FRASAL - Rilevazione delle anomalie con tecniche di apprendimento automatico dai dati

Introduzione

Nel presente documento viene descritta l’attività svolta per definire la metodologia per l’addestramento di un modello di apprendimento automatico al fine di classificare automaticamente anomalie e/o presunti guasti.

Analisi dei dati

Il dataset a disposizione è composto dai seguenti attributi:

Gruppo Frigo/Gruppo Frigo

* Misura Termica Mandata
* Misura Termica Ritorno

Canali di Mandata e Ritorno

* Misura Termica Mandata
* Misura Termica Ritorno

Temperatura Stanze

* Temperatura Ambiente

Gruppo Pompaggio

* Delta Pressione Differenziale

Gruppo UTA

* Delta Pressione Filtri Polveri Grandi
* Delta Pressione Filtri Polveri Sottili

Misuratori Amperometrici

* Gruppo Ventilatori/Estrattori

Gli attributi sono relativi a misure di pressione/temperatura e corrente, le cui varianze disomogenee non permettono di utilizzare metodi di analisi vettoriali instance-based.

Le possibili classi di funzionamento sono individuate nelle seguenti valorizzazioni:

* Normale, **d’ora in poi N**
* Guasto al sistema di pompaggio, oppure perdita d’acqua, **d’ora in poi P**
* Causa filtri UTA intasati, non favoriscono lo scambio termico, nonostante il gruppo frigo funzioni bene la temperatura non è quella ottimale, **d’ora in poi F**
* Causa Rottura Ventole di estrazione, peggiora la fluidodinamica negli ambienti da condizionare e seppur immettendo aria fredda non si riescono a raffreddare le stanze ICT, **d’ora in poi V**
* Rottura Gruppo frigo, Problema compressore/perdita liquido/altra rottura, **d’ora in poi C**
* Perdita nel sistema di aria di mandata (entrata aria calda) enorme dispendio di energia per un piccolo problema di perdite che costringe il gruppo frigorifero a lavorare ad una temperatura più bassa di quella di progetto da 7 a 6-6,5°C, **d’ora in poi A**

Ai fini dell’addestramento supervisionato, si aggiungono 2 colonne al dataset, una identificativa la presenza di errore nell’istanza nominata “Classe N/P” con possibili valori N o P, l’altra identificativa la classe di errore, nominata “Classe” con possibili valori N,P,F,V,C,A seguendo la tassonomia esposta nel punto elenco precedente.

A titolo di esempio un estratto di dataset è il seguente:

| misura term. mand. | misura term. Rit. | misura term. mand. | misura term. Rit. | Temp. Ambiente | Delta pressione differenziale | Delta Pressione Filtri polveri grandi [Pa] | Delta Pressione Filtri polveri sottili [Pa] | Misuratori amperometrici | Classe | Classe N/P |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7.0 | 12.0 | 16.0 | 26.0 | 26.0 | 4.0 | 105 | 115 | 3.5 | N | N |
| 7.0 | 12.0 | 16.0 | 26.0 | 26.0 | 4.2 | 105 | 115 | 3.5 | N | N |

Il dataset così elaborato è stato caricato all'interno di un foglio gsheet esportabile in csv.

Preliminarmente il dataset viene caricato all'interno del notebook attraverso una fetch http da gsheet con la funzione

fetch\_frasal\_data (file\_url, data\_path)

import os

import tarfile

import urllib.request

DOWNLOAD\_FILE = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSL7SdxKVAe864US4WNcKQiwFNHR8qf4SG5fWy0R4wpFNX5ECGgOlrqMZ1yvGtKPG9k2lHI5caxmt\_B/pub?gid=273652227&single=true&output=csv"

DATA\_DIR = "dataset\_frasal"

DATA\_FILE = "frasal\_data.csv"

DATA\_PATH = os.path.join(DATA\_DIR,DATA\_FILE)

def fetch\_frasal\_data(file\_url=DOWNLOAD\_FILE,data\_path=DATA\_PATH):

os.makedirs(DATA\_DIR, exist\_ok=True)

urllib.request.urlretrieve(file\_url,data\_path)

fetch\_frasal\_data();

Il file viene analizzato con le librerie Panda che permette l'utilizzo della struttra data DataFrame. Il DataFrame è contenuto nella variabile data

import pandas as pd

def load\_frasal\_data(data\_path=DATA\_PATH):

return pd.read\_csv(data\_path)

data = load\_frasal\_data()

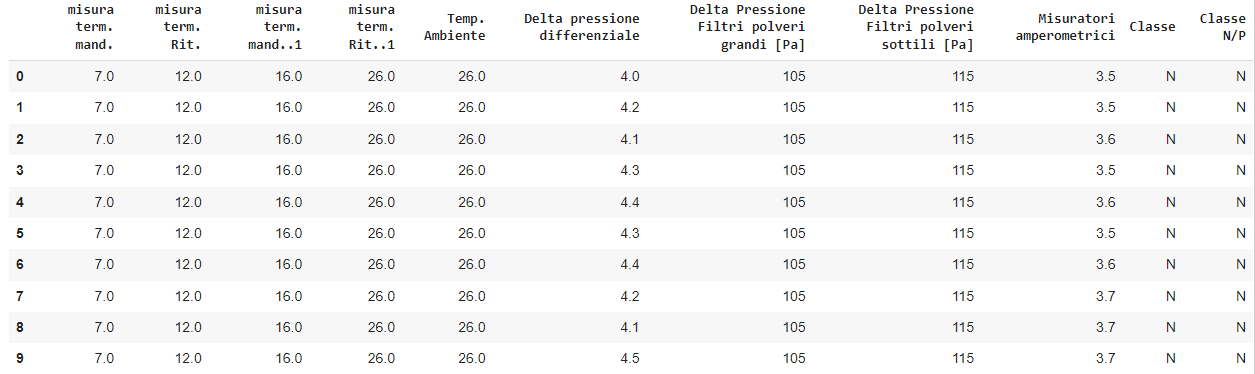
Con **head** vengono mostrate le prime 10 righe del DataFrame.

La struttura del dataset è composta da 9 attributi che definiscono l'istanza da classificare e 2 attributi per la classificazione.

Il primo attributo è la Classe che definisce la categoria del malfunzionamento:

1. **A** → Perdita nel sistema di aria di mandata (entrata aria calda)
2. **F** → Causa filtri UTA intasati, non favoriscono lo scambio termico.
3. **P** → Guasto al sistema di pompaggio, oppure perdia di acqua
4. **V** → Causa Rottura Ventole di estrazione, peggiora la fluidodinamica negli ambienti da condizionare
5. **C** → Rottura Gruppo frigo, Problema compressore/perdita liquido/altra rottura.
6. **N** → Normale

Il secondo attributo è Classe N/P che definisce la se il malfunzionamento esiste (**N**) oppure no (**P**).



Con **info** si ottiene una breve descrizione dei dati, in particolare del numero totale di righe (153), tipo di ogni attributo, eventuali valori nulli (non presenti).

Gli attributi sono tutti completi (non esistono valori nulli) inoltre sono tutti numerici. La natura dei valori è disomogenea poiché alcuni attributi rappresentano temperature altri la pressione ed uno la corrente (Misuratori amperometrici).

data.info

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 153 entries, 0 to 152

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 misura term. mand. 153 non-null float64

1 misura term. Rit. 153 non-null float64

2 misura term. mand..1 153 non-null float64

3 misura term. Rit..1 153 non-null float64

4 Temp. Ambiente 153 non-null float64

5 Delta pressione differenziale 153 non-null float64

6 Delta Pressione Filtri polveri grandi [Pa] 153 non-null int64

7 Delta Pressione Filtri polveri sottili [Pa] 153 non-null int64

8 Misuratori amperometrici 153 non-null float64

9 Classe 153 non-null object

10 Classe N/P 153 non-null object

dtypes: float64(7), int64(2), object(2)

memory usage: 13.3+ KB

L'attributo Classe rappresenta il tipo di guasto da analizzare/prevedere nel modello. La distribuzione è sbilanciata sui guasti in particolare sulla classe F (Filtri UTA intasati).

data["Classe"].value\_counts()

F 51

P 28

V 22

C 20

N 19

A 13

Name: Classe, dtype: int64

*L*'attributo Classe N/P rappresenta se esiste il guasto o meno. La distribuzione è sbilanciata sui guasti (*P* ).

data["Classe N/P"].value\_counts()

P 134

N 19

Name: Classe N/P, dtype: int64

Per l'analisi della distribuzione dei valori degli attributi che definiscono l'istanza vengono utilizzati nel seguito gli istogrammi di distribuzione su 10 bin.

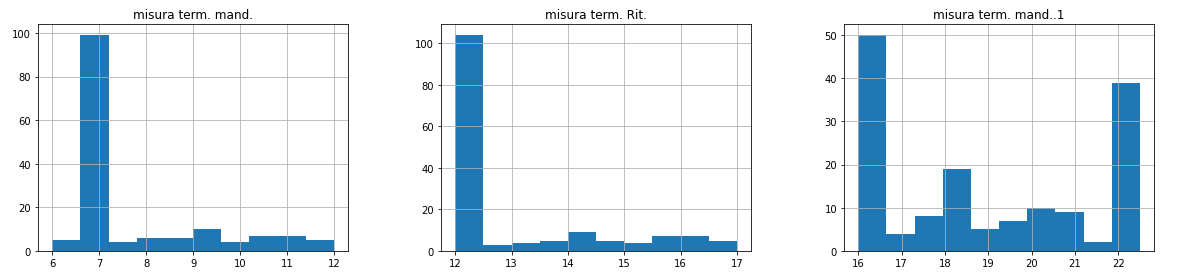
Le figure mostrano che ci sono dei valori di disribuzione dicotomiche 'standard' per le misure di pressione rappresentative la normalità e l'anomalia.

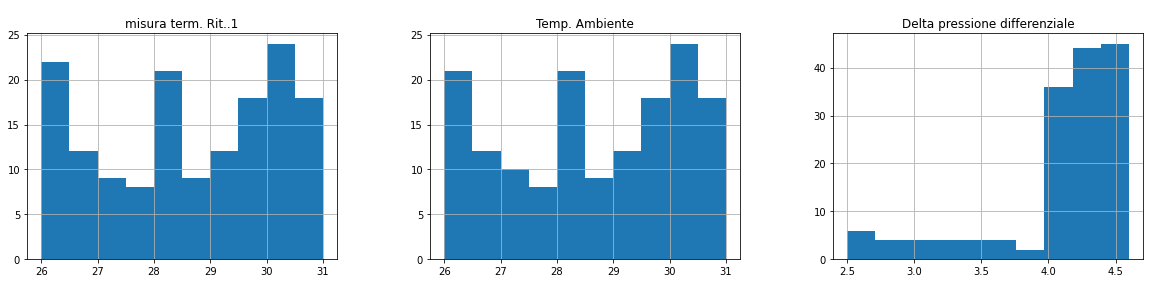
**misura term. Rit..1** e **Temperatura ambiente** hanno una distribuzione molto simile, suggerendo una stretta correlazione.

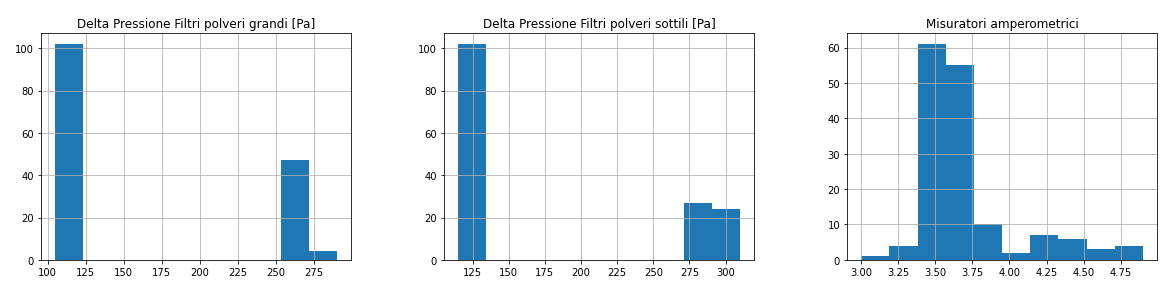
import matplotlib.pyplot as plt

data.hist(bins=10, figsize=(20,15))

plt.show()







L'analisi prosegue con la correlazione tra le variabili, attraverso l'analisi della matrice di correlazione.

Ci sono alcune feature strettamente correlate.

La correlazione può essere utile nel momento in cui alcuni valori non sono presenti in fase di test, creando artificialmente i valori mancanti.

import seaborn as sns

corr\_matrix = data.corr()

plt.figure(figsize=(15, 12))

sns.heatmap(corr\_matrix, annot=True)



Lo split dei dati è fatto mediante una funzione che prende in modo casuale il dataset di partenza e lo divide in 2 secondo la proporzione 80/20 (training/test)

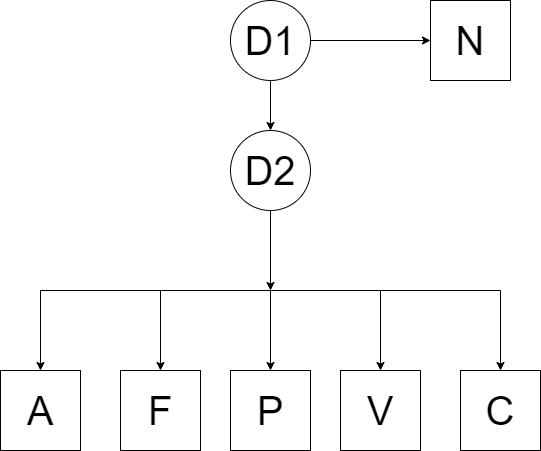
from sklearn.model\_selection import train\_test\_split

train\_set, test\_set = train\_test\_split(data,test\_size=0.2,random\_state=42

Topologia dei decisori

La topologia dell'addestramento prevede un decisore (D1) che fornisce una predizione se l'istanza analizzata è un'anomalia o meno.

A seguito dell'alert viene analizzato il contenuto dell'istanza da un altro decisore (D2) che ha lo scopo di classificare l'istanza nelle varie categorie di anomalia previste dal dataset.



Prima di procedere all'addestramento dei modelli, è necessario creare opportunamento dati e label di addestramento del primo decisore, per la classificazione binaria.

training\_data\_01 = train\_set.drop("Classe",axis=1)

training\_data\_01 = training\_data\_01.drop("Classe N/P",axis=1)

training\_data\_01 = training\_data\_01.values

training\_label\_01 = train\_set["Classe N/P"].copy()

test\_data\_01 = test\_set.drop("Classe",axis=1)

test\_data\_01 = test\_data\_01.drop("Classe N/P",axis=1)

test\_data\_01 = test\_data\_01.values

test\_label\_01 = test\_set["Classe N/P"].copy()

Il primo modello è un albero di decisione.

Nelle figure è mostrato l'output (testuale e grafico).

from sklearn import tree

tree\_clf\_01 = tree.DecisionTreeClassifier()

tree\_clf\_01 = tree\_clf\_01.fit(training\_data\_01, training\_label\_01)

text\_representation = tree.export\_text(tree\_clf\_01)

print(text\_representation)

fig = plt.figure(figsize=(10,10))

\_ = tree.plot\_tree(tree\_clf\_01)

|--- feature\_4 <= 26.25

| |--- feature\_0 <= 6.50

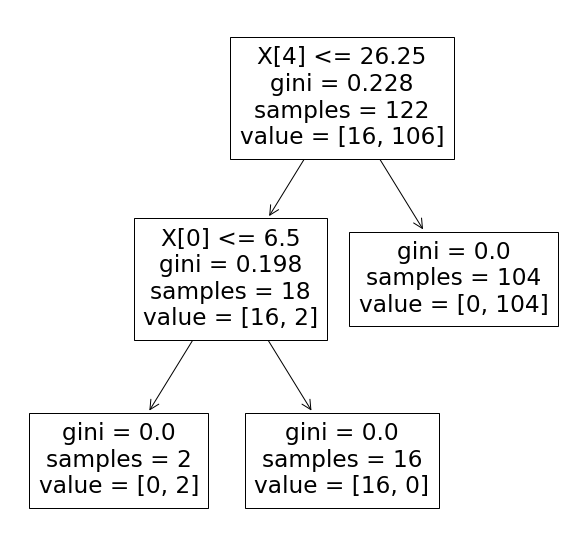
| | |--- class: P

| |--- feature\_0 > 6.50

| | |--- class: N

|--- feature\_4 > 26.25

| |--- class: P



Il secondo modello è una rete neurale multistrato con uno strato di 5 neuroni nascosti.

from sklearn.neural\_network import MLPClassifier

NN\_clf\_01 = MLPClassifier(solver='sgd',

hidden\_layer\_sizes=(5, 2),

random\_state=1,

learning\_rate\_init=0.001,

max\_iter=99999)

NN\_clf\_01 = NN\_clf\_01.fit(training\_data\_01, training\_label\_01)

Il terzo modello è una SVM lineare

from sklearn.svm import LinearSVC

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures

from sklearn.preprocessing import StandardScaler

svm\_clf\_01 = Pipeline([

("poly\_features", PolynomialFeatures(degree=3)),

("scaler", StandardScaler()),

("svm\_clf", LinearSVC(C=10, loss="hinge", random\_state=42,max\_iter=990000))

])

svm\_clf\_01 = svm\_clf\_01.fit(training\_data\_01, training\_label\_01)

Infine i classificatori vengono assemblati con un meccanismo di votazione a maggioranza: la classe più votata è quella generata dal classificatore generale.

from sklearn.ensemble import VotingClassifier

voting\_clf\_01 = VotingClassifier(

estimators=[('lr', tree\_clf\_01), ('rf', NN\_clf\_01), ('svc', svm\_clf\_01)],

voting='hard')

voting\_clf\_01 = voting\_clf\_01.fit(training\_data\_01, training\_label\_01)

Decisore multiclasse

Specularmente al decisore a singola classe, viene preparato il dataset da utilizzare per l’addestramento:

data\_01 = train\_set.drop(train\_set[train\_set.Classe=='N'].index)

training\_data = data\_01.drop("Classe",axis=1)

training\_data = training\_data.drop("Classe N/P",axis=1)

training\_data = training\_data.values

training\_label = data\_01["Classe"].copy()

tdata\_01 = test\_set.drop(test\_set[test\_set.Classe=='N'].index)

test\_data = tdata\_01.drop("Classe",axis=1)

test\_data = test\_data.drop("Classe N/P",axis=1)

test\_data = test\_data.values

test\_label = tdata\_01["Classe"].copy()

Addestramento dell’albero di decisione Id3, di cui si riporta il modello generato al termine dell’addestramento (sia in modalità testuale che grafica):

tree\_clf = tree.DecisionTreeClassifier()

tree\_clf = tree\_clf.fit(training\_data, training\_label)

text\_representation = tree.export\_text(tree\_clf)

print(text\_representation)

fig = plt.figure(figsize=(10,10))

\_ = tree.plot\_tree(tree\_clf)

|--- feature\_6 <= 180.00

| |--- feature\_5 <= 3.90

| | |--- class: P

| |--- feature\_5 > 3.90

| | |--- feature\_1 <= 12.25

| | | |--- feature\_8 <= 3.75

| | | | |--- class: A

| | | |--- feature\_8 > 3.75

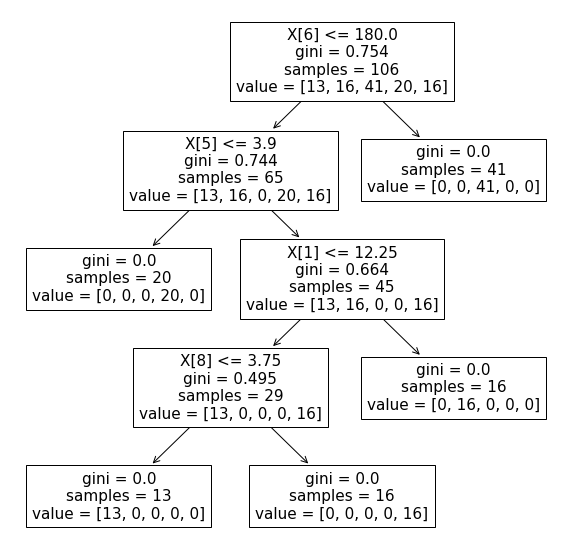
| | | | |--- class: V

| | |--- feature\_1 > 12.25

| | | |--- class: C

|--- feature\_6 > 180.00

| |--- class: F



Il secondo modello è una rete neurale con diversi parametri di cross validation e selezione della parametrizzazione più performante (mediante l’utilizzo della funzione GridSearch).

import numpy as np

import pandas as pd

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import GridSearchCV

parameters = {'solver': ['lbfgs'],

'max\_iter': [1000,1100,1200,1300,1400 ],

'alpha': 10.0 \*\* -np.arange(1, 5),

'hidden\_layer\_sizes':np.arange(5, 10),

'random\_state':[0,1]}

NN\_clf\_grid = GridSearchCV(MLPClassifier(), parameters, n\_jobs=-1)

NN\_clf\_grid.fit(training\_data, training\_label)

NN\_clf = NN\_clf\_grid.best\_estimator\_

NN\_clf = NN\_clf.fit(training\_data, training\_label)

Il terzo modello è una svm con kernel polinomiale di grado 3:

from sklearn.svm import LinearSVC

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures

from sklearn.preprocessing import StandardScaler

svm\_clf = Pipeline([

("poly\_features", PolynomialFeatures(degree=3)),

("scaler", StandardScaler()),

("svm\_clf", LinearSVC(C=10, loss="hinge", random\_state=42,max\_iter=90000))

])

svm\_clf = svm\_clf.fit(training\_data, training\_label)

Test del modello addestrato

Nel seguito viene mostrato il codice utilizzato per il test del modello con i dati dello split, ed i relativi risultati di performance.

Il test di performance viene prima eseguito sul modello basato sull’albero di decisione:

from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict

from sklearn.metrics import confusion\_matrix

print(cross\_val\_score(tree\_clf\_01,test\_data\_01,test\_label\_01,cv=3, scoring="accuracy"))

print(cross\_val\_score(tree\_clf,test\_data,test\_label,cv=3, scoring="accuracy"))

y\_train\_pred\_tree=cross\_val\_predict(tree\_clf,test\_data,test\_label,cv=3)

conf\_mx\_tree = confusion\_matrix(test\_label,y\_train\_pred\_tree)

print(conf\_mx\_tree)

plt.matshow(conf\_mx\_tree, cmap=plt.cm.gray)

plt.show()

[1. 1. 1.]

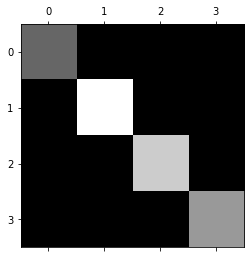
[0.9 1. 1. ]

[[ 4 0 0 0]

[ 0 10 0 0]

[ 0 0 8 0]

[ 0 0 0 6]]



La matrice di confusione mostrata, evidenzia una classificazione perfetta, senza errori di mismatch. La fold cross validation a 3 mostra un errore che dipende dallo split utilizzato e fornisce gli elementi per assumere l’assenza di overfitting.

Il secondo test di performance viene eseguito sulla rete neurale:

print(cross\_val\_score(NN\_clf\_01,test\_data\_01,test\_label\_01,cv=3, scoring="accuracy"))

print(cross\_val\_score(NN\_clf,test\_data,test\_label,cv=3, scoring="accuracy"))

y\_train\_pred\_NN=cross\_val\_predict(NN\_clf,test\_data,test\_label,cv=3)

conf\_mx\_NN = confusion\_matrix(test\_label,y\_train\_pred\_NN)

print(conf\_mx\_NN)

plt.matshow(conf\_mx\_NN, cmap=plt.cm.gray)

plt.show()

[0.90909091 0.9 0.9 ]

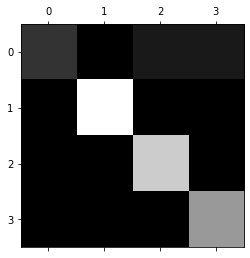
[0.8 1. 1. ]

[[ 2 0 1 1]

[ 0 10 0 0]

[ 0 0 8 0]

[ 0 0 0 6]]



Dalla matrice di confusione la rete neurale mostra dei risultati meno precisi con il vantaggio che fornisce la classificazione corretta dove l’albero di decisione aveva fatto in modo errato.

La support vector machines si caratterizza dai seguenti valori di performance:

print(cross\_val\_score(svm\_clf\_01,test\_data\_01,test\_label\_01,cv=3, scoring="accuracy"))

print(cross\_val\_score(svm\_clf,test\_data,test\_label,cv=3, scoring="accuracy"))

y\_train\_pred\_svm=cross\_val\_predict(svm\_clf,test\_data,test\_label,cv=3)

conf\_mx\_svm = confusion\_matrix(test\_label,y\_train\_pred\_svm)

print(conf\_mx\_svm)

plt.matshow(conf\_mx\_svm, cmap=plt.cm.gray)

plt.show()

[1. 0.9 0.9]

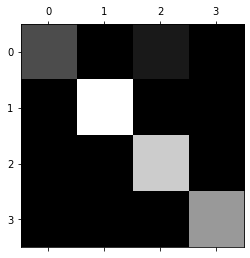
[0.9 1. 1. ]

[[ 3 0 1 0]

[ 0 10 0 0]

[ 0 0 8 0]

[ 0 0 0 6]]



La classificazione multilabel con voting fornisce dei valori molto vicini al classificatore perfetto fornendo gli elementi per affermare che il modello così realizzato è affidabile e utilizzabile in ambienti di produzione.

voting\_clf = VotingClassifier(

estimators=[('lr', tree\_clf),('nn',NN\_clf),('svc', svm\_clf)],

voting='hard')

voting\_clf = voting\_clf.fit(training\_data, training\_label)

print(cross\_val\_score(voting\_clf\_01,test\_data\_01,test\_label\_01,cv=3, scoring="accuracy"))

print(cross\_val\_score(voting\_clf,test\_data,test\_label,cv=3, scoring="accuracy"))

y\_train\_pred\_vt=cross\_val\_predict(voting\_clf,test\_data,test\_label,cv=3)

conf\_mx\_vt = confusion\_matrix(test\_label,y\_train\_pred\_vt)

print(conf\_mx\_vt)

plt.matshow(conf\_mx\_vt, cmap=plt.cm.gray)

plt.show()

[1. 1. 0.9]

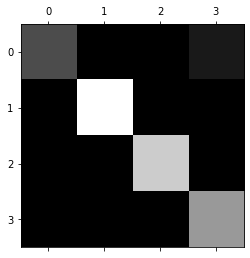
[0.9 1. 1. ]

[[ 3 0 0 1]

[ 0 10 0 0]

[ 0 0 8 0]

[ 0 0 0 6]]



Salvare e caricare il modello generato mediante servizi REST

Per salvare il modello si è fatto utilizzo della libreria joblib, che salva in modo persistente i decisori addestrati con i dati di training

from joblib import dump, load

dump(voting\_clf\_01, 'frasal\_model\_D1.joblib')

dump(voting\_clf, 'frasal\_model\_D2.joblib')

Per utilizzare il modello è necessario ricaricare i decisori addestrati e procedere in cascata in coerenza con la topologia descritta precedentemente.

predict\_clf\_01 = load('frasal\_model\_D1.joblib')

predict\_clf = load('frasal\_model\_D2.joblib')

st = [7.5, 12.5, 16.5, 26.5, 26.5, 4.2, 105, 115, 3.5]

val = predict\_clf\_01.predict([st])[0]

if (val=='P'):

print(predict\_clf.predict([st]))

else:

print(val);

Per utilizzare il modello all'interno di un servizio REST in FLASK, si suggerisce di installare la versione di scikitlearn 1.0.2

pip install scikit-learn==1.0.2

All'interno dell'applicazione FLASK inserire il seguente codice di esempio che espone 2 servizi in get, uno per l’addestramento l’altro per la classificazione:

# url per l'interrogazione di test

# http://localhost:5000/?x1=7.5&x2=12.5&x3=16.5&x4=26.5&x5=26.5&x6=4.2&x7=105&x8=115&x9=3.5

#

# url per l'addestramento

# http://localhost:5000/learn

import flask

from flask import request, jsonify

from joblib import dump, load

import os

import tarfile

import urllib.request

import pandas as pd

from sklearn.svm import LinearSVC

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import VotingClassifier

from sklearn import tree

from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict

import logging

DOWNLOAD\_FILE = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSL7SdxKVAe864US4WNcKQiwFNHR8qf4SG5fWy0R4wpFNX5ECGgOlrqMZ1yvGtKPG9k2lHI5caxmt\_B/pub?gid=273652227&single=true&output=csv"

DATA\_DIR = "dataset\_frasal"

DATA\_FILE = "frasal\_data.csv"

DATA\_PATH = os.path.join(DATA\_DIR,DATA\_FILE)

app = flask.Flask(\_\_name\_\_)

@app.route('/', methods=['GET'])

def home():

x1 = request.args.get('x1')

x2 = request.args.get('x2')

x3 = request.args.get('x3')

x4 = request.args.get('x4')

x5 = request.args.get('x5')

x6 = request.args.get('x6')

x7 = request.args.get('x7')

x8 = request.args.get('x8')

x9 = request.args.get('x9')

predict\_clf\_D1 = load('frasal\_model\_D1.joblib')

predict\_clf\_D2 = load('frasal\_model\_D2.joblib')

st = [x1, x2, x3, x4, x5, x6, x7, x8, x9]

val = predict\_clf\_D1.predict([st]).flatten()[0]

if (val == 'P'):

val = predict\_clf\_D2.predict([st]).flatten()[0]

return jsonify(val)

def fetch\_frasal\_data(file\_url=DOWNLOAD\_FILE,data\_path=DATA\_PATH):

os.makedirs(DATA\_DIR, exist\_ok=True)

# urllib.request.urlretrieve(file\_url,data\_path)

def load\_frasal\_data(data\_path=DATA\_PATH):

return pd.read\_csv(data\_path)

@app.route('/learn', methods=['GET'])

def learnModel():

fetch\_frasal\_data()

data = load\_frasal\_data()

train\_set, test\_set = train\_test\_split(data, test\_size=0.2, random\_state=42)

training\_data\_01 = train\_set.drop("Classe", axis=1)

training\_data\_01 = training\_data\_01.drop("Classe N/P", axis=1)

training\_data\_01 = training\_data\_01.values

training\_label\_01 = train\_set["Classe N/P"].copy()

test\_data\_01 = test\_set.drop("Classe", axis=1)

test\_data\_01 = test\_data\_01.drop("Classe N/P", axis=1)

test\_data\_01 = test\_data\_01.values

test\_label\_01 = test\_set["Classe N/P"].copy()

tree\_clf\_01 = tree.DecisionTreeClassifier()

tree\_clf\_01 = tree\_clf\_01.fit(training\_data\_01, training\_label\_01)

NN\_clf\_01 = MLPClassifier(solver='sgd',

hidden\_layer\_sizes=(5, 2),

random\_state=1,

learning\_rate\_init=0.001,

max\_iter=99999)

NN\_clf\_01 = NN\_clf\_01.fit(training\_data\_01, training\_label\_01)

svm\_clf\_01 = Pipeline([

("poly\_features", PolynomialFeatures(degree=3)),

("scaler", StandardScaler()),

("svm\_clf", LinearSVC(C=10, loss="hinge", random\_state=42, max\_iter=990000))

])

svm\_clf\_01 = svm\_clf\_01.fit(training\_data\_01, training\_label\_01)

voting\_clf\_01 = VotingClassifier(

estimators=[('lr', tree\_clf\_01), ('rf', NN\_clf\_01), ('svc', svm\_clf\_01)],

voting='hard')

voting\_clf\_01 = voting\_clf\_01.fit(training\_data\_01, training\_label\_01)

data\_01 = train\_set.drop(train\_set[train\_set.Classe == 'N'].index)

training\_data = data\_01.drop("Classe", axis=1)

training\_data = training\_data.drop("Classe N/P", axis=1)

training\_data = training\_data.values

training\_label = data\_01["Classe"].copy()

tdata\_01 = test\_set.drop(test\_set[test\_set.Classe == 'N'].index)

test\_data = tdata\_01.drop("Classe", axis=1)

test\_data = test\_data.drop("Classe N/P", axis=1)

test\_data = test\_data.values

test\_label = tdata\_01["Classe"].copy()

tree\_clf = tree.DecisionTreeClassifier()

tree\_clf = tree\_clf.fit(training\_data, training\_label)

parameters = {'solver': ['lbfgs'],

'max\_iter': [5000, 10000, 40000, 100000, 900000],

'alpha': 10.0 \*\* -np.arange(1, 5),

'hidden\_layer\_sizes': np.arange(5, 10),

'random\_state': [0, 1]}

NN\_clf\_grid = GridSearchCV(MLPClassifier(), parameters, n\_jobs=-1)

NN\_clf\_grid.fit(training\_data, training\_label)

NN\_clf = NN\_clf\_grid.best\_estimator\_

NN\_clf = NN\_clf.fit(training\_data, training\_label)

svm\_clf = Pipeline([

("poly\_features", PolynomialFeatures(degree=3)),

("scaler", StandardScaler()),

("svm\_clf", LinearSVC(C=10, loss="hinge", random\_state=42, max\_iter=90000))

])

svm\_clf = svm\_clf.fit(training\_data, training\_label)

voting\_clf = VotingClassifier(

estimators=[('lr', tree\_clf), ('nn', NN\_clf), ('svc', svm\_clf)],

voting='hard')

voting\_clf = voting\_clf.fit(training\_data, training\_label)

dump(voting\_clf\_01, 'frasal\_model\_D1.joblib')

dump(voting\_clf, 'frasal\_model\_D2.joblib')

out\_1 = (cross\_val\_score(voting\_clf\_01, test\_data\_01, test\_label\_01, cv=3, scoring="accuracy"))

out\_2 = (cross\_val\_score(voting\_clf, test\_data, test\_label, cv=3, scoring="accuracy"))

str\_1 = np.array2string(out\_1, precision=2, separator=',',suppress\_small=True)

str\_2 = np.array2string(out\_2, precision=2, separator=',', suppress\_small=True)

return "LEARNED: "+str\_1+" - "+str\_2;

app.run()