UNIVERSITÀ
DEGLI STUDI
DI GENOVA

Surface cracks

Andrea Di Via & Francesco Minervini

MPA – Pattern Recognition

Introduzione

Le crepe sono indicatori critici di danno strutturale e sono perciò una delle principali preoccupazioni per garantire la sicurezza, la durata e la funzionalità delle strutture. Ispezionare visivamente gli elementi dell'edificio è fondamentale. Per questo motivo è bene introdurre un sistema di ***surfaces crack detection***, che a differenza dell'ispezione manuale è più efficiente sia in termini di tempo che in termini di accuratezza perché non coinvolge i giudizi soggettivi degli ispettori.

Il nostro dataset

Comprende 40.000 foto di superfici, rispettivamente 20.000 intere e 20.000 crepate. Per praticità ne andiamo a considerare solamente 5000 per ciascun tipo.

```
imageDir = 'ImageProgetto';
ids = imageDatastore(imageDir,'IncludeSubfolders',true,'LabelSource','foldernames');
nImmagini = numel(ids.Files);
countEachLabel(ids)

% estrae una immagine su 4 per velocizzare
ids = subset(ids,1:4:nImmagini);
nImmagini = numel(ids.Files);
```

```
2×2 table

    Label       Count
    ________    _____

    Negative    5000
    Positive    5000
```

Obiettivi

01. **Analisi delle features**

Binarizzazione, Segmentazione, Zonatura ed Estrazione delle features con metodi lineari (pca) e non (t-SNE).

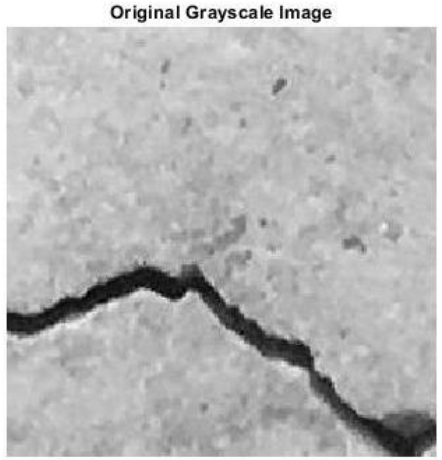
02. **Classificazione**

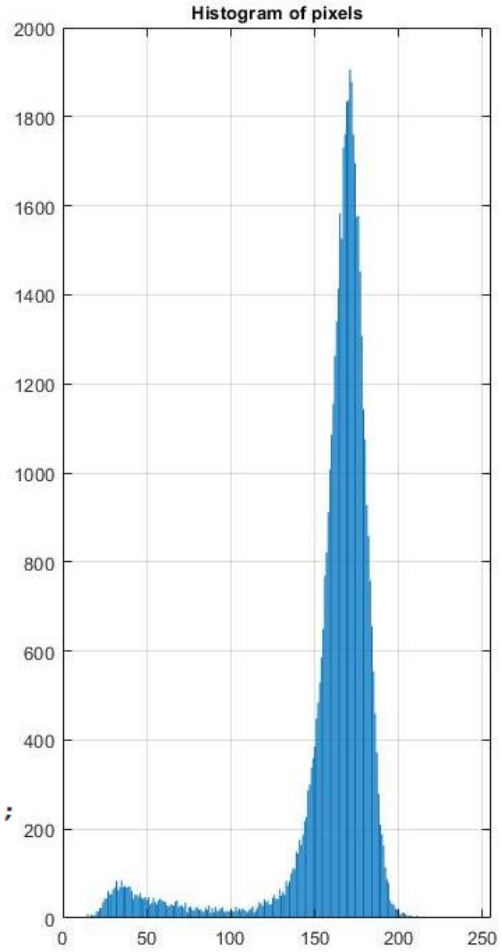
Mediante classificatori SVM e Random forest.

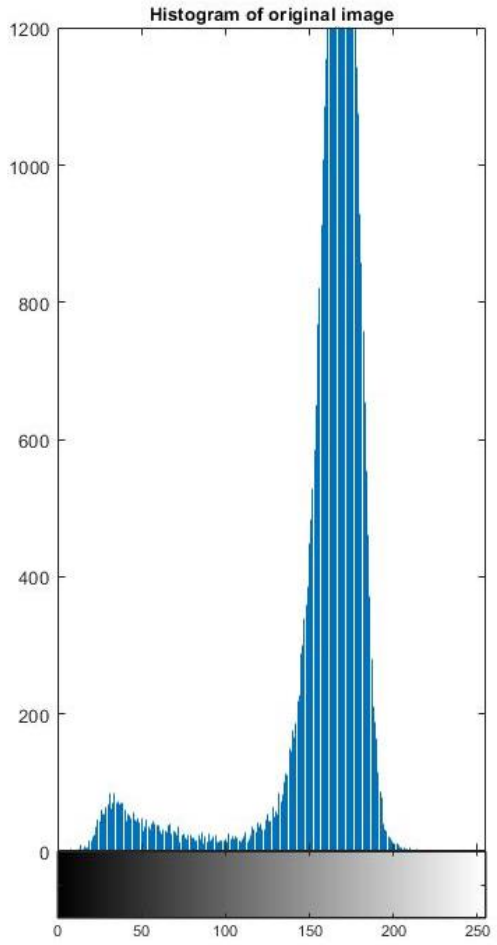
03. **Detection**

Mediante Image Labeling e trainCascade.

SEGMENTATION








```
img = readimage(ids,8762);

[row, col, channels] = size(img);
if (channels == 3)
    img = rgb2gray(img);
end
```

A causa della caratteristiche delle crepe, che sono costituite da linee e curve distinguibili, il valore in scala di grigi di una crepa è in questo caso un massimo lo cale all' interno di un'immagine. In modo da separare i pixel dell'immagine in cre pe e non crepe, è utile escogitare un approccio per discriminare meglio i due gru ppi di pixel di interesse. Vediamo la binarizzazione.

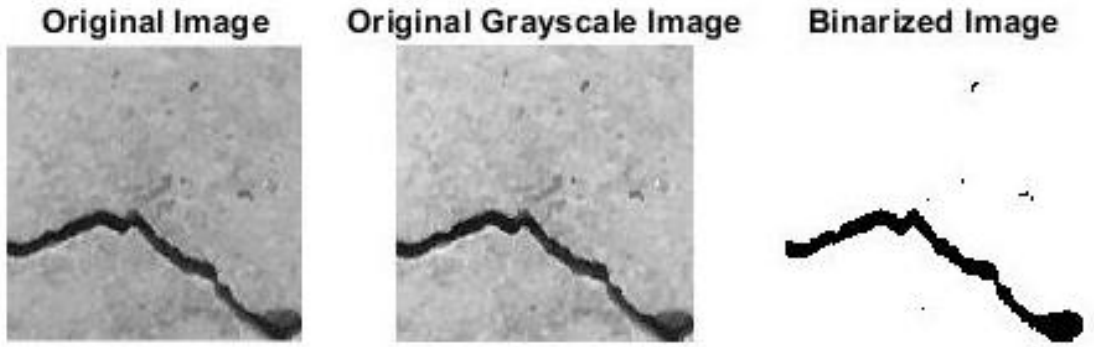
Scale di grigi

Il metodo più vecchio per segmentare una immagine è l'analisi dell'istogramma. L'idea è di considerare come unico attributo il livello di grigio, e andare a contare il numero di pixel che assume ogni singolo livello di grigio presente nella immagine. In questo modo si riesce a distinguere quali componenti sono di sfondo e quali in primo piano.

Le superfici spesso sono caratterizzate da basso contrasto e illuminazione irregolare: ciò può arrecare un forte disturbo. Per correggere questo fenomeno, è necessario eseguire un trattamento per migliorare le prestazioni di rilevamento di crack.

```
%ISTOGRAMMA SCALE DI GRIGIO
figure(1)
subplot(1, 3, 1);
imshow(img, []);
fontSize = 10;
title('Original Grayscale Image', 'FontSize', fontSize);
[pixelCount , grayLevels] = imhist(img);

subplot(1, 3, 2);
bar(pixelCount);
title('Histogram of pixels', 'FontSize', fontSize);

subplot(1,3,3);
imhist(img);
title('Histogram of original image', 'FontSize', fontSize);
```

SEGMENTATION



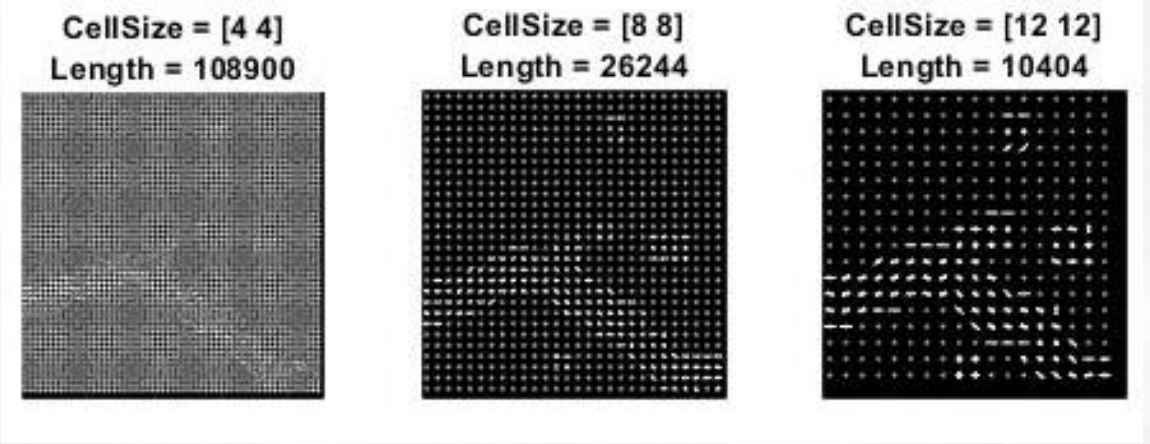

```
%BINARIZZAZIONE
imgBW=imbinarize(img);

figure(2);
subplot(1, 3, 1);
imshow(img);
title('Original Image', 'FontSize', fontSize);
subplot(1, 3, 2);
imshow(img, []);
title('Original Grayscale Image', 'FontSize', fontSize);
subplot(1,3,3);
imshow(imgBW);
title('Binarized Image', 'FontSize', fontSize);
```

```
% dimensione dell'operatore di HoG
CellSize1 = [ 4 4 ];
CellSize2= [ 8 8 ];
CellSize3 = [ 12 12 ];
[hog1, visualHoG1] = extractHOGFeatures(imgBW,'CellSize',CellSize1);
[hog2, visualHoG2] = extractHOGFeatures(imgBW,'CellSize',CellSize2);
[hog3, visualHoG3] = extractHOGFeatures(imgBW,'CellSize',CellSize3);
```

Binarization

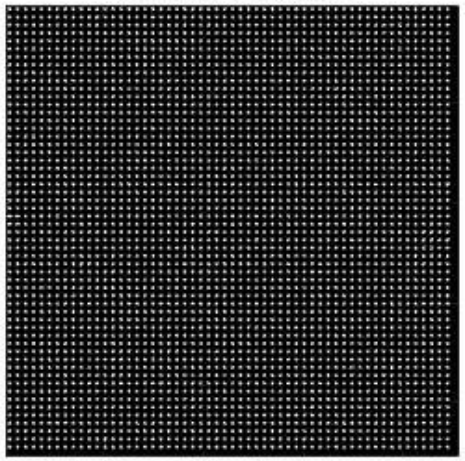
Binarizzazione delle immagini, ampiamente utilizzata per il testo, il riconoscimento e l'elaborazione delle immagini mediche, si presta molto facilmente ad essere utilizzata per il rilevamento di crepe. Questo perché i testi e le crepe hanno proprietà simili e presentano linee e curve distinguibili.
Dopo aver dunque osservato che una caratteristica significativa che discrimina le due classi è proprio quella della forma e dei contorni, ci apprestiamo ad effettuare uno zoning con uso degli istogrammi dei gradienti orientati HOG.
Su immagini generiche è bene osservare i contorni, in particolare come i loro gra dienti risultano orientati. Sulle immagini positive ci sarà una direzione del gradiente predominante, per le immagini negative invece ce ne saranno di tutti i tipi.

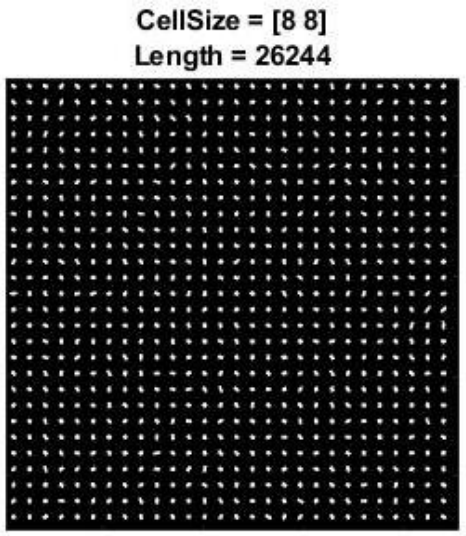
Estraiamo perciò le features di Hog.

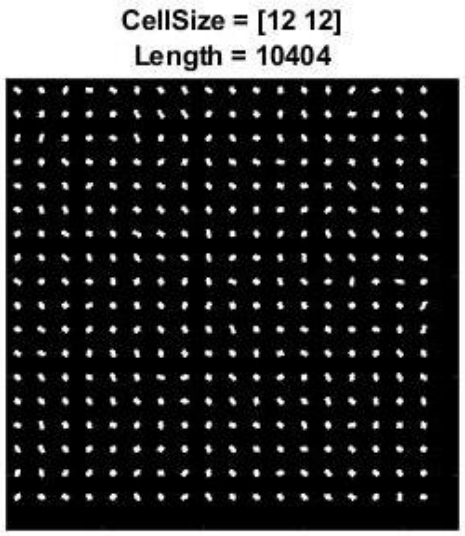


Zoning

Estrazione delle Hog Features





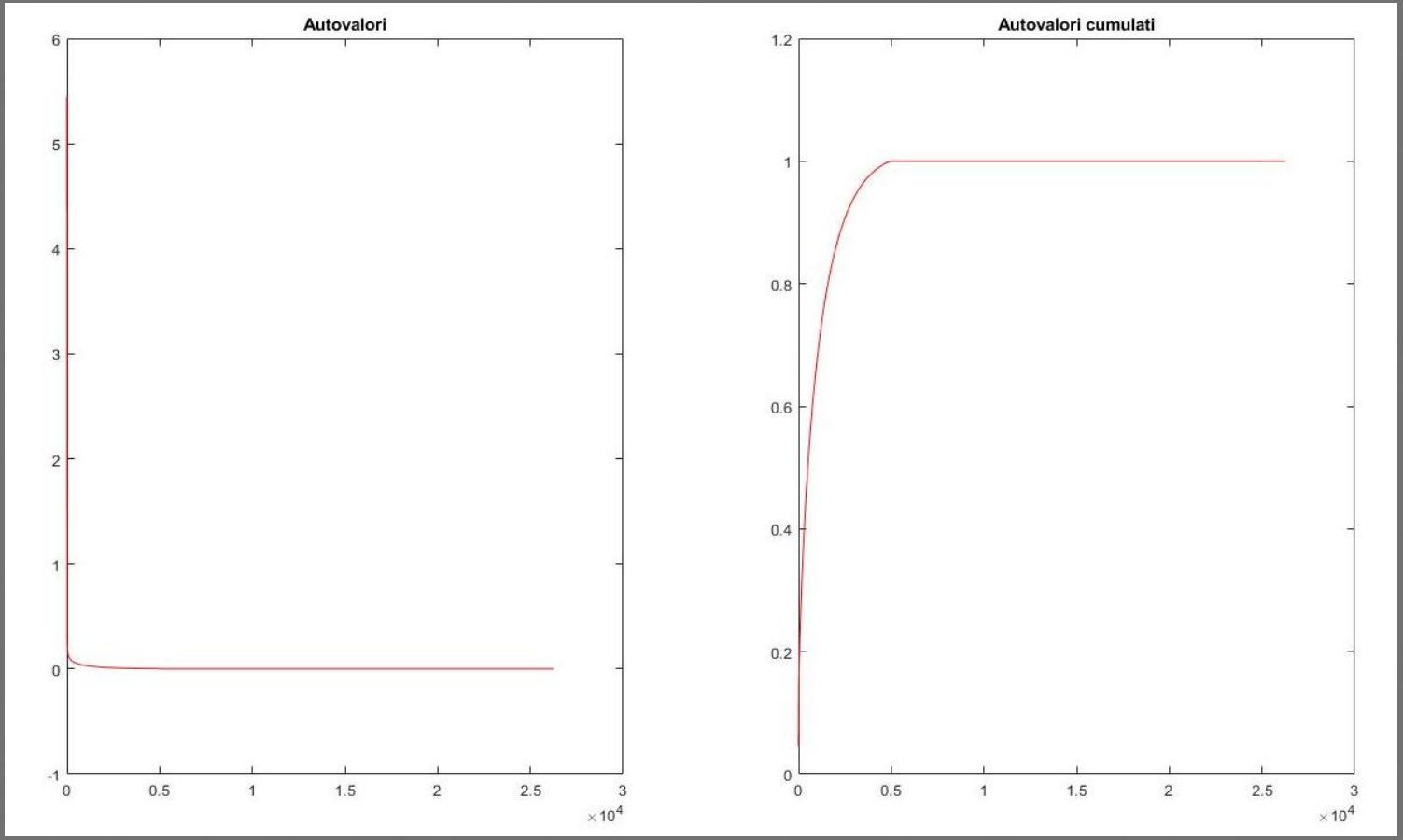



```
% visualizza immagine e features
figure(3)
subplot(2,3,1:3);
imshow(imgBW);
subplot(2,3,4);
plot(visualHoG1);
title({'CellSize = [4 4]'; ['Length = ' num2str(length(hog1))]});
subplot(2,3,5);
plot(visualHoG2);
title({'CellSize = [8 8]'; ['Length = ' num2str(length(hog2))]});
subplot(2,3,6);
plot(visualHoG3);
title({'CellSize = [12 12]'; ['Length = ' num2str(length(hog3))]});
```

Ci vogliamo però occupare del caso generale, per osservare le differenze di classificazione che si riscontrano senza effettuare una binarizzazione.

Si osserva che una cella di dimensione [12, 12] non codifica abbastanza informazioni sulla forma. Al contrario una cella di dimensione [4, 4] codifica un numero elevato di informazioni ma comporta anche un aumento significativo del vettore della features HOG. Un buon compromesso è la cella di dimensione [8, 8].

Estrazione delle features



In particolare, già intorno al 2456 valore cominciano a tendere a valori nulli, poco significativi.

Tuttavia, per praticità, proiettiamo lo spazio vettoriale di dimensione 26244 in uno spazio vettoriale di dimensione 1000 anzichè 2456.

Analisi PCA

Prima di utilizzare i diversi metodi di classificazione ci assicuriamo che non sia troppo difficile riuscire a distinguere le classi. Operiamo prima attraverso un metodo lineare: l'analisi in componenti principali.

In questo modo possiamo proiettare lo spazio di Train su uno spazio di dimensione minore senza una grande perdita dell'informazione, evitando problemi computazionali e il fenomeno di Hughes.

Dopo aver calcolato la matrice di covarianza dell'insieme di train di dimensione 26244, osserviamo dal grafico cumulate che gli autovalori significativi non sono più di 5000.

```
%PCA
ncomp = 1000;
coeff = pca(XTrain,'NumComponents',ncomp);
XTrain1=XTrain*coeff;
XTest1=XTest*coeff;
```

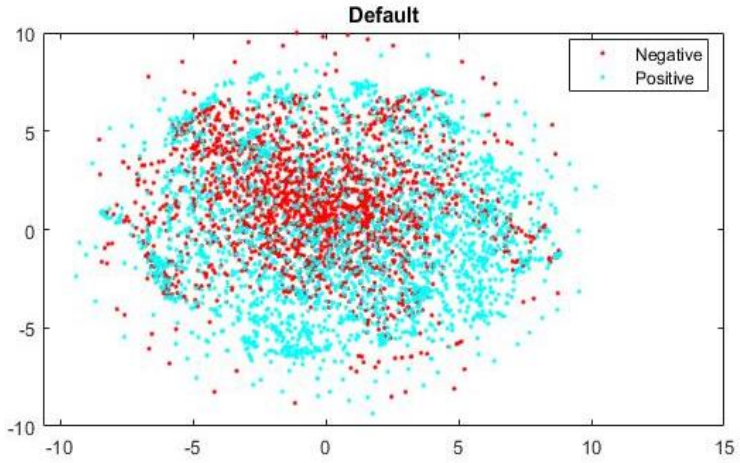
Estrazione delle features

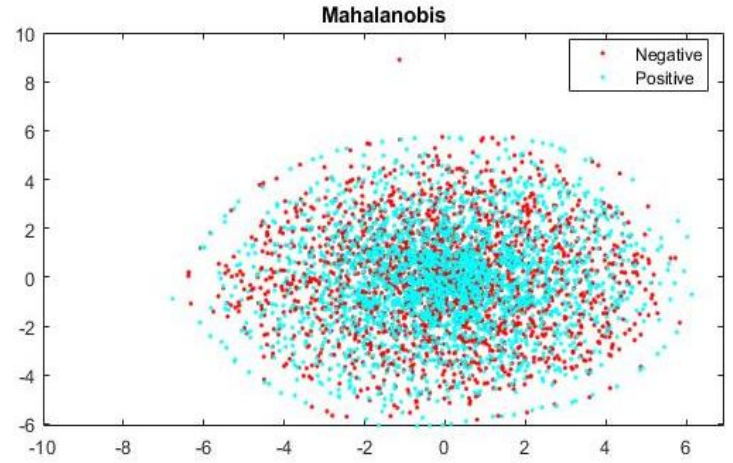
Metodo t-SNE

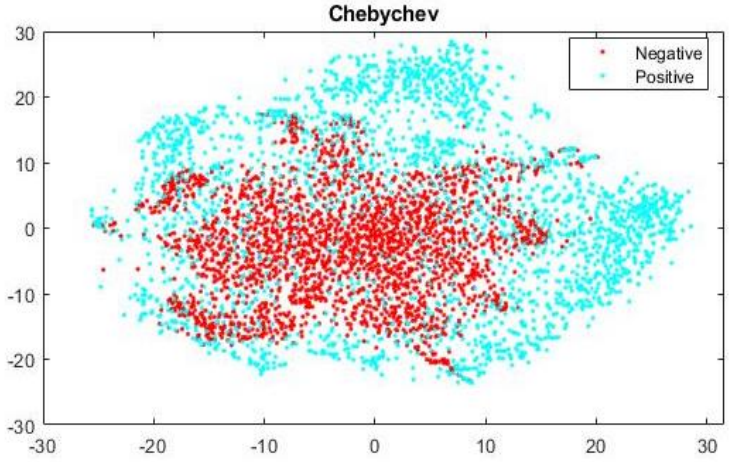
Dopo aver utilizzato la pca, utilizziamo il metodo probabilistico non supervisionato t-SNE, così che si preservi anche la topologia. Infatti riesce a trovare ottime separazioni minimizzando la diver genza tra la distribuzione originale e quella nel sottospazio di arrivo.
Osserviamo dai diversi grafici, ciascuno ottenuto a partire dalle diverse distanze considerate, che la distanza di Chebychev risulta la più efficace. In tal modo, le classi risultano separabili seppur non attraverso un separatore lineare.

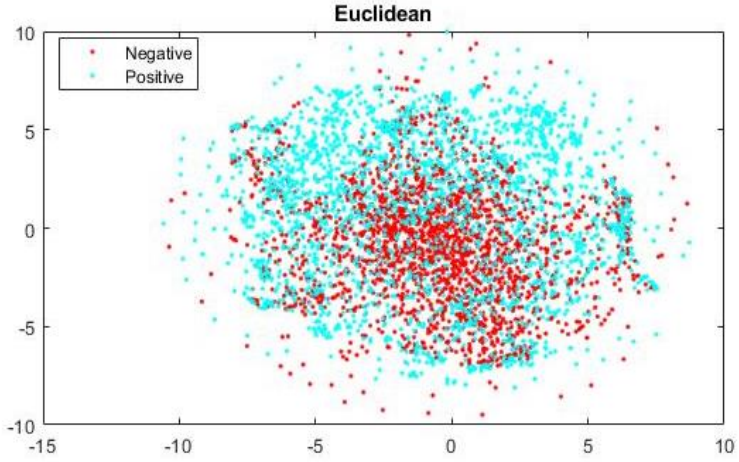
```
%METODO t-SNE
fprintf('Valutazione t-SNE ... \n')

T = tsne(XTrain,'NumPCAComponents',1000);
T1 = tsne(XTrain1,'Distance','mahalanobis');
T2 = tsne(XTrain1,'Distance','chebychev');
T3 = tsne(XTrain1,'Distance','euclidean');
```

```
%SCATTER PLOT t-SNE
figure(6)
subplot(2,2,2);
gscatter(T1(:,1),T1(:,2),YTrain)
title('Mahalanobis');
subplot(2,2,3);
gscatter(T2(:,1),T2(:,2),YTrain)
title('Chebychev');
subplot(2,2,4);
gscatter(T3(:,1),T3(:,2),YTrain)
title('Euclidean');
subplot(2,2,1);
gscatter(T(:,1),T(:,2),YTrain)
title('Default');
```



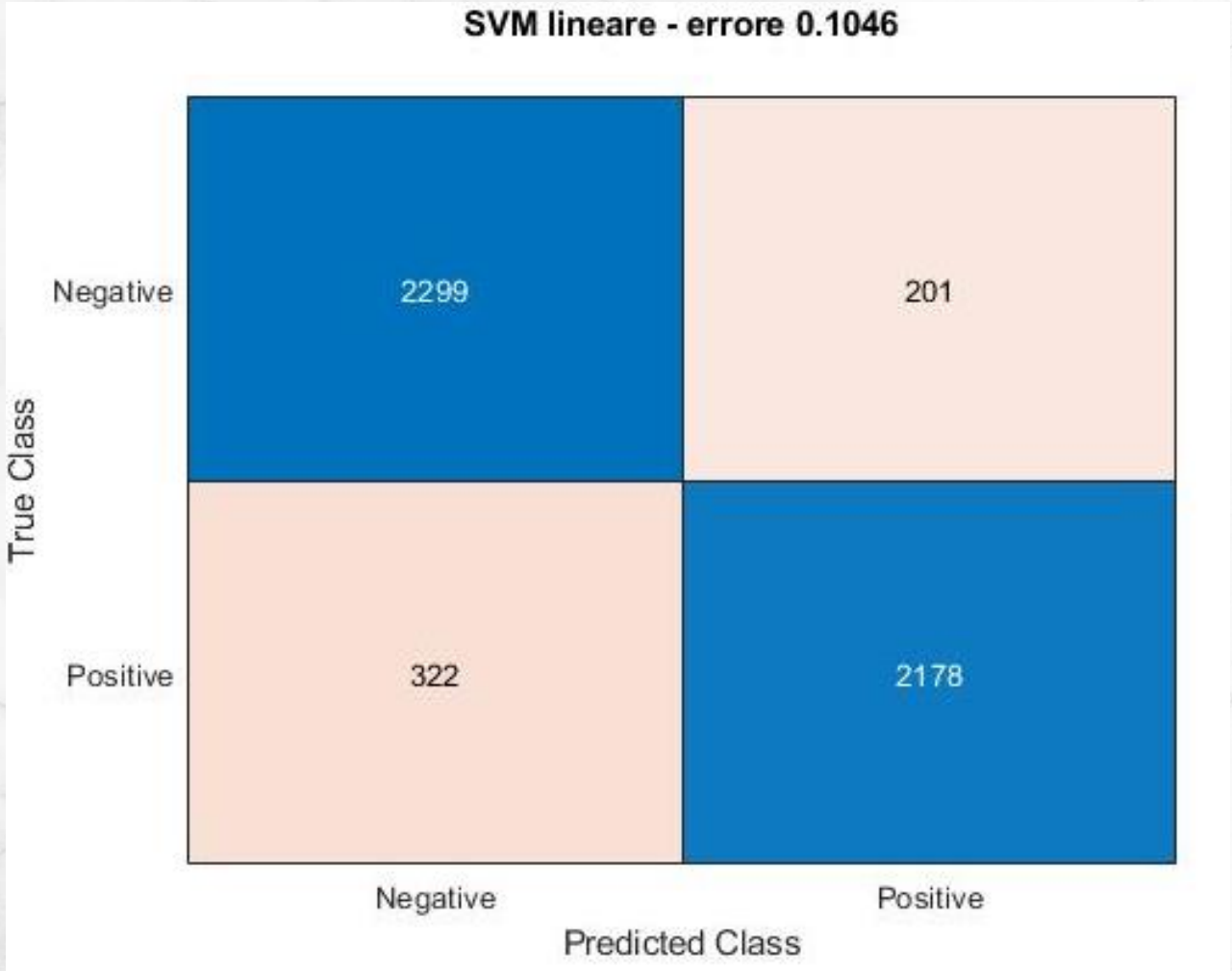






Classificazione

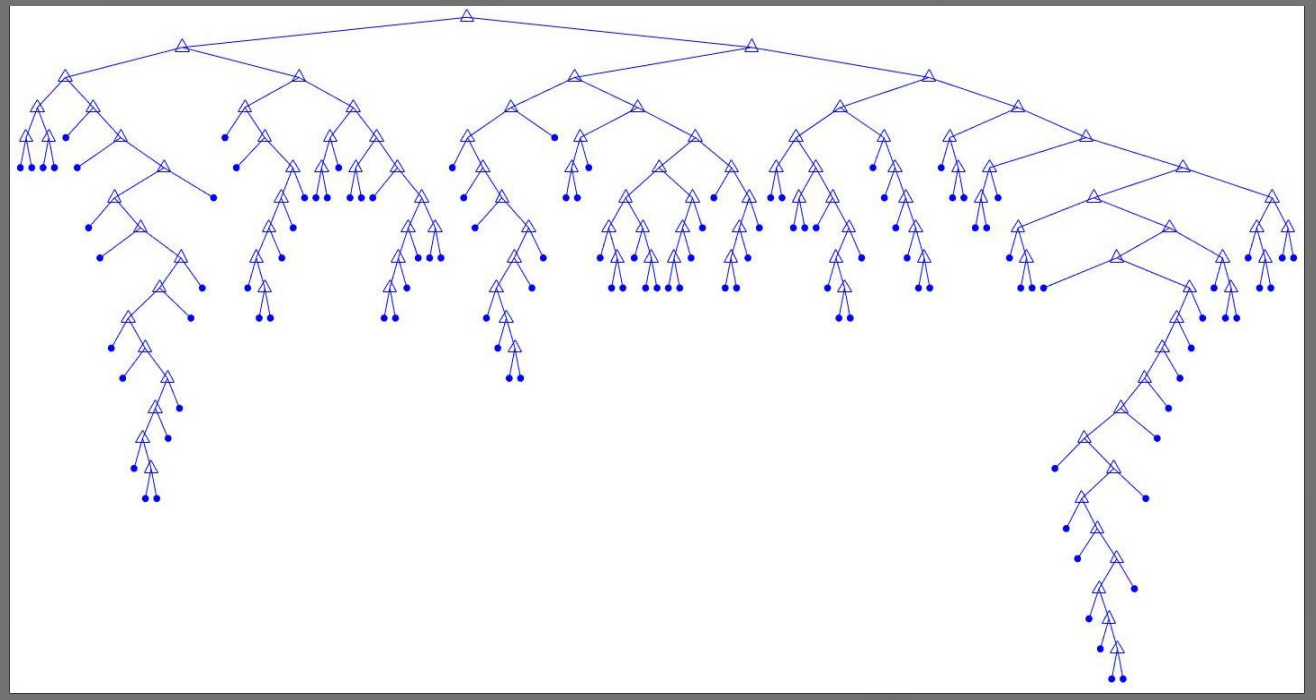
Macchina a supporto vettoriale SVM



Iniziamo la classificazione testando il classificatore lineare SVM.
SVM è un ottimo schema di classificazione binaria supervisionata.
L'algoritmo di base trova l'iperpiano (che non dipende dalla posizione iniziale e dunque dal training set) che separa due classi con il massimo margine.
Si osserva tuttavia che l'errore è piuttosto alto, pari al 10%.
Cerchiamo dunque un altro metodo che presenti un errore inferiore, e con sé il tasso di falsi negativi in particolare.

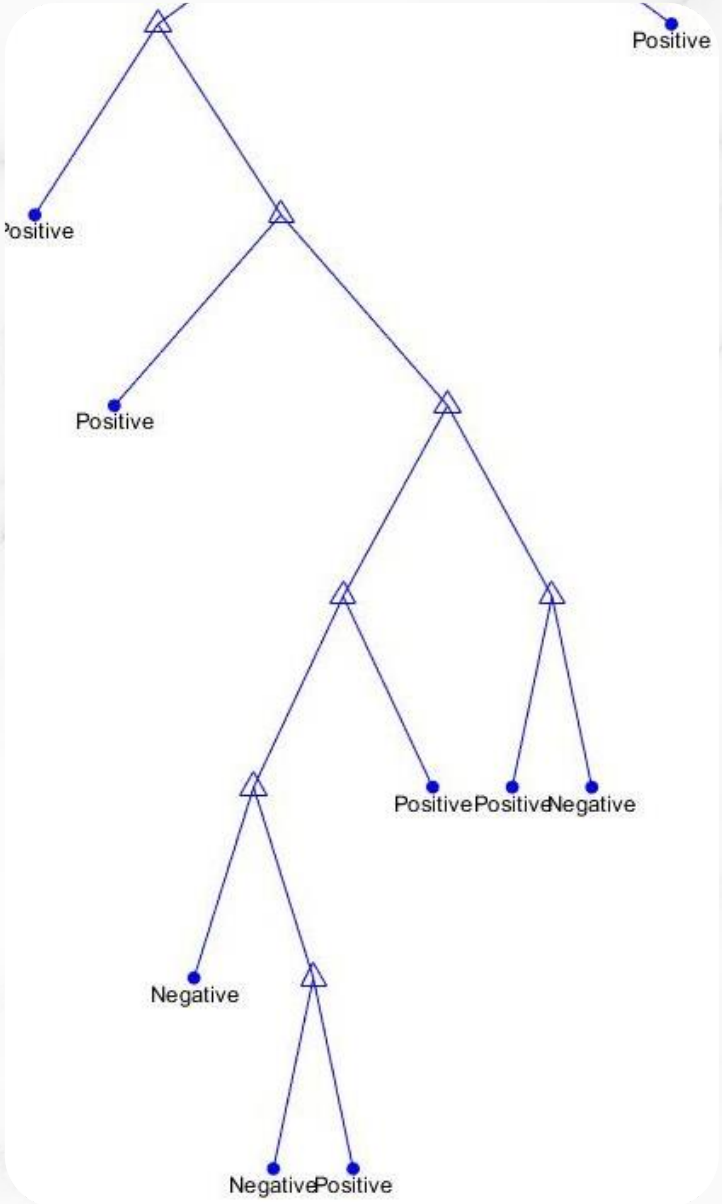
```
%classificatore lineare (SVM per default)
fprintf('Classificazione SVM ... \n')
classifier = fitcsvm(XTrain, YTrain);
% classifica le features di test
[predictedLinear, scoreLinear] = predict(classifier, XTest);
cp3 = classperf(cellstr(YTest), cellstr(predictedLinear));

%MATRICE DI CONFUSIONE
figure(7)
confusionchart(YTest, predictedLinear)
str = sprintf('SVM lineare - errore %.4f\n',cp3.ErrorRate);
title(str)
```

Random Forest



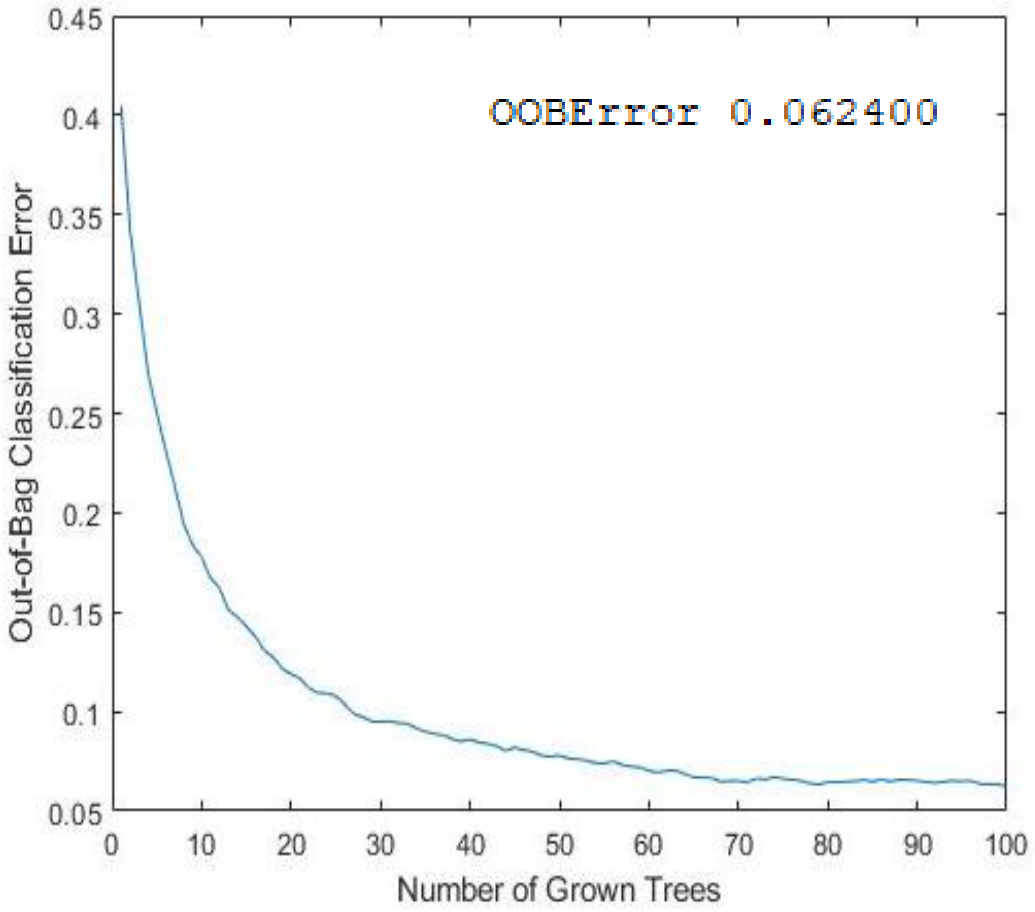
L’albero di decisione rappresenta il modo più semplice di definire una partizione del nostro insieme di dati in 2 classi. La suddivisione viene ottenuta attraverso passi successivi in cui l’insieme viene suddiviso in sottoinsiemi sempre più omo genei rispetto alla etichetta di classe «positive» o «negative».
Sulla destra è riportata un’immagine ingrandita degli ultimi rami del primo albero addestrato, e si osserva che il processo termina sempre con dei nodi foglia in cui le classi «negative» e «positive» sono univocamente determinate.

L’operazione utilizzata fa uso per default del metodo CART (Classification and Regression Tree), una procedura di ricerca dello split binario ottimale in base al massimo guadagno informativo secondo l’indice di Gini, ed un criterio di pruning basato sul compromesso tra errore di classificazione e numero di foglie.




```
%RANDOM FORESTS
fprintf('Classificazione RF ... \n')
ntrees1 = 100;
RF_Model1 = TreeBagger(ntrees1, XTrain, YTrain, 'OOBPrediction','on');
view(RF_Model1.Trees{1},'Mode','graph')
```

Bagging

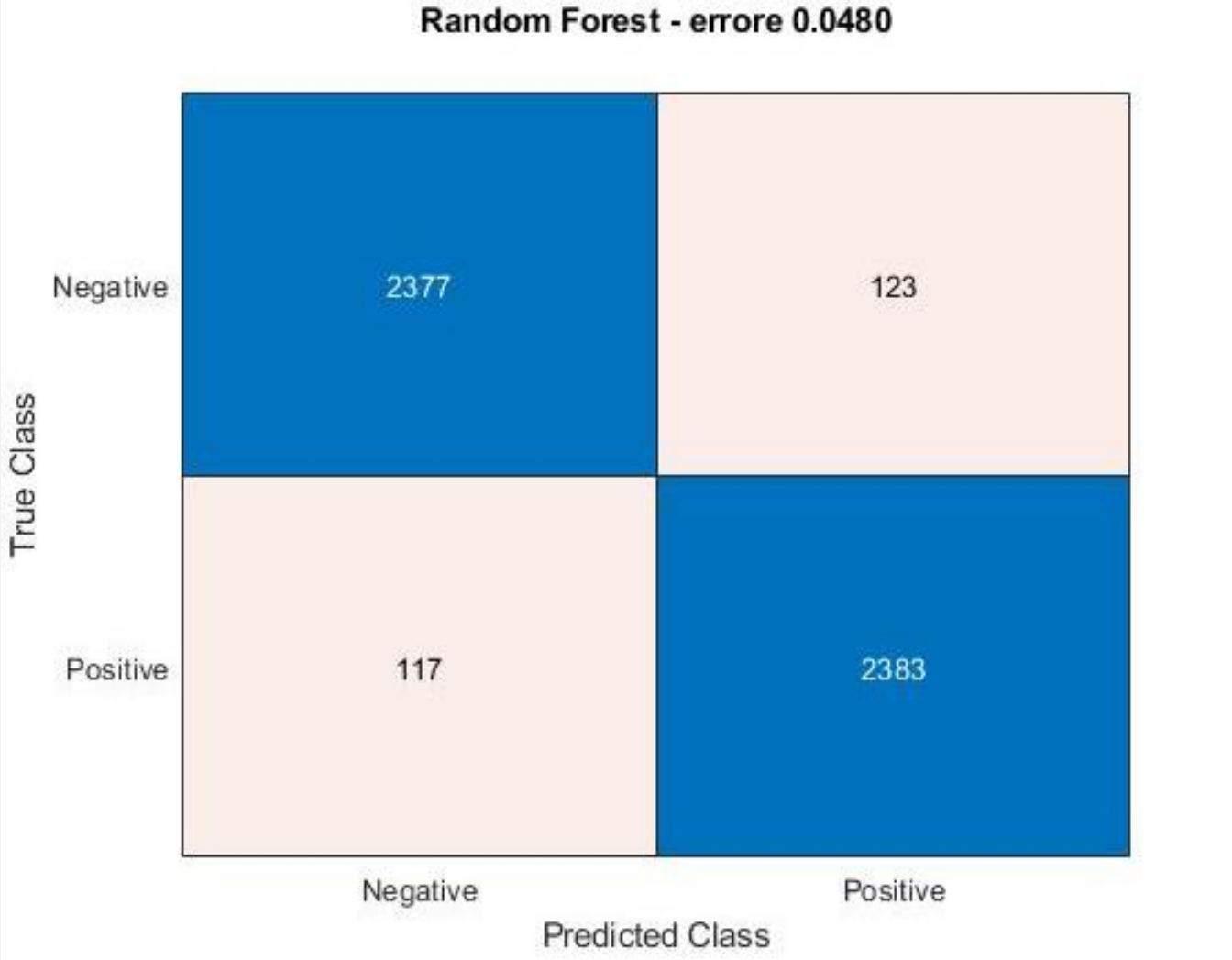


Il training set viene ricampionato estraendo ogni volta un sottoinsieme diverso dei dati originali con dei campioni a caso (anche ripetuti). Ogni sottoinsieme produce un albero di classificazione e la classe finale da assegnare ai dati ignoti è quella maggioritaria tra quelle prodotte dai singoli alberi.
Il bagging permette di stimare le prestazioni senza usare un test set, grazie all'uso dei vettori out-of-bag (i dati non estratti da ogni albero, che sono in media i 2/3).

Il grafico mostra come varia l'errore dei dati "out of bag" all'aumentare degli alberi prodotti.
In particolare si osserva come, superato il centinaio di alberi, l'errore sia molto vicino a 0.

```
%OOBERROR
figure(10)
Errore = oobError(RF_Model1);
plot(Errore);
xlabel('Number of Grown Trees')
ylabel('Out-of-Bag Classification Error')
fprintf('OOBError %f\n',Errore(end));
```

Matrice di confusione

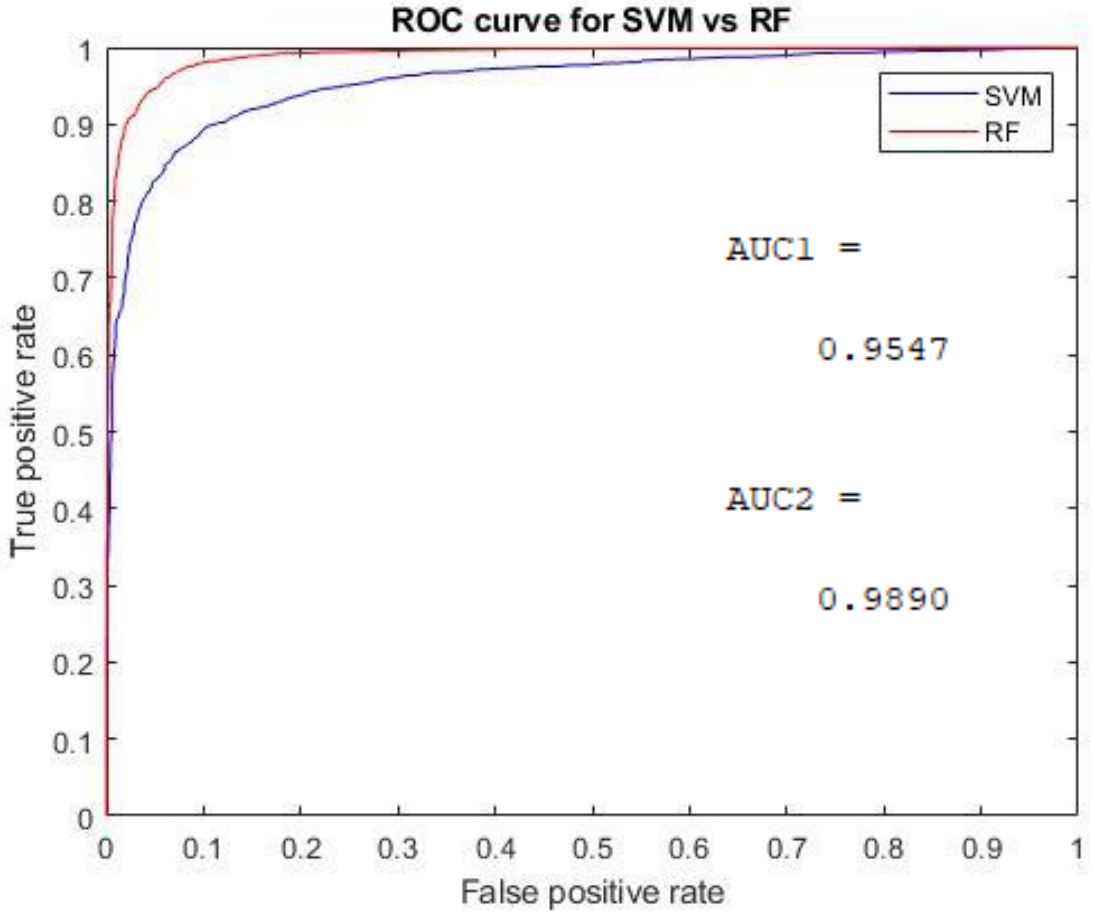


Si osserva che stavolta l'errore è accettabile, pari al 5%.
Dunque in questo caso il metodo della random forest performa decisamente meglio rispetto alle macchine a support vettoriale.
In particolare notiamo che i falsi negativi risultano minori rispetto ai falsi positive.

```
[predictedForest, scoreForest] = predict(RF_Model1, XTest);
cp2 = classperf(cellstr(YTest), cellstr(predictedForest));
fprintf('Errore test %f\n',cp2.ErrorRate);
```

```
%matrice di confusione
figure(11)
confusionchart(YTest, categorical(predictedForest))
str = sprintf('Random Forest - errore %.4f\n',cp2.ErrorRate);
title(str)
```

Curva ROC




```
%OOBERROR
figure(10)
Errore = oobError(RF_Model1);
plot(Errore);
xlabel('Number of Grown Trees')
ylabel('Out-of-Bag Classification Error')
fprintf('OOBError %f\n',Errore(end));
```

La curva ROC rappresenta tutte le possibili coppie di errori per un dato classificatore. Dall'analisi della curva ROC si è in grado di valutare meglio le precisioni dei modelli.

```
% comparazione oggettiva mediante curva ROC
fprintf('Valutazione curve ROC ... \n')
[X1,Y1,T1,AUC1] = perfcurve(YTest,scoreLinear(:,2),'Positive');
AUC1
[X2,Y2,T2,AUC2] = perfcurve(YTest,scoreForest(:,2),'Positive');
AUC2
```

In particolare nel nostro caso si osserva che il modello generato dal metodo della random forest risulta decisamente più efficace dei modelli lineari, si osserva infatti che l'area sottesa alla curva assume valori molto vicini ad 1.

```
%grafico curva
figure(12)
plot(X1,Y1,'b')
hold on
plot(X2,Y2,'r')
xlabel('False positive rate')
ylabel('True positive rate')
title('ROC curve for SVM vs RF')
legend('SVM','RF')
hold off
```

Falsi positivi e falsi negativi

Mostriamo adesso alcuni esempi dei falsi positivi e dei falsi negativi ottenuti dal metodo SVM, che è risultato piuttosto impreciso

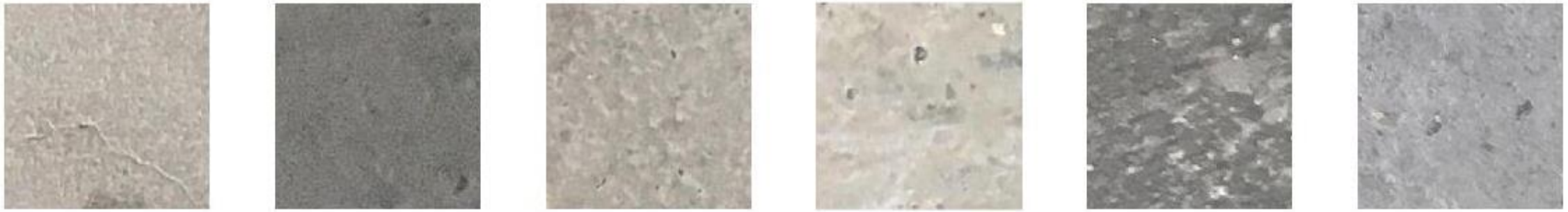
```
% mappa esempi di immagini errate (per SVM)
fprintf('Ricerca errori ... \n')
Falsi_neg = find((YTest == 'Positive')&(predictedLinear == 'Negative'));
NumFalsi_Neg = numel(Falsi_neg);
Falsi_pos = find((YTest == 'Negative')&(predictedLinear == 'Positive'));
NumFalsi_Pos = numel(Falsi_pos);
```

```
% output grafico con subplot
figure(13)
N_img = 6;
% ciclo sui falsi negativi
for k=1:min(NumFalsi_Neg,N_img)
    p = 2*Falsi_neg(k);
    img = readimage(ids, p);
    subplot(2,N_img,k);
    imshow(img);

end
title('Falsi Negativi', 'FontSize', fontSize);
```

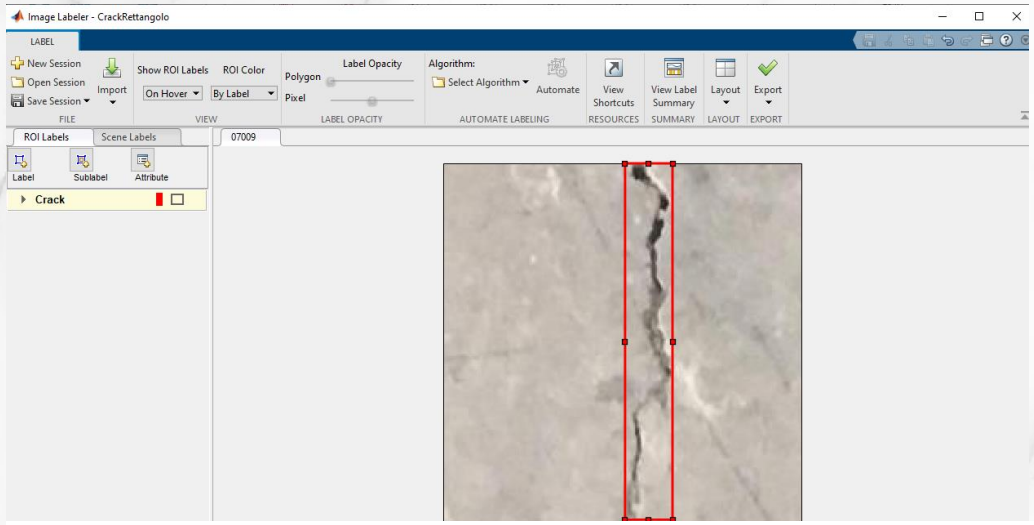
```
% ciclo sui falsi positivi
for k=1:min(NumFalsi_Pos,N_img)
    p = 2*Falsi_pos(k);
    img = readimage(ids, p);
    subplot(2,N_img,k+N_img);
    imshow(img);

end
title('Falsi Positivi', 'FontSize', fontSize);
```

Falsi Negativi

Falsi Positivi



Detection

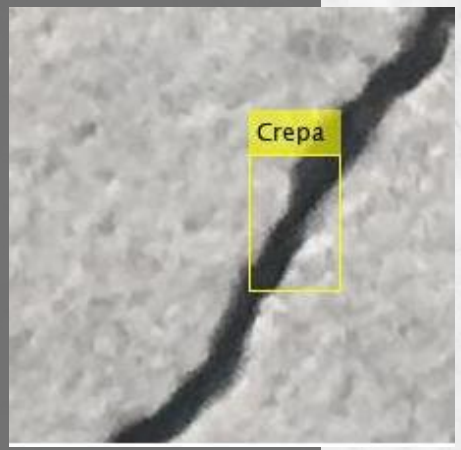


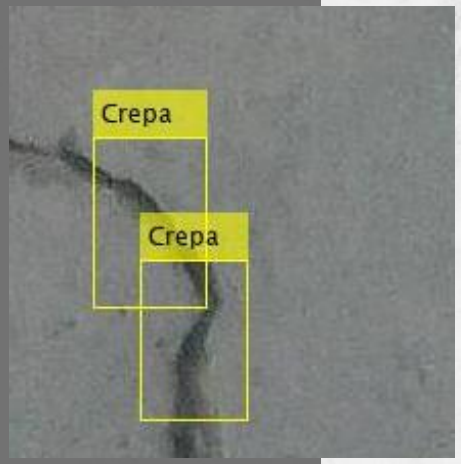
```
load('CrackRettangolo.mat');
negativeFolder = fullfile('NonCrepe');

% addestramento del detector (produzione xml di configurazione)
trainCascadeObjectDetector('my_CrackDetector.xml', gTruth ,negativeFolder,'FalseAlarmRate',0.1,'NumCascadeStages',5);
 % crea il detector sulla base del file xml
 detector = vision.CascadeObjectDetector('my_CrackDetector.xml');

 % carica le immagini di test
 idsTest = imageDatastore('CrepeTest');
 numTest = numel(idsTest.Files);
```

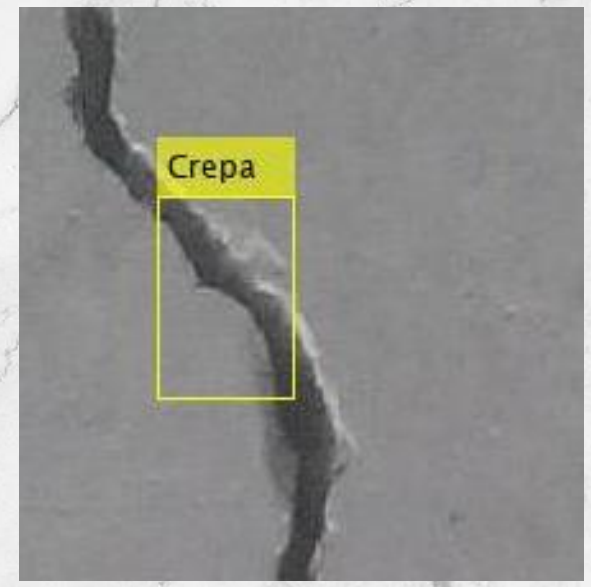
```
for i=1:numTest
    img = readimage(idsTest,i);
    % applica il detector e trova le box
    bbox = step(detector,img);
    % aggiunge le annotazioni
    detectedImg = insertObjectAnnotation(img,'rectangle',bbox,'Crepa');
    % visualizza
    figure(1);
    imshow(detectedImg);
    pause
end
```

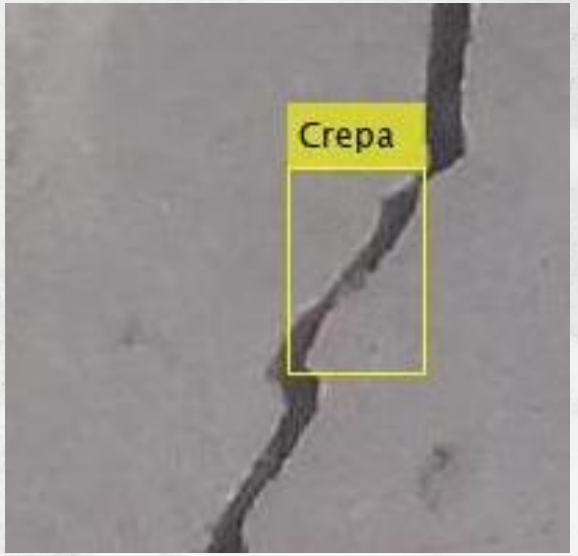
Abbiamo prima definito quali sono le ROI, regioni di interesse, da considerare attraverso l'app Image Labeler di Matlab. Dopodichè si ricorre alla funzione TrainCascade. Per velocizzare il calcolo, si genera una gerarchia di classificatori deboli con basso tasso di falsi negativi negli stadi iniziali. Alcuni classificatori molto veloci sono quindi applicati per scartare immediatamente le finestre che sono sicuramente negative.

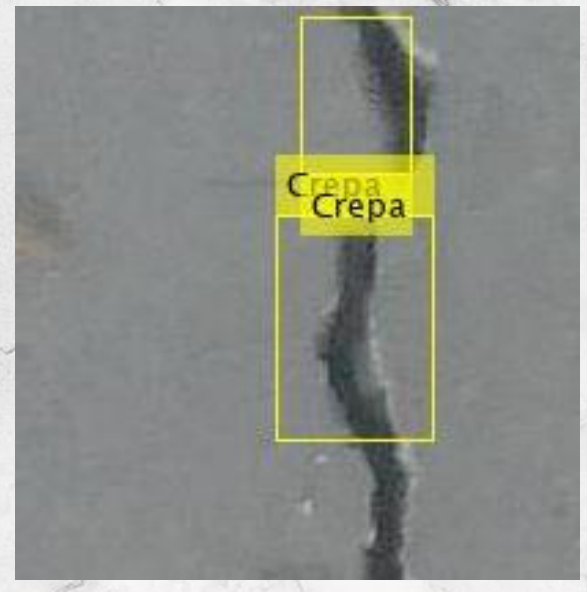
Esempi positivi



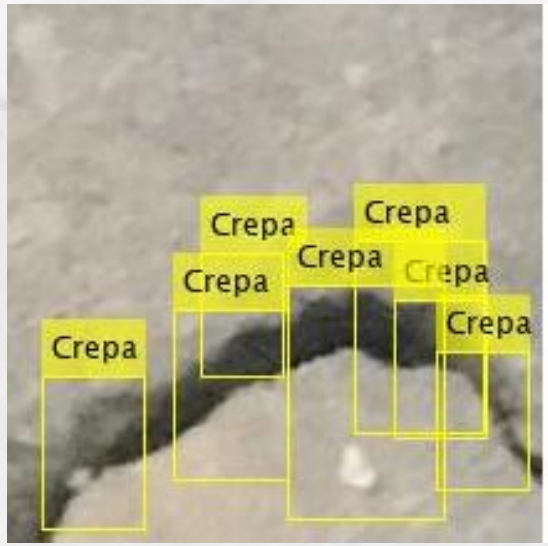


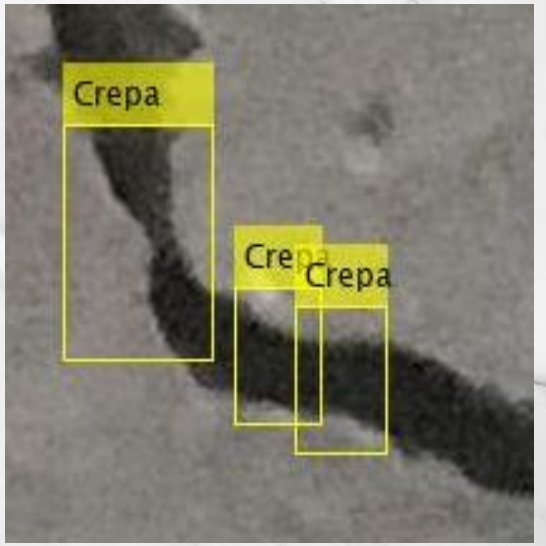


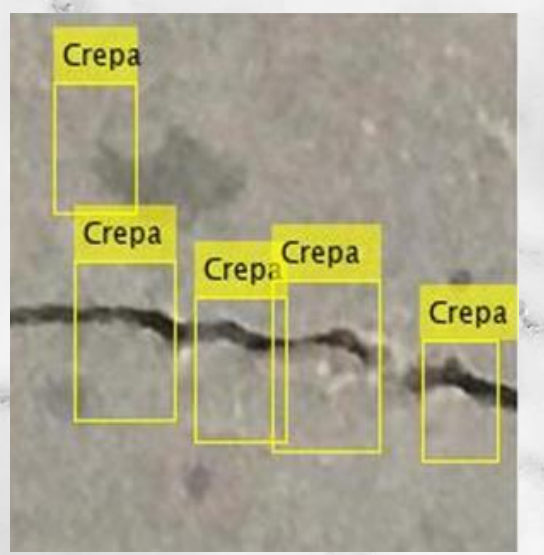


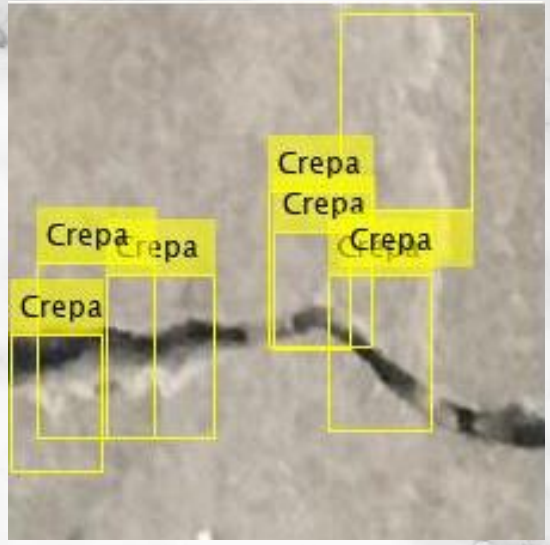


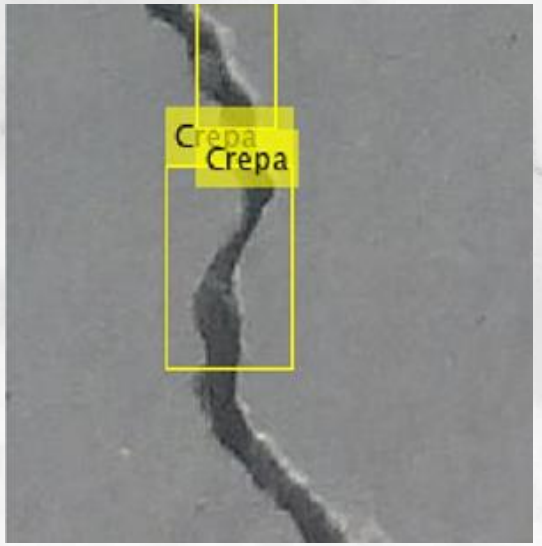
Esempi negativi











Conclusione

- Attraverso la binarizzazione si ottengono risultati più precisi e si ha una classificazione semplificata.
- La classificazione con Random Forest si è rivelata molto più efficace delle altre.
- La detection si è mostrata imprecisa soprattutto per quel che riguarda le crepe orizzontali, probabilmente a causa della scarsità di campioni simili nel training set.

Fine.