetto atto a monitorare le aree verdi del pianeta.

Dal punto di vista tecnico, è importante notare che:

- Tali satelliti acquisiscono immagini multispettrali
  - 3 bande (R, G, B) nel visibile (risoluzione spaziale 10m)
  - 1 banda nel *NIR* – vicino infrarosso (risoluzione spaziale 10m)
  - 6 bande infrarosso (risoluzione spaziale 20m)
  - 1 banda nel blu e 2 nell'infrarosso (risoluzione spaziale 60m)
- Le immagini di una stessa superficie sono catturate ogni 5 giorni



Ogni banda fornisce informazioni utili relativamente ad una tipologia di oggetto sulla superficie terrestre. Dato lo scopo del progetto, le bande di un'immagine Sentinel-2 sono particolarmente interessanti per lo studio della vegetazione terrestre.

A titolo di esempio si riportano alcune descrizioni [1]:

Numero banda	Informazioni
1	*Atmospheric correction (aerosol scattering)*
2	*Sensitive to vegetation senescing, carotenoid, browning, and soil background; atmospheric correction (aerosol scattering)*
4	*Maximum chlorophyll absorption*
9	*NIR plateau; sensitive to total chlorophyll, biomass, LAI, and protein; water vapor absorption reference; retrieval of aerosol load and type*

Inoltre, dalla **combinazione dei valori di tali bande**, si estrapolano degli indici che sono molto utili e molto specifici per alcune categorie di oggetti terrestri. Dato il grandissimo numero di indici esistenti per un'immagine Sentinel-2, nel paragrafo successivo si analizzeranno brevemente **solo gli indici utilizzati nel progetto**.

1.3.2. INDICI UTILIZZATI

Nel seguito si elencano gli indici utilizzati nel progetto:

- **MSAVI (*Modified Soil-Adjusted Vegetation Index*)**
  - Indice usato per identificare la presenza della vegetazione nonché lo stato della stessa. È un'estensione dell'indice SAVI.
    È usato per l'analisi di immagini con una copertura di vegetazione bassa e dove è presente molto suolo (terreno non vegetato) tenendo conto della luminosità di quest'ultimo.
  - Utilizza le bande NIR, RED ed L (luminosità del suolo).
- **NDVI (*Normalized Difference Vegetation Index*)**
  - Indice relativo al benessere della vegetazione. Fa uso della banda NIR e Red, che sono sensibili al cambiamento di concentrazione di clorofilla nelle piante.
    Un limite è che, quando il livello di quest'ultima sale, può saturare l'indice NDVI che rimane costante sul valore 1.
  - In realtà, grazie alla libreria "*pheno*" *[8]*, non si utilizza l'indice NVDI in sé, ma si effettuano alcune analisi statistiche estraendo valori significativi come: *max*, *maxval*, *min*, *minval*, *greenup*, *senescence*, *mean*, *cv*.
  - Fa uso delle bande NIR e RED.
- **NDRE (*Normalized Difference Red Edge*)**
  - Indice relativo allo stato di benessere delle piante, come l'NDVI. Valori alti corrispondono a benessere della vegetazione elevato, mentre valori bassi corrispondono al "suolo", cioè nessuna vegetazione.
    Tale indice è sensibile ai livelli di clorofilla nelle foglie delle piante.
    NDRE è indicato per i periodi dell'anno in cui la vegetazione ha raggiunto livelli alti di clorofilla, per cui l'indice NDVI saturerebbe come già accennato in precedenza.
  - Fa uso delle bande NIR e REDEDGE.
- **VARI (*Visual Atmospheric Resistance Index*)**
  - Indice capace di stimare la frazione di vegetazione della zona inquadrata, in un grande *range* di condizioni atmosferiche. Tale indice, infatti, è poco sensibile a quest'ultime condizioni e quindi indipendente da esse.
- **BRIGHTNESS *[7]***
  - Indice utile nella discriminazione di strutture artificiali (edifici, strade, zone estrattive) caratterizzate da alti valori di riflettanza.
  - L'indice è il risultato di una media tra le bande R, G, B nel visibile.
- **NDBBBI (*Normalized Difference Blue Band Built-up Index*)**
  - Indice usato per le aree urbane e terreni non coltivati.
  - Fa uso delle bande BLU e SWIR.
- **NDBSI (*Normalized Difference Bare Soil Index*)**
  - Indice che ha l'obiettivo di migliorare l'individuazione di aree incolte nonché zone non vegetative. Usato per l'individuazione di strade ed edifici.
  - Fa uso delle bande NIR e SWIR.

È possibile, inoltre, raggruppare gli indici in base al loro utilizzo: MSAVI, NDVI, NDRE, VARI sono usati per la classificazione di aree vegetative, mentre, gli indici BRIGHTNESS, NDBBBI e NDBSI per le aree edificate (non-vegetative).

1.4. SOFTWARE UTILIZZATI

Nel seguito si elencano i software esterni utilizzati in questo lavoro.

1.4.1. ENVI

ENVI è un software commerciale di *image analysis* che, citando la documentazione, "*is used by GIS professionals, remote sensing scientists, and image analysts to extract meaningful information from imagery to make better decisions*".

Nel nostro caso particolare, tale software, è stato usato per una comparazione di risultati. Infatti, come si vedrà nel *Par. 3.6*, sia con il software proposto in questo lavoro, sia con ENVI, sono state lanciate due sessioni di classificazione SVM e se ne sono valutati i risultati.

1.4.2. QGIS

L'altro software utilizzato è stato QGIS: "*QGIS is a free and open-source cross-platform desktop geographic information system (GIS) application that supports viewing, editing, and analysis of geospatial data*".

QGIS è stato utilizzato sostanzialmente per motivi di comodità, in quanto:

- Una volta che lo script in R ha generato e scritto le immagini classificate su disco in formato "*.envi*", è molto comodo aprire quest'ultime in QGIS ed effettuare un'analisi visiva del risultato ottenuto.
- È un software multipiattaforma, pertanto disponibile anche per ambienti Linux, dove si è operato per sviluppare lo script R.
- L'installazione è molto semplice. Ad esempio, su un sistema Linux con distribuzione *Ubuntu 18.04*:

```
sudo apt update -y && sudo apt dist-upgrade -y

apt-get install software-properties-common apt-transport-https
dirmngr

sudo add-apt-repository https://qgis.org/ubuntu

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-key
CAEB3DC3BDF7FB45

sudo apt update


sudo apt install qgis
```

1.5. OBIETTIVI

Questo lavoro, come si vedrà sia nel capitolo relativo ai test (*vedi Cap. 3*), sia nelle conclusioni (*vedi Cap. 4*), è partito con l'obiettivo di considerare i seguenti aspetti:

- **Valutazione della classificazione di immagini Sentinel-2, effettuata con la tecnica SVM, in particolare quella contenuta nella libreria R *e1071*.**
- **Comparazione della classificazione ottenuta al punto precedente, con una ottenuta col software commerciale ENVI.**

Però, come si vedrà nel capitolo dei test, nel momento in cui si è implementato il sistema, si sono testate tante altre situazioni la cui analisi era interessante.

2. APPROCCIO PROPOSTO

In questo capitolo si analizza in dettaglio l'implementazione del sistema sviluppato.

La principale caratteristica del software è la sua suddivisione in 3 versioni:

- "***k-bands***"
  - Questa versione del software è la più elementare nonché la prima ad essere stata sviluppata. **In maniera fissa si crea il dataset prelevando le prime "k" bande da tutte le immagini di input**.
    Come è possibile capire, quindi, le features del sistema non vengono scelte con una logica ma specificate banalmente dalla variabile "*number_bands*".
    Dal momento che la semantica del dataset non influenza le prestazioni del sistema, su questa versione del software sono stati effettuati i test computazionali, in particolare i test 1,2,3,4,5,8 (*vedi Cap. 3*).
- "***2-steps***"
  - Questa versione è nata dalla considerazione che, come già detto, gli indici satellitari usati possono essere divisi in due grandi categorie: zone vegetative e non-vegetative. Si è pensato, quindi, di eseguire una classificazione a **due *step***:
    - Si è costruito il primo dataset, utilizzando solo le features relative alle zone **non**-vegetative
    - Si è effettuata la predizione dei pixel
    - Osservando le metriche della matrice di confusione, si sono mascherati tutti i pixel che ricadevano in classi con un alto valore di *f1-score*
    - Una volta "eliminati" tali pixel, si è ricreato un nuovo sistema creando un dataset con le sole features relative a zone vegetative
    - Al termine, si sono uniti sia i pixel precedentemente mascherati, sia i pixel dell'ultima predizione
- "***all***"
  - La terza variante del software è stata necessaria in quanto, oltre a considerare le features vegetative e non-vegetative separatamente, si volevano analizzare i risultati ottenuti considerandole insieme.
    La predizione SVM, quindi, avviene in un'unica passata, utilizzando un dataset creato utilizzando **tutte** le features, in particolare:
    - **12 bande** (1/mese) per gli indici: MSAVI, VARI, NDRE, NDBBBI, NDBSI
    - **6 bande** per gli indici: NDVI (estratti con la libreria *pheno [8]*)
    - **1 banda** per l'indice BRIGHTNESS (prelevando il valore massimo per ogni pixel)

2.1. LINGUAGGIO UTILIZZATO

Come precedentemente accennato, per lo sviluppo di tale sistema è stato usato:

- Linguaggio R
- Piattaforma: *x86_64-pc-linux-gnu*
- Versione: *R version 3.5.3*

2.2. AMBIENTE DI SVILUPPO

Al fine di rendere riproducibili tutti gli esperimenti, è utile fornire informazioni anche sull'ambiente di sviluppo utilizzato:

- Distribuzione: *Ubuntu 18.04*
- Kernel: *Linux 4.18.0-17-generic*
- Piattaforma: *Piattaforma: x86_64*

2.3. LIBRERIE R UTILIZZATE

Si sono utilizzate diverse librerie R e, di seguito, vengono brevemente descritte quelle principali.

Esse si possono dividere in tre grandi categorie:

- Librerie utilizzate per trattare immagini "*raster*"
  - *raster*
  - *rgdal*
- Librerie utilizzate per il modello SVM
  - *e1071*
  - *RgtSvm*
- Librerie di supporto
  - *tictoc*
  - *tools*

2.3.1. Libreria "raster" [2]

Dalla documentazione ufficiale: "*The raster package provides classes and functions to manipulate geographic (spatial) data in 'raster' format. Raster data divides space into cells (rectangles; pixels) of equal size*".

La libreria fornisce utili funzioni che permettono di gestire le immagini ad alto livello di astrazione. Si elencano le funzionalità utilizzate nel progetto:

- "*Creation of Raster* objects from file*"
- "*Raster algebra and overlay functions*"
- "*Polygon, line and point to raster conversion*"
- "*Easy access to raster cell-values*"
- "*Computation of row, col and cell numbers to coordinates and vice versa*"
- "*Reading and writing various raster file types*"

Per meglio comprendere le righe di codice che seguiranno, si ricorda la tipologia di immagine che il sistema prende in input:

- Immagine Sentinel-2
- Formato "*.envi*"
- Ogni immagine ha diverse bande (es. 132)

Si mostra brevemente come è stata utilizzata la libreria *raster*:

- Caricamento di un'immagine in formato "*envi*":

```
image <- stack("image_name.envi")
```

- Caricamento di una singola banda di un'immagine:

```
banda <- raster(image, layer=i)
```

- Prelievo del valore di ogni singolo pixel della banda:

```
valori <- getValues(banda)
```

- Caricamento di uno *shapefile*:

```
shape_file_training <- shapefile("nome_shapefile.shp")
```

- Estrazione dei valori dei pixel di uno *shapefile* che ricadono in un'immagine:

```
cells_extract <- extract(banda, shape_file, cellnumbers=TRUE)
```

- Inserimento di nuovi valori associati ai pixel dell'immagine:

```
new_image <- setValues(image, new_values)
```

- Salvataggio di un'immagine *raster* con determinate opzioni su disco:

```
image_hdd <- writeRaster(image, "new_name", format="ENVI",
bandorder='BIL')
```

2.3.2. *Libreria "rgdal" [3]*

Libreria di supporto a quella "*raster*". Infatti, quest'ultima, utilizza le funzioni "*shapefile()*" che non sono altro che wrapper delle funzioni "*readOGR()*" della libreria "*rgdal*".

Sostanzialmente, il package "*rgdal*" fornisce delle interfacce per le librerie "*GDAL*" e "*PROJ4*", che, rispettivamente, servono per leggere vari formati di dati geografici e per la conversione di coordinate geografiche in formato diverso.

2.3.3. *Libreria "e1071" [4]*

Il package "*e1071*" è una libreria fondamentale di tutto l'ecosistema matematico in R, pertanto, fornire una descrizione unica è pressoché impossibile.

La definizione ufficiale, infatti, è abbastanza generica: "*Misc Functions of the Department of Statistics, Probability Theory Group*".

Le funzioni che si sono andate ad utilizzare nel progetto, sono tutte riguardanti la gestione del modello SVM:

- *Tuning* di un modello SVM:

```
tuned_svm <- tune.svm(
              x=dati_training,
              y=factor(dati_target),
              kernel ="radial",
              gamma = 2^(-5:5), cost = 2^(-5:5), #Grid-Search
              tunecontrol = tune.control(cross = 10) #Cross-
Validation

)
```

Con il comando precedente si vuole effettuare l'operazione di "*tuning*" di un modello SVM.
Questa operazione, per quanto detto nel paragrafo introduttivo 1.2, è di vitale importanza, in quanto, ogni sistema SVM ha bisogno che siano specificati dei parametri (*C* e *gamma*).

Tali parametri, possono sia essere impostati manualmente in maniera random, oppure, anche se l'operazione non è particolarmente leggera dal punto di vista computazionale, si può lasciare alla libreria e1071 il compito di trovare **i migliori valori** di *C* e *gamma* per quel determinato dataset di input.
Questa operazione viene fatta con la cosiddetta "***grid search***", cioè vengono dati dei ***range*** di *C* e *gamma*, e il metodo "*tune.svm()*" testa ogni combinazione di quest'ultimi per trovare quella con la migliore performance.
Inoltre, è possibile anche notare che sul dataset di training viene effettuata la ***cross-validation***.

- Creazione di un modello SVM:

```
svm_model <- svm(x=dati_training,
                 y=factor(dati_target),
                 type = "C-classification",
                 kernel ="radial",
                 gamma = gamma_value,
                 cost = C_value)
```

Una volta che tutti i parametri del modello SVM sono stati decisi, è possibile creare quest'ultimo.
Si può notare:

  - *type = "C-classification"*: si specifica che il modello SVM da creare, deve essere un classificatore. Questo perché può essere usato anche come uno strumento di regressione lineare.

- Predizione:

```
prediction <- predict(svm_model,dati_prediction)
```

Con tale comando si effettua l'operazione di predizione.
Al sistema SVM vengono passati dei dati da predire e, dato che si è creato un classificatore, **ogni sample di input, verrà classificato come appartenente ad una determinata classe**.

2.3.4. Libreria "RgtSvm"

Sin dal primo momento in cui si sono iniziate a gestire le immagini di input, ci si è resi conto che si aveva a che fare con grandi quantità di dati in memoria RAM.

Infatti, a titolo esemplificativo, si mostrano le grandezze di alcuni dataset:

- Dataset completo: 2.6 GB
- Dataset Training: 24.4 MB
- Dataset Test: 7.4 MB
- Dataset NoData: 0.3 MB
- Dataset Prediction: 2.6 GB

Oltre alle dimensioni statiche di ciascun database**, va anche considerato che** in molte operazioni, come per esempio il tuning SVM dove si effettua la cross-validation, **il sistema deve creare più copie** dello stesso per effettuare eventuali comparazioni.

Si è pensato, perciò, di utilizzare anche **librerie diverse dalla "*e1071*"**, sia per agevolare i calcoli, sia per testare e comparare il comportamento di quest'ultime.

In particolare, si è scelto di utilizzate la libreria ***RgtSvm [5]*** che sfrutta il calcolo parallelo nonché la GPU della scheda video per effettuare i calcoli.

Tale libreria, però, necessita di requisiti stringenti come:

- Framework *CUDA* e quindi di una scheda video *Nvidia*
- Sistema operativo *Linux* / *Mac OSX*

Le caratteristiche positive che è possibile attribuirle *[5]*:

- Uso del processore grafico GPU anziché CPU.
- Molte azioni che occorrono nella creazione di un modello SVM si sposano bene con il calcolo parallelo. Basti pensare alle operazioni effettuate dal kernel, dove, per ogni pixel in input, viene effettuata la stessa tipologia di operazione. Questo è esattamente il caso in cui la tipologia di calcolo *SIMD* (*Single Instruction Multiple Data*) dei processori paralleli trae enorme vantaggio.
- A differenza della libreria *e1071*, permette di passare come dati in input, la *reference* ad una matrice, anziché copiare tutti i valori presenti in essa. Ciò comporta un **notevole** risparmio di tempo.
- **L'elemento più fondamentale, è, senza alcun dubbio, la completa compatibilità con *e1071*. Con ciò si intende che tutte le funzioni in *RgtSvm* hanno la stessa interfaccia di quelle *e1071***. In tal modo, tutti gli script e tutti i *developer* che sanno usare le funzioni della libreria standard, non devono cambiare alcunché.

Nel capitolo dedicato ai test (*vedi Cap. 3*) si approfondirà in maniera dettagliata sia il comportamento di tale libreria, sia il risparmio in termini di tempo che essa comporta rispetto alla libreria e1071.

2.3.4.1 *Installazione RgtSvm*

La libreria in questione non è presente nei repository *CRAN*, pertanto non è possibile installarla con il semplice comando "*install.packages()*".

Per tale motivo e, dato che l'installazione non è banale, **si fornisce di seguito una guida su come installare sia la libreria *RgtSvm* sia il framework *CUDA* 9.0**:

```
#---------------PREREQUISITES----------------#
sudo apt-get install freeglut3 freeglut3-dev libxi-dev libxmu-dev
gcc-multilib g++-multilib


#--------------GRAPHIC DRIVERS---------------#
sudo apt purge nvidia*
sudo apt autoremove
reboot
sudo add-apt-repository ppa:graphics-drivers/ppa
sudo apt update
sudo apt upgrade
sudo ubuntu-drivers autoinstall
reboot
#check installation with: nvidia-smi

#---------------INSTALL GCC v.6---------------#
#CUDA 9 necessita di gcc-6
sudo apt install gcc-6 gcc+-6
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-6
10
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-6
10

#---------------INSTALL CUDA-----------------#
#download cuda*.run from https://developer.nvidia.com/cuda-toolkit:
sudo chmod +x cuda*.run
./cuda*.run --override --silent --toolkit

#---------SETTING ENVIRONMENT CUDA-----------#
echo -e "export
LD_LIBRARY_PATH=\"$LD_LIBRARY_PATH:/usr/local/cuda/lib64:/usr/local
/cuda/extras/CUPTI/lib64" >> ~/.bashrc
echo -e "export CUDA_HOME=/usr/local/cuda" >> ~/.bashrc
source .bashrc
sudo ldconfig

#---------------INSTALL BOOST----------------#
sudo apt-get install libboost-all-dev

#---------SETTING ENVIRONMENT BOOST----------#
echo -e "export BOOST_HOME=/usr/include/boost" >> ~/.bashrc
source .bashrc
sudo ldconfig

#---------------INSTALL RGTSVM---------------#
#In RStudio:
#      install.packages("bit64")
#      install.packages("snow")
#      install.packages("SparseM")
#      install.packages("Matrix")
git clone https://github.com/Danko-Lab/Rgtsvm.git
cd Rgtsvm
make R_dependencies
R CMD INSTALL --configure-args="--with-cuda-home=$CUDA_HOME --with-
boost-home=/usr" Rgtsvm
```

2.4. FUNZIONALITÀ PRINCIPALI DEL SISTEMA

I paragrafi che seguono hanno una notevole importanza in quanto spiegano le funzionalità principali che sono state implementate nel sistema. **Inoltre, visto che il software presenta 3 versioni, nel paragrafo *2.6* si analizzeranno le sottili differenze che caratterizzano ciascuna variante**.

Dal momento che l'ordine con cui si eseguono le varie azioni è fondamentale, si è deciso di descrivere le varie componenti del software così come si susseguono cronologicamente all'interno dello script.

2.4.1. VARIABILI DI SISTEMA (main.R)

Per rendere agevole l'utilizzo del software ad utenti che non vogliono addentrarsi nel codice, si è pensato di predisporre una parte iniziale al settaggio di alcune **semplici variabili booleane**, che in base al valore che viene impostato, **attivano o disattivano alcune funzionalità del sistema**.

In particolare, le variabili sono:

- *instructions*: Se impostata a *TRUE* permette di mostrare le istruzioni sull'utilizzo del software quando si avvia.
- *debug*: Se impostata a *TRUE* permette di mostrare a video eventuali informazioni di debug, come ad esempio, il tempo che ciascuna fase ci impiega.
- *cuda_installed*: Se impostata a *TRUE* permette di utilizzare la libreria *RgtSvm* e quindi sfruttare il calcolo parallelo su *GPU.*
- *tuning*, *C_standard_value*, *gamma_standard_value*: Se impostata a TRUE, la variabile "*tuning*" permette di effettuare il *tuning* del modello SVM. Altrimenti, i valori usati saranno "*C_standard_value*" e "*gamma_standard_value*".

2.4.2. CARICAMENTO DEI MODULI DEL SISTEMA (main.R)

Dal momento che il software è stato sviluppato cercando un approccio modulare, le varie funzionalità del sistema sono stati inglobate in diversi file ".*R*".

Prima di utilizzare le funzioni dei vari moduli, quindi, bisogna caricare i relativi file.

2.4.3. CONTROLLO DELLE DIPENDENZE (checkDependencies.R)

Si è pensato di rendere il software quanto più automatizzato possibile. Pertanto il codice relativo al file "*checkDependencies.R*", **carica tutte le librerie che il sistema necessita e, se esse non sono installate, provvede a farlo**.

2.4.4. CONTROLLO DELLE DIRECTORY (checkDirectory.R)

Lo script fa uso di diverse directory, in particolare:

- *images/*: Contiene le immagini di input
- *training/*: Contiene i dati (*shapefile*) di training
- *test/*: Contiene i dati (*shapefile*) di test
- *final_images/*: Contiene le immagini finali che il sistema genera

Il sistema effettua un controllo su quest'ultime, e, se non esistono, vengono create.

2.4.5. CARICAMENTO FILE DI TRAINING E DI INPUT (checkTrainingTest.R) (loadInput.R)

In questa sezione il sistema analizza tutte le immagini di input, tutti i dati (*shapefile*) di training e tutti i dati (*shapefile*) di test. Di tutti questi dati, poi, crea delle liste contenenti i nomi dei rispettivi file. Quest'ultime saranno poi usate per caricare fisicamente i file.

Inoltre, il sistema genera una tabella "*corrispondenza_classi*" in cui sono inserite tutte le categorie in cui il sistema dovrà classificare i pixel.

2.4.6. VALIDAZIONE IMMAGINI (validateImage.R)

Le immagini di input vengono "validate". Con questo si intende che vengono fatti dei controlli al fine di verificare che tali immagini siano compatibili, cioè che i seguenti dati siano uguali per tutte le immagini:

- Dimensione x (numero pixel)
- Dimensione y (numero pixel)
- Risoluzione spaziale (es. in Sentinel-2 10m x 10m)

2.4.7. COSTRUZIONE VARI DATASET (createDataset.R) e (splitDataset.R)

Innanzitutto viene creato un dataset contenente tutti i pixel come **righe** e tutte le features (bande delle immagini di input) che si sono scelte come **colonne**. Il dataset ha dunque la seguente forma:

n° cella	Feature 1	…	Feature n	Target Y
…	…	…	…	…
…	…	…	…	…

In seguito, si caricano gli shapefile i cui nomi sono stati ricavati in *Par. 2.4.5*. Con "caricamento degli shapefile" si intende che, in riferimento ad un singolo shapefile che può essere considerato un generico poligono, **si ricavano quali pixel delle immagini di input "ricadono" all'interno di questi poligoni**.

Una volta che sono stati individuati tutti i pixel di Training, tutti i pixel di Test e, per differenza, tutti i pixel che dovranno essere classificati (predetti) dal modello SVM, si creano i dataset:

- *datasetTraining*: che sarà usato per il training del modello SVM
- *datasetTest*: che sarà usato per il test del modello SVM
- *datasetPrediction*: che sarà usato per la predizione del modello SVM

Inoltre, il software prevede la presenza e la gestione dei cosiddetti shapefile "***NO_DATA***".

I pixel corrispondenti a tale categoria, sono in realtà pixel da "mascherare". Quindi essi sono prelevati, esclusi da tutto il processo di *machine learning*, e reinseriti nel momento in cui si va a creare l'immagine finale.

Se essi sono presenti, quindi, viene creato anche il dataset:

- *datasetNoData*

2.4.8. TUNING DEL MODELLO SVM (main.R) [OPZIONALE]

Nel caso la variabile "*tuning*" sia settata a "*TRUE*", viene effettuato il tuning del modello SVM. Se si decide di non effettuare tale operazione, vengono passati al kernel dei valori di default.

2.4.9. CREAZIONE DEL MODELLO SVM (main.R)

Creazione del modello SVM a cui si passano o i valori ottimali ottenuti dalla frase precedente di tuning, oppure i valori di default impostati nella sezione di variabili di sistema (*vedi Par. 2.4.1*).

2.4.10. TEST, MATRICE DI CONFUSIONE E METRICHE DEL SISTEMA (main.R)

Al modello SVM creato viene passato il datasetTest, in cui, per ogni pixel, si conosce il valore della variabile target associato. Quindi, il test del sistema consiste nella predizione di quest'ultimi da parte di SVM.

Vengono quindi valutati, classe per classe, i pixel che sono stati classificati correttamente e quelli che sono stati classificati erroneamente. Questo risultato, in forma tabulare, è conosciuto col nome di "***confusion matrix***".

Su tale matrice, poi, sono calcolate usa serie di metriche che permettono di valutare in maniera quantitativa il sistema:

- *Accuracy* (*Overall Accuracy*)
- *Precision*
- *Recall*
- *F1 – Score*

2.4.11. PREDIZIONE (main.R)

Una volta che il sistema è stato creato e testato, viene effettuata l'operazione di predizione. Con ciò si intende che al sistema SVM vengono passati tutti i samples da classificare. Nell'esempio specifico di questo lavoro, ciò corrisponde ai pixel delle immagini di input.

2.4.12. RICOSTRUZIONE DELL'IMMAGINE (main.R)

L'operazione finale, è quella di ricostruzione dell'immagine.

Tecnicamente, dal momento che, per ogni pixel, si ha un valore classificato, tali valori si inseriscono al posto di quelli originali in una delle immagini di input.

Questa operazione di sfruttare un'immagine preesistente è funzionale ed allo stesso tempo molto veloce. Infatti, anziché ricostruire un'immagine da zero, si prende una già completa, e, banalmente, al posto dei valori originali si sostituiscono quelli ricavati da SVM.

Infine, tale immagine viene scritta su disco nella directory "*final_images/*" ed in formato "*.envi*".

2.5. DIFFERENZE SIGNIFICATIVE TRA LE TRE VERSIONI DEL SOFTWARE

Come già accennato all'inizio di questo capitolo, il software viene presentato con 3 versioni differenti. In tutte queste varianti, le funzionalità principali sono uguali e sono state già esposte nei paragrafi precedenti.

In questo, invece, si vogliono sottolineare solo le differenze a livello di funzionalità tra le 3 versioni.

2.5.1. VERSIONE "k-bands"

La versione "*k-bands*" non rappresenta alcuna particolarità rispetto alle fasi già precedentemente descritte nei *Par. 2.4.1 – 2.4.12*.

2.5.2. VERSIONE "all"

Il problema della versione "*k-bands*" sta nel fatto che non c'è nessuna logica nel prelevare in maniera fissa un certo numero di bande dalle immagini di input.

Si è pensato, dunque, di creare la versione denominata "*all*" che esegue tutti gli step dei *Par. 2.4.1 – 2.4.12*, come la versione "*k-bands*", ma **con la differenza che le bande vengono prelevate con una certa semantica**:

```
#Prelevo le bande da MSAVI, VARI, NDRE, NDBBBI, NDBSI
image_msavi <- stack(paste(image_dir,"MSAVI.envi",sep="/"))
image_vari <- stack(paste(image_dir,"VARI.envi",sep="/"))
image_ndre <- stack(paste(image_dir,"NDRE.envi",sep="/"))
image_ndbbbi <- stack(paste(image_dir,"NDBBBI.envi",sep="/"))
image_ndbsi <- stack(paste(image_dir,"NDBSI.envi",sep="/"))
for(i in c(1,13,22,34,45,55,67,78,90,99,109,120)){
  matrice <- cbind(matrice,getValues(raster(image_msavi, layer=i)))
  matrice <- cbind(matrice,getValues(raster(image_vari, layer=i)))
  matrice <- cbind(matrice,getValues(raster(image_ndre, layer=i)))
  matrice <- cbind(matrice,getValues(raster(image_ndbbbi, layer=i)))
  matrice <- cbind(matrice,getValues(raster(image_ndbsi, layer=i)))
}
#Prelevo le bande da PHENO NDVI
image_pheno <- stack(paste(image_dir,"PHENO_NDVI.envi",sep="/"))
for(i in c(1,2,3,4,5,6,7,8)){
  matrice <- cbind(matrice,getValues(raster(image_pheno, layer=i)))
}
#Prelevo il valore MAX di Brightness: per ogni pixel il massimo valore assunto in una
delle bande
image_brightness <- stack(paste(image_dir,"BRIGHTNESS.envi",sep="/"))
raster_stackapply <- stackApply(image_brightness,
indices=rep(1,nlayers(image_brightness)), fun=max)
matrice <- cbind(matrice,getValues(raster_stackapply))
```

2.5.3. VERSIONE "2-steps"

In questa variante del sistema si effettua, di fatto, due volte la predizione SVM. Quindi le fasi descritte dai *Par. 2.4.1 – 2.4.11* sono effettuate una dopo l'altra per due volte.

Le differenze sostanziali, però, sono che:

- Nel primo step il dataset è creato utilizzando **solo** le features "**non-vegetative**", mentre nel secondo step il dataset utilizzando **solo** le features "**vegetative**".
- Tra uno step e l'altro viene effettuato il ***masking*** di alcune classi.
  Questo vuole dire che, innanzitutto, attraverso i valori di *f1-score*, vengono selezionate le classi che sono state predette ottimamente dal sistema nel primo step. In secondo luogo, tutti i pixel che sono stati predetti con valori appartenenti a quelle classi vengono salvati temporaneamente.

Poi tali classi vengono eliminate dal sistema e si effettua un nuovo ciclo completo di azioni descritte nei *Par. 2.4.1 – 2.4.11*. È bene ricordare che i pixel predetti in quest'ultima fase, apparterranno solo a classi **non** contenute in quelle che sono state mascherate.
Infine, nella fase finale di ricostruzione immagine, si uniscono sia i pixel relativi alla seconda predizione, sia quelli che sono stati temporaneamente salvati al termine della prima.
Di seguito, si mostrano le righe di codice significative:

```
#Salvataggio dei pixel da mascherare
datasetZoneEdif <- prediction_step1[prediction_step1[,ncol(prediction_step1)]==2
                                    |prediction_step1[,ncol(prediction_step1)]==9
                                    |prediction_step1[,ncol(prediction_step1)]==11
                                    |prediction_step1[,ncol(prediction_step1)]==12
                                    |prediction_step1[,ncol(prediction_step1)]==13
                                    |prediction_step1[,ncol(prediction_step1)]==14,
                                    ]

#Eliminazioni classi già predette dal sistema, necessario per STEP 2
corrispondenza_classi <- corrispondenza_classi[-index_to_remove,]
```

2.6. OTTIMIZZAZIONI EFFETTUATE

La grandezza dei dati gestiti, come si è già fatto notare nel *Par. 2.3.4*, ha comportato la necessità di effettuare delle ottimizzazioni a livello di codice, al fine di rendere i tempi computazionali gestibili.

A titolo di esempio, si riporta come si è riusciti ad abbattere in maniera **molto significativa** i tempi di creazione dei dataset di *training*, *test*, *nodata*.

2.6.1. METODO NON OTTIMIZZATO

Se si dovesse seguire un modo di procedere "standard", una volta ottenuta la variabile "*matrice*" che contiene il dataset completo, cioè di tutti i pixel, per creare il dataset di test si procederebbe nel seguente modo:

- Ricavare dagli shapefile di test i relativi pixel
- Per ogni pixel, bisognerebbe cercare quest'ultimo nel dataset completo ed impostare la variabile target. Ciò, dal punto di vista pratico, corrisponderebbe ad indicizzare ogni pixel, effettuando una ricerca per riga.
- Dal punto di vista numerico, se si hanno "n" numero di bande (colonne), ed un'immagine di "x · y" pixel, allora si deve gestire una matrice di "n · x · y" elementi. Inoltre, se si denota con "k" il numero di pixel contenuti negli shapefile, si devono effettuare, dunque, "k" indicizzazioni su una matrice.
  Nel caso di questo progetto, ad esempio, ci sono 11965 pixel di test. Inoltre l'immagine Sentinel-2 ha 132 bande ed ha una risoluzione di: 1884 · 1382.
  Pertanto si dovranno effettuare 11965 indicizzazioni su una matrice di 1884 · 1382 · 132 = 343686816 elementi.

2.6.2. METODO OTTIMIZZATO

Il metodo precedentemente esposto non ha nessun difetto concettuale, però, come già accennato, data la mole di dati, comporta un elevato tempo computazionale.

Per cui, si è pensata la seguente variante:

- Si ha il dataset generale (variabile "*matrice*"):

n pixel	feature 1	…	feature n
pixel n. 1	…	…	…
pixel n. 2	…	…	…
pixel n. 3	…	…	…
pixel n. 4	…	…	…
pixel n. 5	…	…	…
pixel n. 6	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

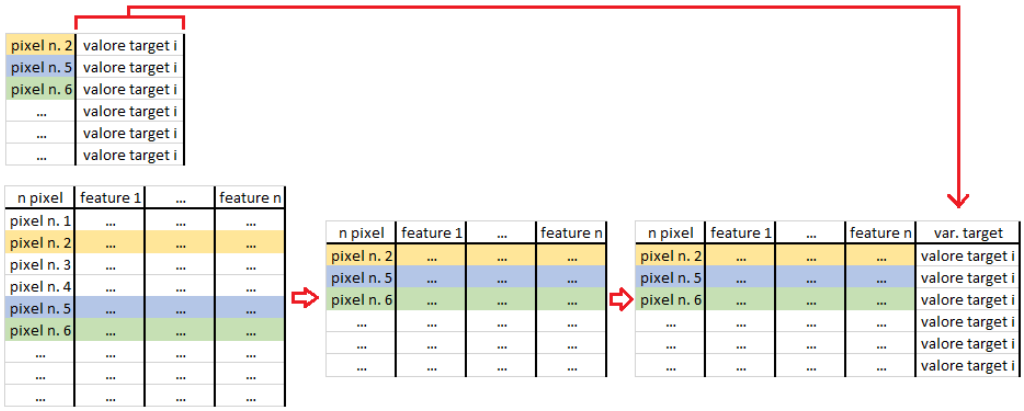
- Ricavare dagli shapefile di test i relativi pixel. Si ha la seguente colonna:

pixel n. 2
pixel n. 5
pixel n. 6
…
…
…

- A questo punto, dato che ogni shapefile è associato ad una singola categoria, semplicemente si aggiunge una seconda colonna:

pixel n. 2	valore target i
pixel n. 5	valore target i
pixel n. 6	valore target i
…	valore target i
…	valore target i
…	valore target i

- **Dal dataset generale, si prelevano le righe corrispondenti ai valori della prima colonna della tabella precedente e si aggiunge la colonna dei valori target**. In definitiva:



- **In questa maniera, al posto di 11965 indicizzazioni su una matrice di 343686816 elementi, si effettuano solo tre operazioni quasi immediate:**
  - **Creazione di una colonna di valori target**
  - **Estrazione di determinate righe da una matrice (operazione che R gestisce molto bene)**
  - **Aggiunta di una colonna (colonna dei valori target al sub-dataset estratto dalla matrice intera)**

3. TEST SUL SISTEMA

Dopo aver sviluppato l'intero sistema di classificazione, si è, ovviamente, passati alla fase di test.

Per quanto riguarda quest'ultimi, si è pensato di testare non solo gli aspetti prettamente numerici, ma di effettuare considerazioni anche sui tempi/costi computazionali e, visto che si parla di immagini, anche confronti prettamente visivi.

Nel seguito saranno presentati i diversi test effettuati, con la seguente impostazione:

- **Descrizione dello scenario**
- **Obiettivo del test**
- **Risultati numerici del test**
- **Osservazioni**

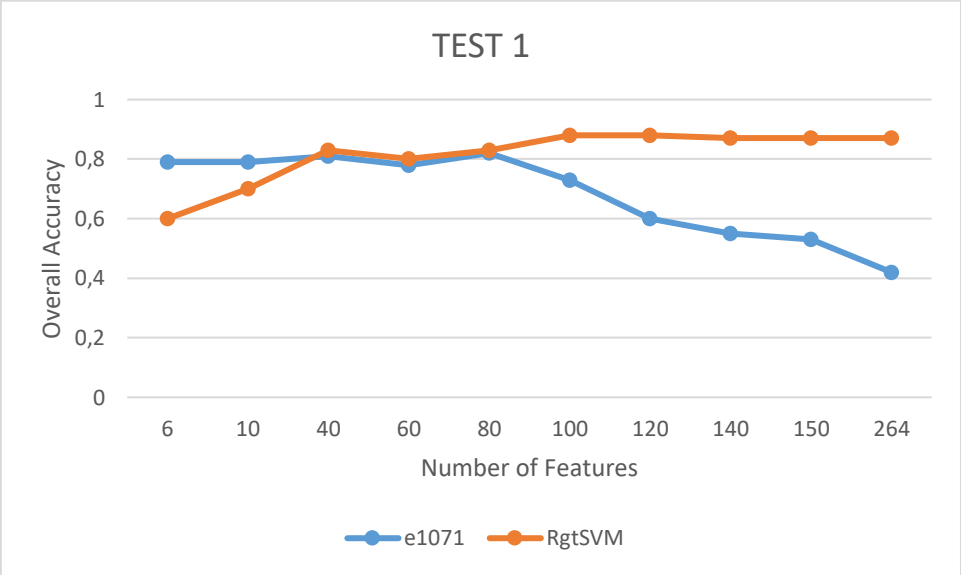
3.1. HARDWARE UTILIZZATO

Dal momento che si effettuano test sulle prestazioni computazionali, è obbligatorio fornire informazioni sull'hardware utilizzato.

- CPU
  - *Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz*
  - Numero di core: *2*
  - Numero di thread: *4*
  - Cache: *4MB*
- RAM:
  - *12 GB*
  - *SODIMM*
  - *DDR3*
- SWAP File
  - *14 GB*
- Scheda video
  - *NVIDIA GeForce 940*M
  - Memoria: *2048 MB*
  - Tipologia: *DDR3*
  - Driver video installati: *nvidia 418.56*
  - Versione CUDA installata: *9.0*

3.2. TEST 1 – CONFRONTO TRA LA LIBRERIA "e1071" E "RgtSVM" (Accuracy)

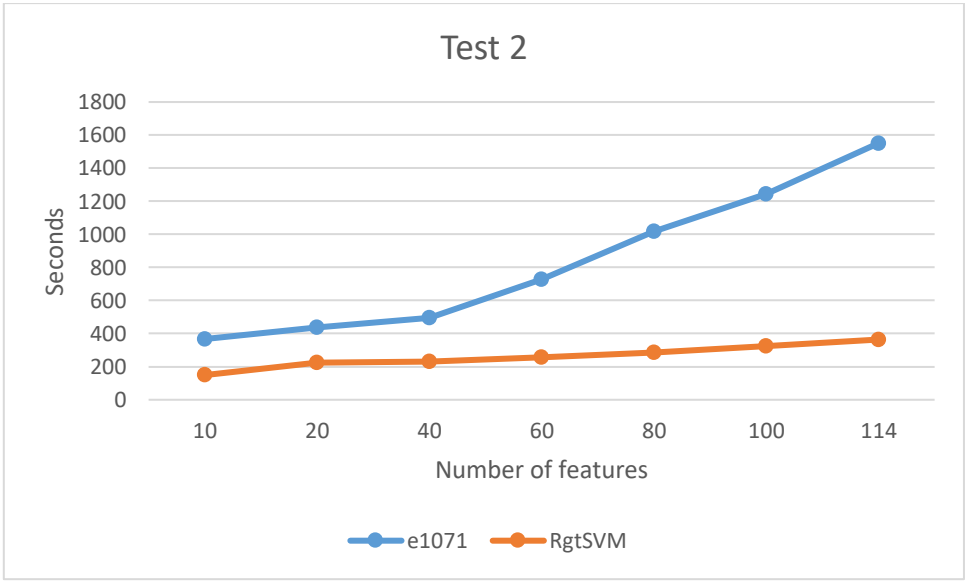
- **Descrizione dello scenario**: Visto che è stata introdotta una libreria di terze parti come "*RgtSVM*" (vedi *Par. 2.3.4*), è stato necessario testare quest'ultima per verificare la veridicità dei suoi risultati.
  In particolare, come riferimento, è stata presa la libreria "e1071", dato che quest'ultima ha molto più supporto ed è inserita nei repository ufficiali CRAN.
- **Obiettivo**: L'obiettivo del test è quello di accertare **se** la **libreria "*RgtSVM*" si comporta come la "*e1071*"**.
- **Risultati numerici del test**: Per verificare che le due librerie abbiano lo stesso comportamento, **si è scelto un insieme di bande (6,10,40,60,80,100,120,140,150,264), si sono fissati i parametri di SVM (C=1000, gamma=0.25, kernel="radial"), si è lanciato il test sia utilizzando "*e1071*" sia "*RgtSVM*" ed, infine, si sono confrontate le rispettive accuratezze**.



- **Osservazioni**: Come è possibile osservare, la libreria "*RgtSVM*" segue abbastanza fedelmente i valori di accuratezza di "*e1071*" fino ad una certa soglia di 80 features, dopo di che fornisce valori di accuratezza anche superiori. **Quest'ultima caratteristica non è da trascurare** in quanto, quando si trattano dati di grandi dimensioni, anche una piccola differenza di accuratezza, può dare risultati assai differenti. Infatti, tale differenza, si propaga sull'intera mole di dati.

3.3. TEST 2 – CONFRONTO TRA LA LIBRERIA "e1071" E "RgtSVM" (Tempi)

- **Descrizione dello scenario**: Una volta confermato che l'accuratezza della libreria "RgtSVM" non fornisce risultati sbagliati (*vedi Par. 3.2*), è necessario effettuare dei test temporali.
- **Obiettivo**: Dato che il motivo principale per cui la libreria è stata introdotta è quello degli elevati tempi di esecuzione con "*e1071*", occorre verificare che effettivamente l'utilizzo di "*RgtSVM*" comporti un vantaggio in termini computazionali.
- **Risultati numerici del test**: Per verificare il comportamento delle due librerie, **si è scelto un insieme di bande (10, 20, 40, 60 , 80 ,100 ,114), si sono fissati i parametri si SVM (C=1000, gamma=0.25, kernel="radial"), si è lanciato il test sia utilizzando "e1071" sia "RgtSVM" ed, infine, si sono confrontati i rispettivi tempi computazionali.** Parlando di "tempi" è necessario specificare che quest'ultimi **non** comprendono la fase di tuning.
  È possibile notare, inoltre, che il numero di bande selezionate in questo test, è diverso dal test precedente. In particolare ci si è fermati al numero di features = 114.
  Tale decisione **non è casuale**, ma frutto di osservazioni mandando in "*run*" il sistema con un numero crescente di features. Quello che accade alla soglia di **n=114**, è che la capacità della RAM del sistema (12 GB) non è più in grado di gestire i dataset, e, quindi, avviene il **fenomeno di swapping**, tra RAM e SWAP file. Ovviamente non si poteva tener conto di risultati numerici in tali condizioni, dal momento che il tempo computazionale calcolato, sarebbe stato "inquinato" dal tempo di swapping che, essendo aleatorio, non si sarebbe nemmeno potuto calcolare.
  Inoltre è bene notare che tale soglia è puramente soggetta alle capacità dell'hardware, quindi se il test viene effettuato su sistemi con più capacità di RAM, tale soglia sarà senza dubbio più alta.



- **Osservazioni**: Tale test conferma a pieno le aspettative. Utilizzando il calcolo parallelo basato sulla GPU, si **abbattono notevolmente** i tempi computazionali. Basti vedere l'ultima casistica, in cui si è passati da 1551 secondi (26 minuti) a 364 secondi (6 minuti). Una riduzione del **77% circa**.
  Tale risultato, prende ancora più importanza se si pensa che ci si è fermati al numero di bande pari a 114 e che, osservando il grafico, si vede come più il numero di features sale, più la forbice tra i due tempi si incrementa.
  Il test 1 insieme al test 2, portano ad un importante risultato, cioè che, dato che la libreria "RgtSVM" ha risultati di accuratezza simili a "e1071" ma che riduce

considerevolmente i tempi di computazione, è possibile utilizzarla al posto di quella standard, avendo delle performances complessivamente maggiori.

3.4. TEST 3 – PERCENTUALI DI TEMPO COMPUTAZIONALE

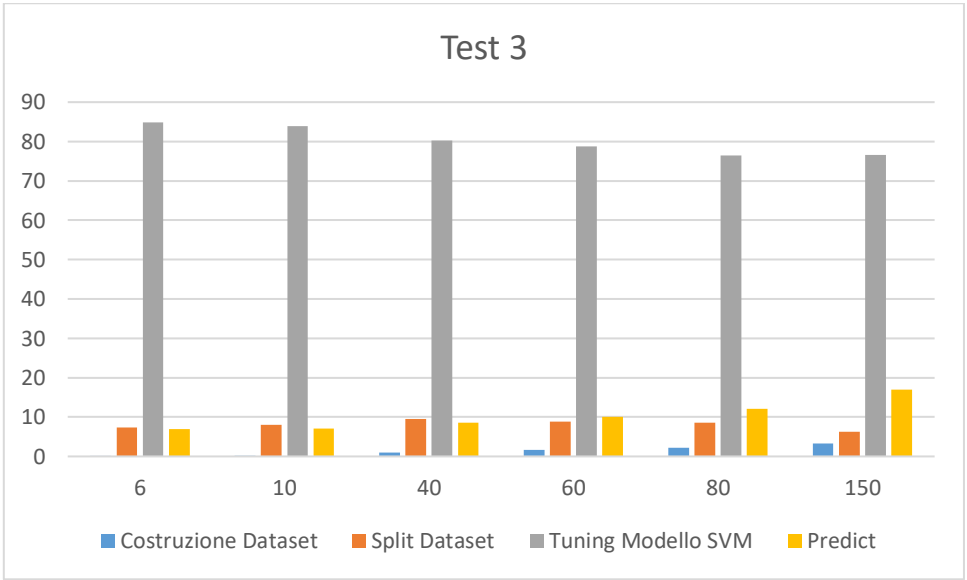
- **Descrizione dello scenario**: Dal momento che lo script prevede diverse fasi, si è pensato di indagare sui tempi computazionali e, in particolare, sulla quota parte di tempo che ciascuna fase occupa rispetto al tempo totale. Inoltre, ci si è concentrati solo sulle **fasi** del sistema **significative**.
- **Obiettivo**: Indagare sulle percentuali di tempo che ciascuna fase occupa rispetto al totale.
- **Risultati numerici del test**:
  Nella tabella seguente sono indicati, rispetto ad un diverso numero di bande, il tempo di ciascuna fase misurato in secondi.

	Tempi					
	6	10	40	60	80	150
Costruzione Dataset	1,856	3,163	14,851	25,914	38,155	75,427
Split Dataset	145,055	144,48	144,187	143,658	144,068	145,685
Tuning Modello SVM	1678,672	1503,495	1223,252	1280,237	1291,535	1768,343
Predict	138,571	127,131	130,709	164,312	204,264	392,131
Altre fasi	…	…	…	…	…	…
Tempo Totale	1977,615	1791,55	1523,58	1625,307	1689,078	2310,285

In seguito, si sono calcolate le percentuali di ciascuna fase rispetto al tempo totale:

	% Tempi rispetto al totale					
	6	10	40	60	80	150
Costruzione Dataset	0,09385	0,176551	0,974744	1,594406	2,258924691	3,264835
Split Dataset	7,334845	8,064525	9,463697	8,838822	8,529387038	6,305932
Tuning Modello SVM	84,88366	83,92146	80,28801	78,76893	76,46390516	76,5422
Predict	7,006976	7,096146	8,57907	10,1096	12,09322482	16,97327

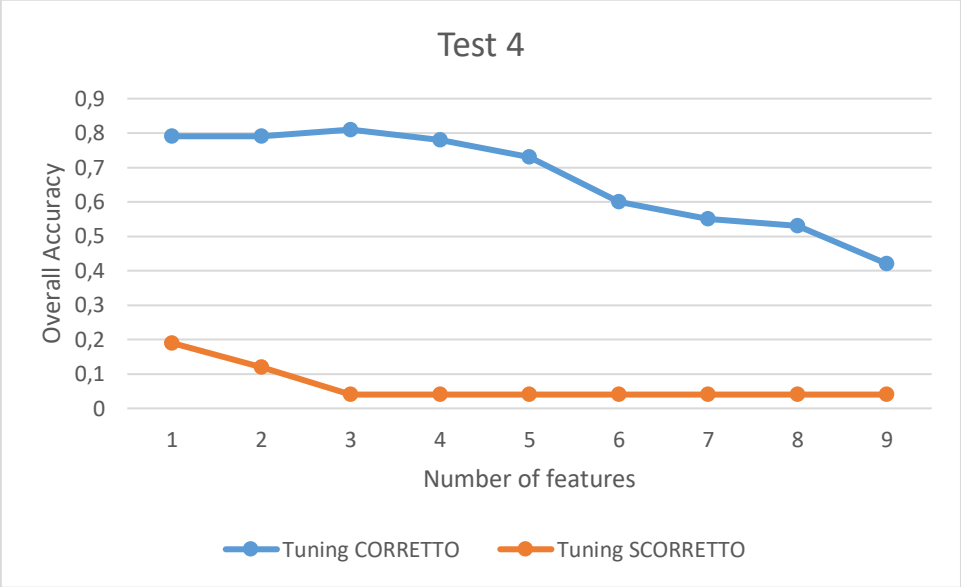
Proiettando i dati su un istogramma, si ottiene la situazione seguente:



- **Osservazioni**: Si evincono diverse osservazioni dal grafico precedente.
    - Osservazione 1: La fase di "*splitting*" non cresce. Quindi si può concludere che **non** è dipendente dal numero di features nonché dalla grandezza del dataset. Ciò è riconducibile all'implementazione di tale fase (*vedi Par. 2.6.2*).
    - Osservazione 2: Anche la fase di "Tuning del modello SVM" sembra essere indipendente dalla grandezza del dataset. Però emerge un'importante considerazione, cioè che **occupa circa il 70-80% del tempo di computazione** totale. Si conferma, quindi, l'idea fino ad ora qualitativa, che il tuning del modello SVM fosse un'operazione "costosa".
    - Osservazione 3: Le fasi di "Costruzione Dataset" e "Predict", occupano una piccola percentuale del tempo totale e sono **dipendenti** dalla grandezza del dataset.

3.5. TEST 4 – IMPORTANZA DEL TUNING DEL SISTEMA

- **Descrizione dello scenario**: Nel *Par. 1.2* si è detto che, già dal punto di vista teorico, il *tuning* di un sistema SVM è molto importante per le performances di quest'ultimo. In questo test, quindi, si è voluto verificare ciò in una casistica reale come il lavoro in oggetto.
- **Obiettivo**: Dimostrare che senza effettuare il *tuning* del sistema SVM, le prestazioni sono pressoché nulle.
- **Risultati numerici del test**: Per arrivare a dimostrare ciò, si è deciso di comparare le prestazioni del modello SVM settato con **parametri di *tuning* appositamente errati e fuori scala**, con un modello settato **con i giusti parametri**, ottenuti dalla procedura di *tuning* automatica della libreria e1071.



Come è possibile notare, con i parametri palesemente errati, l'accuratezza del sistema tende subito a stabilizzarsi su un valore praticamente nullo (≈ 0.04%).
D'altro canto, invece, con i parametri di *tuning* corretti, le prestazioni sono in linea con i risultati che ci si aspetterebbe.
N.B. I risultati numeri sono identici al Test – 1 (*vedi Par. 3.2*).

- **Osservazioni**: Si è potuto appurare, quindi, che **l'operazione di *tuning* di un sistema SVM è praticamente sempre necessaria**.
  D'altra parte, però, essa non è, come si è visto nel test 3 (*vedi Par. 3.4*), un'operazione molto leggera.
  Si è pensato, pertanto, di selezionare alcune bande, effettuare l'operazione, e creare una tabella con i relativi valori. **In questo modo**, nel momento in cui si deve usare un modello SVM, o si sceglie di effettuare l'operazione di *tuning*, oppure **si prendono i valori tabulati** corrispondenti al numero di bande più simile possibile a quello che si deve usare.

	Number of Features						
	6	10	40	60	80	150	264
C	4	4	4	4	4	2	1
gamma	4	4	2	1	0,25	0,25	0,25
Accuracy	0,78	0,79	0,82	0,79	0,86	0,87	0,87

Bisogna tenere conto, però, che il discorso appena fatto è relativo alla versione "*k-bands*" del sistema. Per le altre versioni, invece, il tuning è consigliabile, perché, tenendo conto del fatto che il numero di features e quindi la grandezza del dataset è fissa, l'operazione andrebbe fatta solamente *una tantum*.

3.6. TEST 5 – CONFRONTO TRA IL SOFTWARE PROPOSTO E SOFTWARE COMMERCIALE ENVI

- **Descrizione dello scenario**: Uno degli obiettivi principali di questo tema d'anno, come già detto nel capitolo introduttivo, è stato quello di sviluppare un classificatore SVM con la libreria "e1071" al fine di valutarne le performance rispetto a software commerciali come ***ENVI*** [6].
- **Obiettivo**: Date 2 configurazioni, comparare le prestazioni dei due sistemi.
  Le due configurazioni sono:
  - **Configurazione 1**:
    - Numero features utilizzate = 264 (132 MSAVI + 132 BRIGHTNESS)
    - Tuning **effettuato**
      - C = 1
      - Gamma = 0.03125
    - Numero pixel di training **<** Numero pixel di test

  - **Configurazione 2**:
    - Numero features utilizzate = 264 (132 MSAVI + 132 BRIGHTNESS)
    - Tuning **effettuato**
      - C = 1
      - Gamma = 0.03125
    - Numero pixel di training **>** Numero pixel di test

- **Risultati numerici del test**: Di seguito si mostrano le matrici di confusione delle due configurazioni e le relative "*Overall Accuracy*".

- **Configurazione 1**

		GROUND TRUTH															
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Tot.**
SOFTWARE - ENVI	**1**	317	1	0	0	0	0	20	25	0	0	129	0	0	0	0	492
	2	54	196	1	22	52	34	81	1	9	3	88	12	0	0	0	553
	3	0	0	1218	65	0	0	0	0	4	0	0	4	0	0	0	1291
	4	0	4	46	695	188	0	0	0	2	0	6	192	0	0	0	1133
	5	0	0	0	22	90	0	0	0	0	0	0	2	0	0	0	114
	6	0	0	0	0	0	316	10	0	0	0	0	0	0	0	0	326
	7	0	0	0	0	15	223	304	0	0	1	0	0	0	0	0	543
	8	8	0	0	0	9	0	0	55	0	15	2	0	0	0	0	89
	9	20	32	29	8	22	0	9	14	24	16	10	0	0	0	0	184
	10	37	0	0	0	0	0	0	18	0	68	47	0	0	0	0	170
	11	15	0	0	55	327	151	68	0	0	0	1325	8	0	0	0	1949
	12	0	0	0	0	0	0	0	0	0	0	0	716	387	19	0	1122
	13	0	0	0	0	0	0	0	0	0	0	0	118	1458	8	0	1584
	14	0	0	0	0	0	0	0	0	0	0	0	30	28	264	0	322
	15	0	0	0	0	0	0	0	0	0	0	0	0	9	0	1604	1613
	Tot.	451	233	1294	867	703	724	492	113	39	103	1607	1082	1882	291	1604	11485

		GROUND TRUTH															
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Tot.**
SOFTWARE - PROPOSTO	**1**	303	0	0	2	245	13	0	32	7	15	101	0	9	0	0	727
	2	120	231	0	154	90	53	39	6	17	0	112	17	0	0	0	839
	3	0	0	1343	22	0	0	0	0	0	0	0	0	0	0	0	1365
	4	0	1	0	436	49	6	0	0	0	0	3	24	0	0	0	519
	5	0	0	0	8	118	0	27	0	0	0	0	0	0	0	0	153
	6	0	0	0	2	0	388	75	0	0	0	0	0	0	0	0	465
	7	0	0	0	12	157	218	230	0	0	1	0	0	0	0	0	618
	8	13	0	2	0	0	0	7	56	5	0	72	0	0	0	0	155
	9	0	0	47	0	0	0	0	13	14	0	0	5	0	0	0	79
	10	16	1	0	7	0	0	0	5	0	90	0	3	0	0	0	122
	11	0	0	13	207	80	51	128	2	0	0	1520	1	0	0	0	2002
	12	0	0	15	51	74	0	0	0	0	0	0	846	70	2	0	1058
	13	0	0	0	0	0	0	0	0	0	0	0	159	1718	6	0	1883
	14	0	0	0	0	0	0	0	0	0	0	0	39	86	286	0	411
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1608	1608
	Tot.	452	233	1420	901	813	729	506	114	43	106	1808	1094	1883	294	1608	12004

	SVM sist. proposto	SVM in Envi
C	1	1
Gamma	0.03125	0.03125
OA	0.765	0.753

- **Configurazione 2**

		GROUND TRUTH															
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Tot.**
SOFTWARE - ENVI	**1**	93	4	0	0	0	0	0	9	0	3	0	0	0	0	0	109
	2	0	131	0	0	2	0	0	0	12	3	8	0	0	0	0	156
	3	0	0	720	0	0	0	0	0	2	0	0	0	0	0	0	722
	4	0	0	0	52	82	0	0	0	0	0	0	2	0	0	0	136
	5	0	5	0	52	55	0	0	0	0	1	0	0	0	0	0	113
	6	0	0	0	0	0	135	0	0	0	0	0	0	0	0	0	135
	7	0	0	0	0	0	1	162	0	0	1	0	0	0	0	0	164
	8	116	0	0	0	0	0	0	90	33	104	0	0	0	0	0	343
	9	1	1	0	0	0	0	0	0	13	4	0	0	0	0	0	19
	10	4	0	0	0	0	0	0	0	6	6	0	0	0	0	0	16
	11	0	18	0	0	0	0	0	0	0	4	473	0	0	0	0	495
	12	0	0	0	30	0	0	0	0	0	0	0	99	2	0	0	131
	13	0	0	0	0	0	0	0	0	0	0	0	64	144	1	0	209
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	194	0	194
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	718	718
	Tot.	214	159	720	134	139	136	162	99	66	126	481	165	146	195	718	3660

		GROUND TRUTH															
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Tot.**
SOFTWARE - PROPOSTO	**1**	87	0	0	0	0	0	0	2	8	0	0	0	0	0	0	97
	2	0	127	0	1	0	0	0	0	0	0	0	0	0	0	0	128
	3	0	0	714	0	0	0	0	0	4	0	0	0	0	0	0	718
	4	0	0	0	67	26	0	0	0	0	0	0	0	0	0	0	93
	5	0	0	0	16	112	0	0	0	0	0	0	0	0	0	0	128
	6	0	0	0	0	0	137	0	0	0	0	0	0	0	0	0	137
	7	0	0	0	0	0	0	162	0	0	0	0	0	0	0	0	162
	8	104	0	0	0	0	0	0	53	9	3	0	0	0	0	0	169
	9	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	13
	10	20	0	0	0	0	0	0	19	4	123	0	0	0	0	0	166
	11	3	32	0	49	0	0	0	25	27	0	471	0	0	0	0	607
	12	0	0	0	1	1	0	0	0	1	0	0	143	1	0	0	147
	13	0	0	0	0	0	0	0	0	0	0	0	21	147	2	0	170
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	193	0	193
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	718	718
	Tot.	214	159	714	134	139	137	162	99	66	126	471	164	148	195	718	3646

	SVM sist. proposto	SVM in Envi
C	4	4
Gamma	0.03125	0.03125
OA	0.896	0.843

- **Osservazioni**: **I risultati di "*Overall Accuracy*" attestano che il software proposto, nel momento in cui si effettua il tuning del sistema, ha risultati superiori ad ENVI**.

3.7. TEST 6 – METRICHE DELLA PREDIZIONE A DUE STEP ("2steps")

- **Descrizione dello scenario**: Come già scritto nel *Par. 2.5.3*, la versione del software "***2-steps***" effettua una predizione a due step dove, tra uno passaggio e l'altro, viene effettuato il ***masking*** di alcune classi.
  L'idea alla base di questo modo di procedere è quella per cui, utilizzando un dataset costituito solo da features "vegetative" si riescano a predire meglio i pixel appartenenti a classi "vegetative", mentre, utilizzando un dataset di sole features "non-vegetative" si predicano meglio pixel appartenenti a classi come "strade", "zone edificate", "aree estrattive", etc.
- **Obiettivo**: In questa sede si vuole indagare dal punto di vista **quantitativo** se tale idea ha un riscontro con i dati.
- **Risultati numerici del test**: Il primo step della versione del software "*2-steps*" è effettuare una predizione su un dataset costruito con le features "non-vegetative" BRIGHTNESS, NDBBBI, NDBSI. Le metriche ottenute dalla matrice di confusione derivante da questa prima fase sono:

	ZONE EDIFICATE - STEP 1		
	precision	recall	f1
1-Uliveti	0,9615385	0,4672897	0,6289308
2-Frutteti	0,9716312	0,8616352	0,9133333
3-Vigneti	1	1	1
4-Grano	0,9350649	0,5373134	0,6824645
5-Foraggio	1	0,6906475	0,8170213
6-Leguminose	1	0,8540146	0,9212598
7-Ortivi	0,9876543	0,9876543	0,9876543
8-Evergreen	0,3707865	0,6666667	0,4765343
9-Decidue	0,4864865	0,2727273	0,3495146
10-Pini	0,7202797	0,8174603	0,7657993
11-Prato	0,9271654	1	0,9622063
12-Strade	0,748503	0,7621951	0,755287
13-Edifici	0,7016575	0,8581081	0,7720365
14-Estrattive	1	0,8820513	0,9373297
15-Acqua	0,8457008	1	0,9164008
Overall Accuracy = 0,88			

Come è possibile osservare, effettivamente le classi "non-vegetative" 12,13,14 sono predette con un alto valore di f1-score. **Però**, anche altre classi come 2 e 9 che appartengono a zone "vegetative" hanno un alto valore.
Per quanto riguarda il secondo step, dove si effettua la predizione utilizzando un dataset con features MSAVI, PHENO NDVI, NDRE e VARI, le metriche associate sono:

	VEGETAZIONE – STEP 2		
	precision	recall	f1
1-Uliveti	0,9121622	0,6308411	0,7458564
2-Frutteti	*step 1*	*step 1*	*step 1*
3-Vigneti	1	1	1
4-Grano	0,9666667	0,6492537	0,7767857
5-Foraggio	0,8671329	0,8920863	0,8794326
6-Leguminose	1	0,9635036	0,9814126
7-Ortivi	1	0,9506173	0,9746835
8-Evergreen	0,4965035	0,7171717	0,5867769
9-Decidue	*step 1*	*step 1*	*step 1*
10-Pini	0,8870968	0,8730159	0,88
11-Prato	*step 1*	*step 1*	*step 1*
12-Strade	*step 1*	*step 1*	*step 1*
13-Edifici	*step 1*	*step 1*	*step 1*
14-Estrattive	*step 1*	*step 1*	*step 1*
15-Acqua	0,9031447	1	0,9491077
Overall Accuracy = 0,92			

Anche in questo caso è possibile vedere che i valori relativi alle bande "vegetative" sono abbastanza alti.

- **Osservazioni**: In definitiva è possibile affermare che **l'idea di fondo** della predizione a due step **è confermata** anche dal punto di vista numerico. Va aggiunto, però, che dai dati emerge anche che classi "vegetative" vengano predette molto bene anche partendo da un dataset di features "non-vegetative".

3.8. *TEST 7 – METRICHE DELLA PREDIZIONE "all"*

- **Descrizione dello scenario**: Avendo a disposizione tutte le immagini di input, e quindi tutti gli indici già citati nei capitoli precedenti, è stato naturale considerare di effettuare una classificazione a partire da un dataset che comprendesse **tutti gli indici**.
- **Obiettivo**: Analizzare dal punto di vista quantitativo le metriche legate ad una predizione basata su tutti gli indici.
- **Risultati numerici del test**: Le metriche associate al test sono:

	ALL		
	precision	recall	f1
1-Uliveti	0,9632353	0,6121495	0,7485714
2-Frutteti	0,9858156	0,8742138	0,9266667
3-Vigneti	0,9986014	1	0,9993002
4-Grano	0,9230769	0,7164179	0,8067227
5-Foraggio	0,9440559	0,971223	0,9574468
6-Leguminose	1	1	1
7-Ortivi	1	0,9814815	0,9906542
8-Evergreen	0,4746835	0,7575758	0,5836576
9-Decidue	0,5135135	0,2878788	0,368932
10-Pini	0,777027	0,9126984	0,8394161
11-Prato	0,9363817	1	0,9671458
12-Strade	0,7149758	0,902439	0,7978437
13-Edifici	0,7763158	0,7972973	0,7866667
14-Estrattive	1	0,9076923	0,9516129
15-Acqua	0,9849108	1	0,9923981
Overall Accuracy = 0,92			

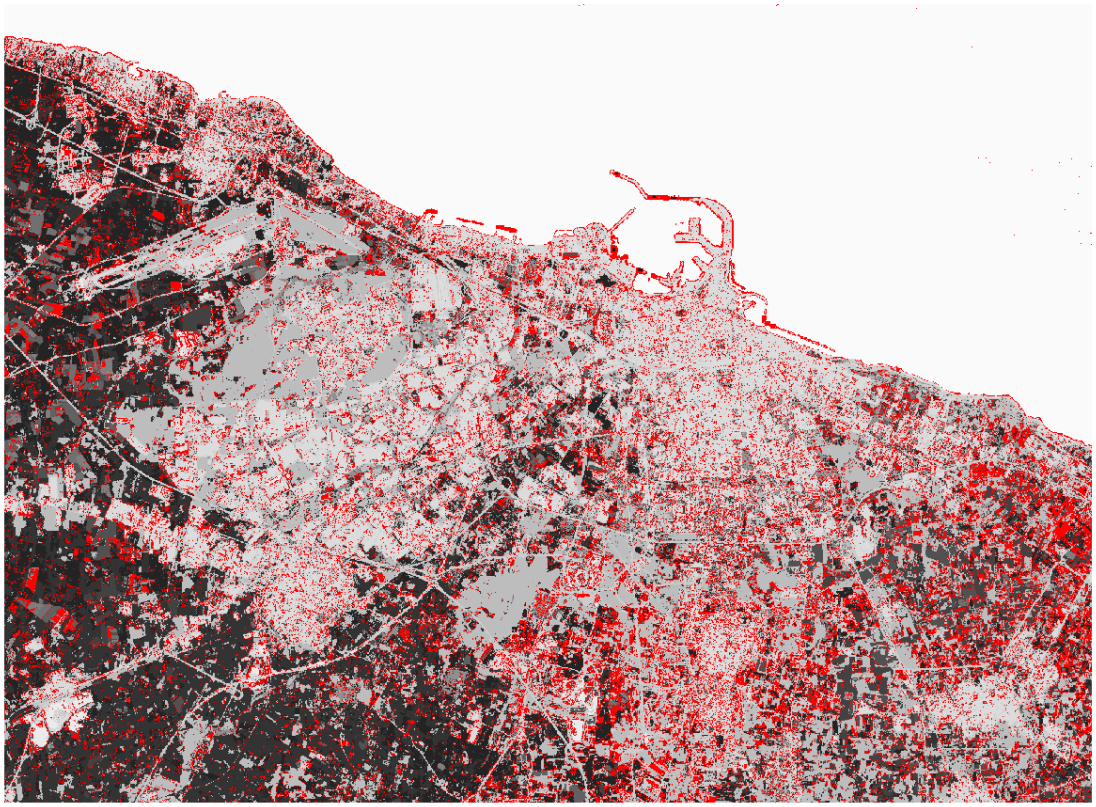
Come è possibile osservare, i valori di f1-score sono tutti abbastanza alti, come, tra l'altro, anche il valore di accuratezza complessiva del sistema.

È però utile mettere in relazione quest'ultimi dati con quelli della classificazione a "*2-steps*" (*vedi Par. 3.7*):

	ZONE EDIFICATE - STEP 1	VEGETAZIONE - STEP 2	ALL
	f1 - score	*f1 - score*	*f1 - score*
1-Uliveti	*step 2*	0,7458564	0,7485714
2-Frutteti	0,9133333	*step 1*	0,9266667
3-Vigneti	*step 2*	1	0,9993002
4-Grano	*step 2*	0,7767857	0,8067227
5-Foraggio	*step 2*	0,8794326	0,9574468
6-Leguminose	*step 2*	0,9814126	1
7-Ortivi	*step 2*	0,9746835	0,9906542
8-Evergreen	*step 2*	0,5867769	0,5836576
9-Decidue	0,3495146	*step 1*	0,368932
10-Pini	*step 2*	0,88	0,8394161
11-Prato	0,9622063	*step 1*	0,9671458
12-Strade	0,755287	*step 1*	0,7978437
13-Edifici	0,7720365	*step 1*	0,7866667
14-Estrattive	0,9373297	*step 1*	0,9516129
15-Acqua	*step 2*	0,9491077	0,9923981

- **Osservazioni**: Il risultato di questo test è **molto importante** in quanto ci dice **che la predizione effettuata utilizzando tutti gli indici, è, dal punto di vista di metriche, migliore rispetto a quella effettuata con 2-steps**. Infatti, 12 indici (*in verde*) hanno un

valore di f1-score maggiore, mentre 3 indici (*in rosso*) hanno un valore di f1-score sì minore, ma molto vicino a quello della predizione in due passate.
Inoltre, un altro aspetto computazionale a favore della versione "*all*" è che bisogna effettuare solo una passata (anche se su dataset più grande), quindi, con risparmio di tempo rispetto ad effettuarne due.
Tra l'altro, oltre che dal punto di vista meramente numerico, ci sono anche da considerare altri requisiti non funzionali del sistema, cioè, per esempio, il fatto che il sistema "*all*" è molto meno complesso ed ha un codice molto più leggibile rispetto alla versione "*2-steps*".
Si è anche voluto approfondire un aspetto già citato nel test – 1 (*vedi Par. 3.2*), cioè che, dal momento che si trattano grandi quantità di dati, anche piccole differenze sui valori di accuratezza, come sulla **seconda** cifra decimale, portano a risultati abbastanza diversi. Per testare quest'ultima ipotesi, si sono innanzitutto create due immagini finali, sia con la versione "*2-steps*" sia con "*all*". Dopo di che, con un semplice script anch'esso in R**, si sono evidenziati in rosso tutti i pixel classificati diversamente**. Dal punto di vista numerico questi sono: 312844 pixel, su un totale di 2603688 pixel, quindi il **12% dei pixel totali**.
Dal punto di vista visivo, invece, la situazione è la seguente:



3.9. TEST 8 – CONFRONTO TRA KERNEL "LINEARE" E "RADIALE"

- **Descrizione dello scenario**: Quando si utilizza il modello SVM in R con la libreria "*e1071*", di *default* viene scelto il *kernel* radiale (*radial*).
  Nella letteratura sull'argomento, però, spesso è citato l'uso del *kernel* lineare ("*linear*") nel momento in cui le features del sistema non sono poche. Questo è dovuto al fatto che uno dei compiti del kernel è quello di proiettare in un iperspazio a più dimensioni i samples del sistema, in maniera tale da trovare più facilmente un iperpiano di separazione. Se, però, come nel caso in esame, **il numero di features**, e quindi il numero di dimensioni **è già abbastanza alto**, si può pensare di non proiettare alcunché ed usare una kernel lineare.
- **Obiettivo**: Come obiettivo, quindi, ci si è posti quello di confrontare, a livello numerico, i due kernel, radiale e lineare.
- **Risultati numerici del test**: Il test è stato svolto sulla versione del software "*k-bands*" prendendo in esame le bande: 6,10,40,60,80,150.

	RADIAL KERNEL					
	Number of Features					
	6	10	40	60	80	150
Kernel Parameters	C,γ = 4,4	C,γ = 4,4	C,γ = 4,2	C,γ = 4,0.5	C,γ = 4,0.5	C,γ = 4,0.25
Time Tuning	1390 s	1416 s	2827 s	5519 s	8425 s	21563 s
Accuracy	0,78	0,79	0,7	0,73	0,72	0,53

	LINEAR KERNEL					
	Number of Features					
	6	10	40	60	80	150
Kernel Parameters	C = 4	C = 4	C = 4	C = 4	C = 4	C = 4
Time Tuning	213 s	236 s	425 s	465 s	444 s	344 s
Accuracy	0,75	0,78	0,85	0,78	0,79	0,89

- **Osservazioni**: Il risultato prettamente numerico è in maniera inconfutabile a vantaggio del kernel lineare, **in quanto ha sia valori uguali o superiori di accuratezza totale (*Overall Accuracy*), sia valori temporali nettamente inferiori (nell'ultima caso 6h vs 6minuti) e, inoltre, necessita del tuning di un solo parametro "C" anziché due "C" e "gamma"**.

4. CONCLUSIONI

Al termine di questo lavoro, analizzando nel suo insieme sia il software proposto, sia i risultati derivanti dai vari test, si sono effettuate alcune considerazioni finali:

4.1. I test svolti hanno confermato ciò che si afferma in letteratura [*9*] ovvero che **con pochi dati** di training a disposizione, nel caso in esame lo 0.46% rispetto ai dati totali, **la tecnica SVM è capace di restituire ottimi risultati**. Infatti, indipendentemente da come si effettua la predizione ("*k-bands*", "*2-steps*", "*all*"), le metriche associate hanno un valore alto.

4.2. L'altro obiettivo principale che ci si era posti, era confrontare il sistema sviluppato in questo tema d'anno con la classificazione SVM effettuata tramite **il software commerciale *ENVI***. Si è visto (*vedi Par. 3.6*) che a parità di parametri (C e gamma), **il software proposto ha prestazioni (*Overall Accuracy*) leggermente più elevate**.

4.3. È stata testata la libreria RgtSVM ed è possibile affermare che ha un comportamento piuttosto stabile, anche se non sempre. A volte si sono presentati dei problemi non risolvibili con la documentazione ufficiale e perciò è stata aperta anche una "*Issue*" sul repository ufficiale GitHub *[10]*. Vi è però da dire che tale libreria non è inserita ancora nei repository ufficiali CRAN, perciò è supportata solo da sviluppatori di terze parti.

Dal punto di vista computazionale, l'utilizzo di RgtSVM comporta un notevole risparmio di tempo ed è capace di ottenere, a parità di ogni altro fattore, risultati di accuratezza migliori. Si è intuito, ma resta da analizzare dal punto di vista quantitativo in lavori futuri, che il risparmio temporale che questa libreria comporta, è maggiore nella fase di "*prediction*", mentre la libreria "e1071" è più veloce nella fase di "*tuning*" del sistema.

4.4. Si è confermata un'altra idea che spesso è citata in letteratura, cioè che, nel momento in cui si ha un dataset con molte features, è buona norma testare il kernel di tipo lineare. Infatti, avendo già diverse dimensioni, è "inutile" utilizzare un kernel che proietta i samples in un iperspazio per trovare la miglior *decision boundary*.

4.5. **Una volta sviluppate tutte le varianti del sistema ed effettuati i relativi test a parità di dataset iniziali, si può concludere che la versione con performances numeriche più alte è la variante "*all*" con kernel "*linear*"**.

4.6. **Una considerazione molto importante**, che è sempre stata fonte di discussione in ogni fase del lavoro svolto, è l'importanza di avere molti dati "*ground truth*", cioè dati che, fissato un pixel, attestino a quale categoria appartiene.

Questa considerazione è importante anche quando si vanno ad analizzare le immagini di output da un punto di vista visivo.

Ci si può trovare davanti a due immagini, una "pulita" ed una "disturbata". La prima presenta, ad esempio, una porzione di prato perfettamente uniforme, dove tutti i pixel appartengono alla categoria "*prato*". La seconda, invece, all'interno di questa area uniforme presenta un "disturbo" di 1 pixel appartenente alla classe "*frutteto*".

Se non si hanno dati "*ground truth*" è impossibile capire se tale pixel è un reale disturbo oppure meno. Infatti, considerando anche la risoluzione spaziale delle immagini Sentinel-2 di 10x10m, quel singolo pixel potrebbe considerarsi come un piccolo frutteto di $100m^2$ all'interno di un grande parco, quindi di un'ipotesi verosimile.

BIBLIOGRAFIA

[1] http://www.alspergis.altervista.org/data/sentinel2.html

[2] https://cran.r-project.org/web/packages/raster/raster.pdf

[3] https://cran.r-project.org/package=rgdal/rgdal.pdf

[4] https://cran.r-project.org/web/packages/e1071/e1071.pdf

[5] https://github.com/Danko-Lab/Rgtsvm/

[6] https://www.harrisgeospatial.com/Software-Technology/ENVI

[7] "*Automatic Spectral Rule-Based Preliminary Mapping of Calibrated Landsat TM and ETM+ Images*", *A.Baraldi*, *V. Puzzolo*, *P. Blonda*, *L. Bruzzone*, *C. Tarantino*

[8] https://cran.r-project.org/web/packages/pheno/index.html

[9] "*Support vector machines in remote sensing: A review*", *G. Mountrakis*, *J. Im*, *C. Ogole*

[10] https://github.com/Danko-Lab/Rgtsvm/issues/13