Apprendimento automatico per rilevamento anomalie e monitoraggio delle condizioni

Un'esercitazione dettagliata dall'importazione dei dati all'output del modello

Vegard Flovik
Apr 23 · 10 min read ★

Il mio precedente articolo sul rilevamento delle anomalie e il monitoraggio delle condizioni ha ricevuto molti feedback. Molte delle domande che ricevo riguardano gli aspetti tecnici e come impostare i modelli, ecc. A causa di ciò, ho deciso di scrivere un articolo di follow-up che copra in dettaglio tutti i passaggi necessari, dalla pre-elaborazione dei dati alla costruzione di modelli e visualizzazione dei risultati.

Per un'introduzione al rilevamento delle anomalie e al monitoraggio delle condizioni, raccomando prima di leggere il mio articolo originale sull'argomento. Ciò fornisce le

informazioni di base necessarie su come è possibile utilizzare l'apprendimento automatico e l'analisi basata sui dati per estrarre informazioni preziose dai dati dei sensori.

L'articolo attuale si concentra principalmente sugli aspetti tecnici e include tutto il codice necessario per impostare modelli di rilevamento delle anomalie basati su analisi statistiche multivariate e reti neurali autoencoder.

. . .

Scarica il set di dati:

Per replicare i risultati nell'articolo originale, è innanzitutto necessario scaricare il set di dati dal database Acustica e vibrazioni della NASA . Consulta il documento Readme scaricato per i dati relativi ai cuscinetti IMS per ulteriori informazioni sull'esperimento e sui dati disponibili.

Ogni set di dati è costituito da singoli file che sono istantanee del segnale di vibrazione di 1 secondo registrate a intervalli specifici. Ogni file è composto da 20.480 punti con la frequenza di campionamento impostata su 20 kHz. Il nome del file indica quando sono stati raccolti i dati. Ogni record (riga) nel file di dati è un punto dati. Intervalli più grandi di timestamp (indicati nei nomi dei file) indicano la ripresa dell'esperimento il giorno lavorativo successivo.

Importa pacchetti e librerie:

Il primo passo è importare alcuni pacchetti e librerie utili per l'analisi:

```
# Importazioni comuni
import os
import panda come pd
import numpy come np
da sklearn import pre-elaborazione
import seaborn come sns
 sns.set (color_codes = True )
import matplotlib.pyplot come plt
 % matplotlib inline

da numpy.random import seed
da tensorflow import set_random_seed

da keras .layers importano Input, Dropout
da keras.layers.core import Dense
da keras.models import Model, Sequential, load_model
```

```
from keras importa regolarizzatori
da keras.models import model_from_json
```

Caricamento e pre-elaborazione dei dati:

Un presupposto è che la degradazione degli ingranaggi si verifichi gradualmente nel tempo, quindi nell'analisi seguente utilizziamo un punto dati ogni 10 minuti. Ogni punto dati di 10 minuti viene aggregato utilizzando il valore assoluto medio delle registrazioni di vibrazione sui 20.480 punti dati in ciascun file. Quindi uniamo tutto in un singolo frame di dati.

Nel seguente esempio, utilizzo i dati del test di guasto della 2a marcia (vedi il documento readme per ulteriori informazioni su quell'esperimento).

```
data_dir = '2nd_test'
merged_data = pd.DataFrame ()

per il nome file in os.listdir (data_dir):
    print (nome file)
    dataset = pd.read_csv (os.path.join (data_dir, nome file), sep =
' \ t ')
    dataset_mean_abs = np.array (dataset.abs (). media ())
    dataset_mean_abs = pd.DataFrame (dataset_mean_abs.reshape (1,4))
    dataset_mean_abs.index = [nome file]
    merged_data = merged_data.append (dataset_mean_abs)

merged_data.columns = ['Cuscinetto 1', 'Cuscinetto 2', 'Cuscinetto
3', 'Cuscinetto 4']
```

Dopo aver caricato i dati di vibrazione, trasformiamo l'indice nel formato datetime (utilizzando la seguente convenzione), quindi ordiniamo i dati per indice in ordine cronologico prima di salvare il set di dati unito come file .csv

```
merged_data.index = pd.to_datetime (merged_data.index, format = '%
Y.% m. % d .% H.% M.% S')

merged_data = merged_data.sort_index ()
merged_data.to_csv ('merged_dataset_BearingTest_2.csv' )
merged_data.head ()
```

	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.058333	0.071832	0.083242	0.043067
2004-02-12 10:42:39	0.058995	0.074006	0.084435	0.044541

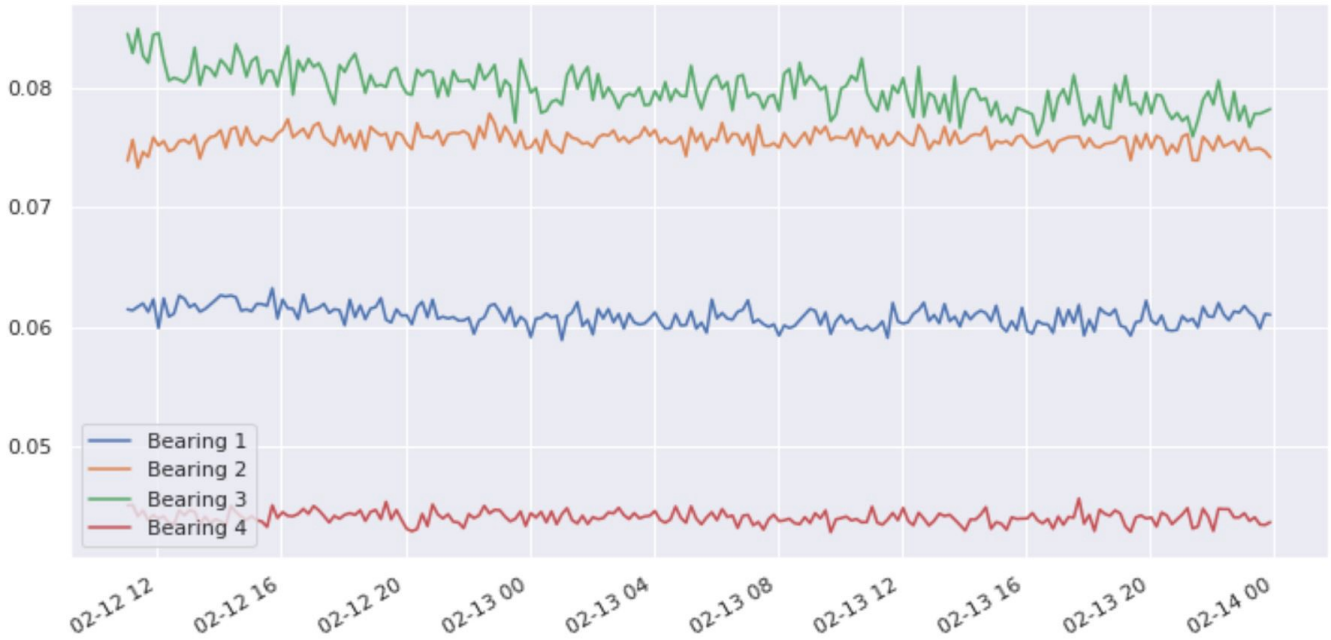
2004-02-12 10:52:39	0.060236	0.074227	0.083926	0.044443
2004-02-12 11:02:39	0.061455	0.073844	0.084457	0.045081
2004-02-12 11:12:39	0.061361	0.075609	0.082837	0.045118

Dataframe risultante: "merged_data"

Definire i dati del treno / test:

Prima di impostare i modelli, è necessario definire i dati treno / test. Per fare ciò, eseguiamo una semplice divisione in cui ci alleniamo sulla prima parte del set di dati (che dovrebbe rappresentare le normali condizioni operative) e testiamo sulle parti rimanenti del set di dati che portano al guasto del cuscinetto.

```
dataset_train = merged_data ['2004-02-12 11:02:39': '2004-02-13
23:52:39']
dataset_test = merged_data ['2004-02-13 23:52:39':]

dataset_train. trama (figsize = (12,6))
```



Dati di allenamento: condizioni operative normali

Normalizzare i dati:

Quindi uso gli strumenti di preelaborazione di Scikit-learn per ridimensionare le variabili di input del modello. "MinMaxScaler" ridimensiona semplicemente i dati in modo che siano nell'intervallo [0,1].

```
scaler = preprocessing.MinMaxScaler ()

X_train = pd.DataFrame (scaler.fit_transform (dataset_train),
                                colonne = dataset_train.columns,
                                index = dataset_train.index)

# Dati di allenamento casuali
 X_train.sample (frac = 1)

X_test (scaler.transform (dataset_test),
                               colonne = dataset_test.columns,
                               index = dataset_test.index)
```

Modello di tipo PCA per il rilevamento di anomalie:

Poiché la gestione di dati di sensori ad alta dimensione è spesso impegnativa, esistono diverse tecniche per ridurre il numero di variabili (riduzione della dimensionalità). Una delle tecniche principali è l' analisi dei componenti principali (PCA). Per un'introduzione più dettagliata, mi riferisco al mio articolo originale sull'argomento.

Come tentativo iniziale, comprimiamo le letture del sensore fino ai due componenti principali principali.

```
da  sklearn.decomposition  import PCA

pca = PCA (n_components = 2, svd_solver = 'full')

X_train_PCA = pca.fit_transform (X_train)
X_train_PCA = pd.DataFrame (X_train_PCA)
X_train_PCA.index =

X_train = Px )
X_test_PCA = pd.DataFrame (X_test_PCA)
X_test_PCA.index = X_test.index
```

La metrica della distanza di Mahalanobis:

La distanza di Mahalanobis è ampiamente utilizzata nell'analisi dei cluster e nelle tecniche di classificazione. Al fine di utilizzare la distanza di Mahalanobis per classificare un punto di prova come appartenente a una delle classi N, si calcola innanzitutto la matrice di covarianza di ciascuna classe, generalmente basata su campioni noti per appartenere a ciascuna classe. Nel nostro caso, poiché siamo interessati solo a classificare “normale” vs “anomalia”, utilizziamo i dati di

interessati solo a classificare "normale" vs "anomalia", utilizziamo i dati di addestramento che contengono solo le normali condizioni operative per calcolare la matrice di covarianza.Quindi, dato un campione di prova, calcoliamo la distanza di Mahalanobis alla classe "normale" e classifichiamo il punto di prova come "anomalia" se la distanza supera una certa soglia.

Per un'introduzione più dettagliata di questi aspetti tecnici, puoi dare un'occhiata al mio precedente articolo , che tratta questi argomenti in modo più dettagliato.

Definire le funzioni utilizzate nel modello PCA:

Calcola la matrice di covarianza:

```
def cov_matrix (dati, verbose = False ):
    covariance_matrix = np.cov (dati, rowvar = False )
    se is_pos_def (covariance_matrix):
        inv_covariance_matrix = np.linalg.inv (covariance_matrix)
        se is_pos_def (inv_covariance_matrix):
            ritorno covariance_matrix, inv_covariance_matrix
        altro :
            print ("Errore: Inverse of Covariance Matrix non è
definito positivo!")
    altrimenti :
        print ("Errore: Covariance Matrix non è definito positivo!")
```

Calcola la distanza di Mahalanobis:

```
def MahalanobisDist (inv_cov_matrix, mean_distr, dati, verbose =
False ):
    inv_covariance_matrix = inv_cov_matrix
    vars_mean = mean_distr
    diff = dati - vars_mean
    md = []
    per i in range (len (diff)):
        md.append (np.sqrt (diff [ i] .dot (inv_covariance_matrix)
.dot (diff [i])))
    return md
```

Rilevamento dei valori anomali:

```
def MD_detectOutliers (dist, extreme = False , verbose = False ):
    k = 3. if extreme else 2.
    soglia = np.mean (dist) * k
    outlier = []
    for i in range (len (dist)):
```

```
        if dist [i]> = soglia:
            outliers.append (i)   # indice del
    ritorno anomalo np.array (outlier)
```

Calcola il valore di soglia per classificare il punto dati come anomalia:

```
def MD_threshold (dist, extreme = False , verbose = False ):
    k = 3. if extreme else 2.
    soglia = np.mean (dist) * k soglia di
    ritorno
```

Verifica se la matrice è definita positiva:

```
def is_pos_def (A):
    if np.allclose (A, AT):
        provare :
            np.linalg.cholesky (A)
            return  True
        tranne np.linalg.LinAlgError:
            return  False
    altro :
        return  False
```

Imposta il modello PCA:

Definire il treno / set di test dai due componenti principali principali:

```
data_train = np.array (X_train_PCA.values)
data_test = np.array (X_test_PCA.values)
```

Calcola la matrice di covarianza e il suo inverso, in base ai dati nel set di addestramento:

```
cov_matrix, inv_cov_matrix = cov_matrix (data_train)
```

Calcoliamo anche il valore medio per le variabili di input nel set di addestramento, poiché questo viene utilizzato in seguito per calcolare la distanza di Mahalanobis rispetto ai punti dati nel set di test

```
mean_distr = data_train.mean (axis = 0)
```

Usando la matrice di covarianza e il suo inverso, possiamo calcolare la distanza di Mahalanobis per i dati di allenamento che definiscono "condizioni normali" e trovare il valore di soglia per contrassegnare i punti dati come anomalia. È quindi possibile calcolare la distanza di Mahalanobis per i punti dati nel set di test e confrontarla con la soglia di anomalia.

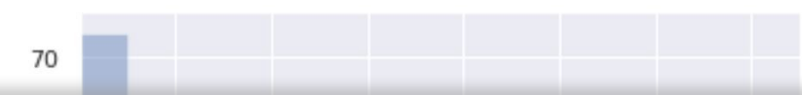
```
dist_test = MahalanobisDist (inv_cov_matrix, mean_distr, data_test,
verbose = False )
dist_train = MahalanobisDist (inv_cov_matrix, mean_distr,
data_train, verbose = False )
soglia = MD_threshold (dist_train, extreme = True )
```

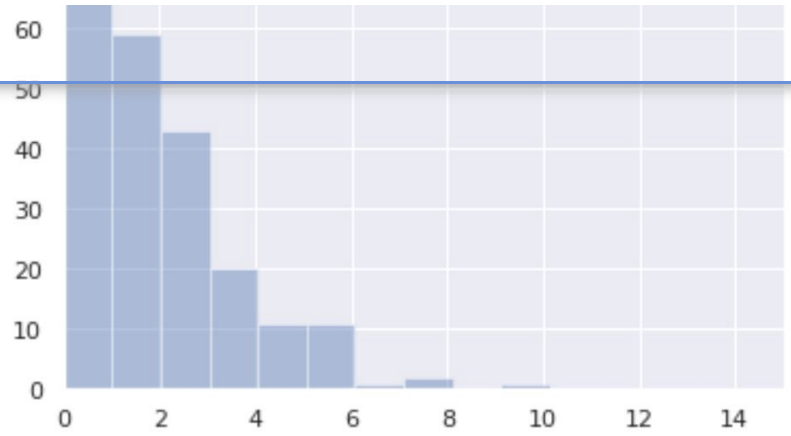
Valore soglia per segnalare un'anomalia:

Il quadrato della distanza di Mahalanobis dal centroide della distribuzione dovrebbe seguire una distribuzione χ2 se si assume che le normali variabili di input distribuite siano soddisfatte. Questo è anche il presupposto alla base del calcolo di cui sopra del "valore di soglia" per la segnalazione di un'anomalia. Poiché questa ipotesi non è necessariamente soddisfatta nel nostro caso, è utile visualizzare la distribuzione della distanza di Mahalanobis per impostare un buon valore di soglia per le anomalie di segnalazione. Ancora una volta, mi riferisco al mio precedente articolo , per un'introduzione più dettagliata a questi aspetti tecnici.

Iniziamo visualizzando il quadrato della distanza di Mahalanobis, che dovrebbe idealmente seguire una distribuzione χ2.

```
plt.figure ()
sns.distplot (np.square (dist_train),
             bin = 10,
             kde = False );
plt.xlim ([0.0,15])
```

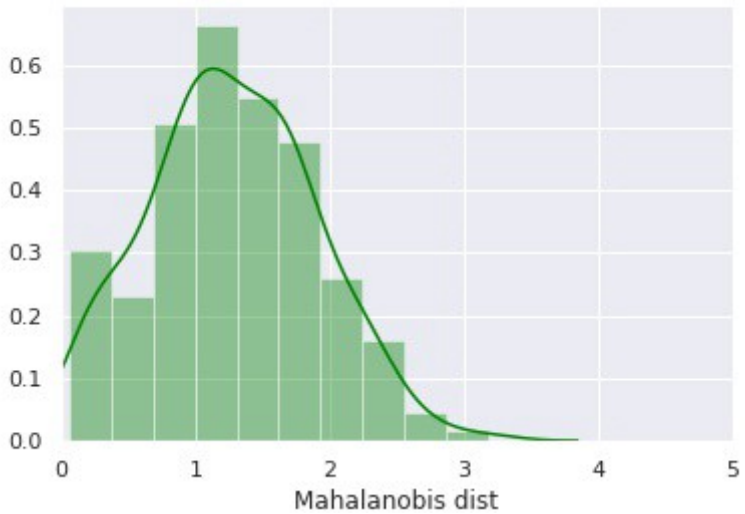




Piazza della distanza Mahalanobis

Quindi visualizzare la distanza Mahalanobis stessa:

```
plt.figure ()
sns.distplot (dist_train,
              bins = 10,
              kde = True ,
             color = 'green');
plt.xlim ([0.0,5])
plt.xlabel ('Mahalanobis dist')
```



Dalle suddette distribuzioni, il valore di soglia calcolato di 3,8 per la segnalazione di un'anomalia sembra ragionevole (definito come 3 deviazioni standard dal centro della distribuzione)

Possiamo quindi salvare la distanza di Mahalanobis, nonché il valore di soglia e la variabile "flag di anomalia" sia per il treno che per i dati di test in un frame di dati:

```
anomaly_train = pd.DataFrame ()
anomaly_train ['Mob dist'] = dist_train
anomaly_train ['Thresh'] = soglia
# Se Mob dist sopra la soglia: Contrassegna come anomalia
 anomaly_train ['Anomaly'] = anomaly_train ['Mob dist']>
anomaly_train ['Thresh']
anomaly_train.index = X_train_PCA.index

anomaly = pd.DataFrame ()
anomaly ['Mob dist'] = dist_test
anomaly ['Thresh'] = soglia
# Se Mob dist sopra la soglia: Contrassegna come anomalia
 anomalia ['Anomalia '] = anomaly [' Mob dist ']> anomaly [' Thresh
']
anomaly.index = X_test_PCA.index
anomaly.head ()
```

	Mob dist	Thresh	Anomaly
2004-02-13 23:52:39	1.032676	3.812045	False
2004-02-14 00:02:39	1.148163	3.812045	False
2004-02-14 00:12:39	1.509998	3.812045	False
2004-02-14 00:22:39	1.849725	3.812045	False
2004-02-14 00:32:39	0.701075	3.812045	False

Dataframe risultante per i dati di test

Sulla base delle statistiche calcolate, qualsiasi distanza superiore al valore di soglia verrà contrassegnata come anomalia.

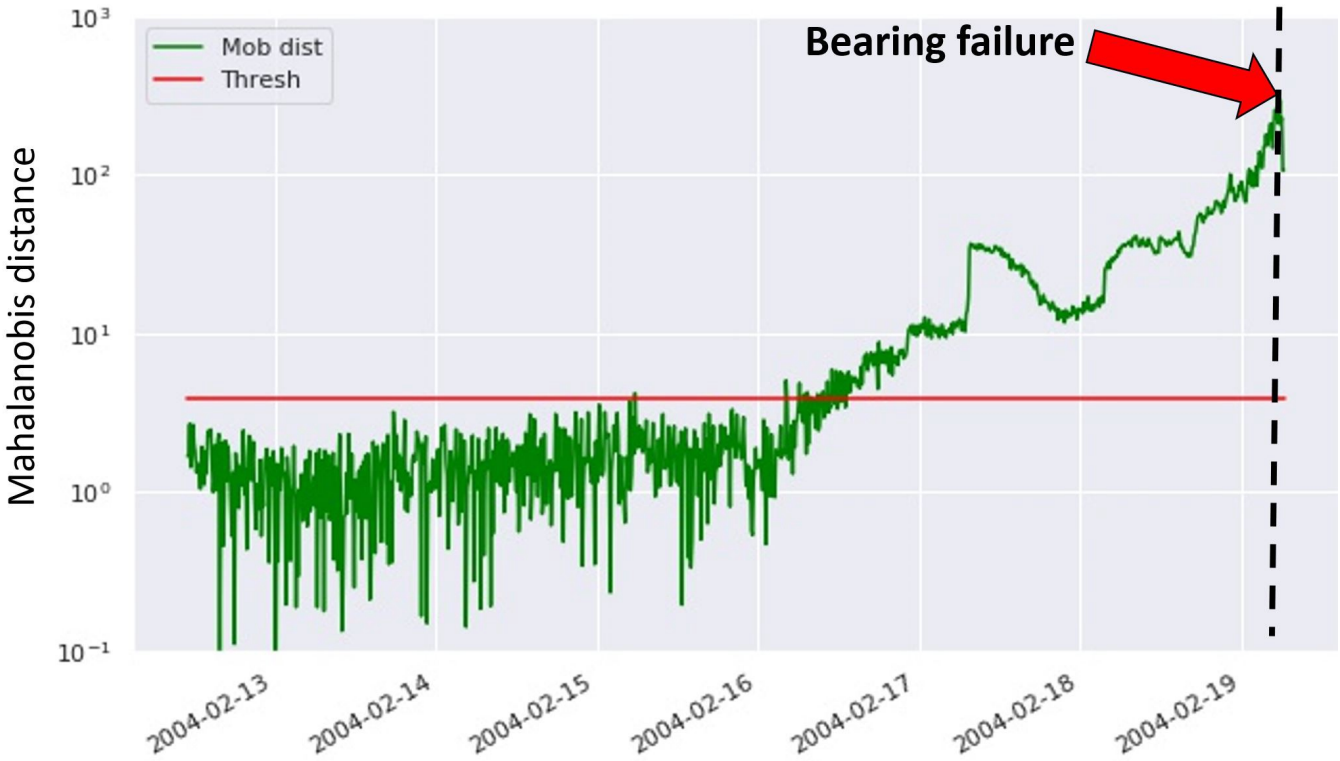
Ora possiamo unire i dati in un singolo frame di dati e salvarli come file .csv:

```
anomaly_alldata = pd.concat ([anomaly_train, anomaly])
anomaly_alldata.to_csv ('Anomaly_distance.csv')
```

Verifica del modello PCA sui dati di test:

Ora possiamo tracciare la metrica di anomalia calcolata (Mob dist) e controllare quando attraversa la soglia di anomalia (notare l'asse y logaritmico).

```
anomaly_alldata.plot (logy = True , figsize = (10,6), ylim = [1e-
1,1e3], color = ['green', 'red'])
```



Dalla figura sopra, vediamo che il modello è in grado di rilevare l'anomalia circa 3 giorni prima dell'effettivo guasto del cuscinetto.

Altro approccio: modello di Autoencoder per il rilevamento di anomalie

L'idea di base qui è quella di utilizzare una rete neurale autoencoder per "comprimere" le letture del sensore in una rappresentazione a bassa dimensione, che cattura le correlazioni e le interazioni tra le varie variabili. (Sostanzialmente lo stesso principio del modello PCA, ma qui permettiamo anche di non linearità tra le variabili di input).

Per un'introduzione più dettagliata ai codificatori automatici puoi dare un'occhiata al mio articolo precedente , che tratta l'argomento in modo più dettagliato.

Definizione della rete del codificatore automatico:

Usiamo una rete neurale a 3 strati: il primo strato ha 10 nodi, il livello intermedio ha 2 nodi e il terzo livello ha 10 nodi. Usiamo l'errore quadratico medio come funzione di perdita e formiamo il modello usando l' ottimizzatore "Adam".

```
seed (10)
```

```
set_random_seed (10)
act_func = 'elu'

# Input layer:
 model = Sequential ()
# Primo layer nascosto, collegato al vettore di input X.
 model.add (Dense (10, activation = act_func,
                kernel_initializer = 'glorot_uniform' ,
                kernel_regularizer = regularizers.l2 (0.0),
                input_shape = (X_train.shape [1],)
               )
         )

model.add (Dense (2, activation = act_func,
                kernel_initializer = 'glorot_uniform'))

model.add (Dense (10, Activation = act_func,
                kernel_initializer = 'glorot_uniform'))

model.add (Dense (X_train.shape [1],
                kernel_initializer = 'glorot_uniform'))

model.compile (loss = 'mse', optimizer = 'adam')

# Modello di treno per 100 epoche, dimensione del lotto di 10:
 NUM_EPOCHS = 100
BATCH_SIZE = 10
```

Montaggio del modello:

Per tenere traccia dell'accuratezza durante l'allenamento, utilizziamo il 5% dei dati di allenamento per la convalida dopo ogni epoca (validation_split = 0.05)

```
history = model.fit (np.array (X_train), np.array (X_train),
                  batch_size = BATCH_SIZE,
                  epochs = NUM_EPOCHS,
                  validation_split = 0.05,
                  verbose = 1)
```

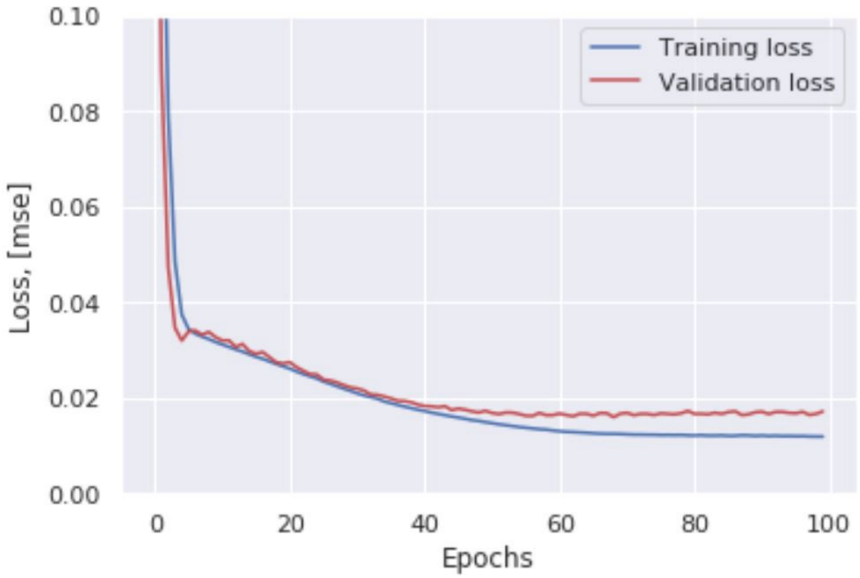
```
Train on 210 samples, validate on 12 samples
Epoch 1/100
210/210 [==============================] - 1s 5ms/step - loss: 0.2798 - val_loss: 0.1875
Epoch 2/100
210/210 [==============================] - 0s 167us/step - loss: 0.1489 - val_loss: 0.0885
Epoch 3/100
210/210 [==============================] - 0s 182us/step - loss: 0.0794 - val_loss: 0.0474
Epoch 4/100
210/210 [==============================] - 0s 134us/step - loss: 0.0486 - val_loss: 0.0347
Epoch 5/100
210/210 [==============================] - 0s 156us/step - loss: 0.0375 - val_loss: 0.0320
Epoch 6/100
210/210 [==============================] - 0s 145us/step - loss: 0.0344 - val_loss: 0.0339
Epoch 7/100
210/210 [==============================] - 0s 138us/step - loss: 0.0334 - val_loss: 0.0341
Epoch 8/100
```

```
210/210 [==============================] - 0s 141us/step - loss: 0.0329 - val_loss: 0.0332
Epoch 9/100
210/210 [==============================] - 0s 139us/step - loss: 0.0323 - val_loss: 0.0338
```

Processo di formazione

Visualizza perdita di addestramento / convalida:

```
plt.plot (history.history ['loss'],
          'b',
          label = 'Perdita dell'allenamento')
plt.plot (history.history ['val_loss'],
          'r',
          label = 'Perdita di validazione')
plt. legenda (loc = 'in alto a destra')
plt.xlabel ('Epochs')
plt.ylabel ('Loss, [mse]')
plt.ylim ([0, .1])
plt.show ()
```



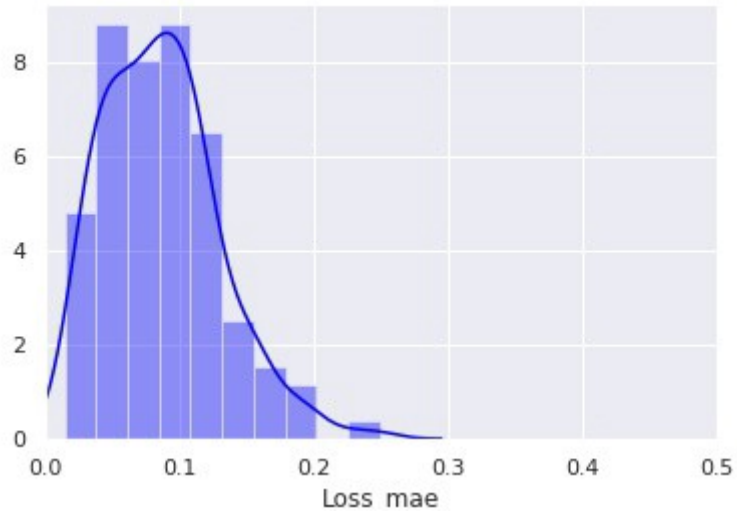
Perdita di treno / convalida

Distribuzione della funzione di perdita nel set di allenamento:

Tracciando la distribuzione della perdita calcolata nel set di addestramento, è possibile utilizzarlo per identificare un valore di soglia adatto per identificare un'anomalia. Nel fare ciò, è possibile assicurarsi che questa soglia sia impostata al di sopra del "livello di rumore" e che eventuali anomalie contrassegnate debbano essere statisticamente significative al di sopra del rumore di fondo.

```
X_pred = model.predict (np.array (X_train))
X_pred = pd.DataFrame (X_pred
```

```
X_pred = pd.DataFrame (X_pred,
                       colonne = X_train.columns)
X_pred.index = X_train.index

segnato = pd.DataFrame (indice = X_train.index)
segnato ['Loss_mae '] = np.mean (np.abs (X_pred-X_train), axis = 1)

plt.figure ()
sns.distplot (segnato [' Loss_mae '],
              bin = 10,
              kde = True ,
             color =' blue ') ;
plt.xlim ([0,0, 0,5])
```



Distribuzione delle perdite, set di allenamento

Dalla suddetta distribuzione delle perdite, proviamo una soglia di 0,3 per segnalare un'anomalia. Possiamo quindi calcolare la perdita nel set di test, per verificare quando l'uscita supera la soglia di anomalia.

```
X_pred = model.predict (np.array (X_test))
X_pred = pd.DataFrame (X_pred,
                       colonne = X_test.columns)
X_pred.index = X_test.index

segnato = pd.DataFrame (indice = X_test.index)
segnato ['Loss_mae '] = np.mean (np.abs (X_pred-X_test), asse = 1)
segnato [' Soglia '] = 0,3
segnato [' Anomalia '] = segnato [' Loss_mae ']> segnato [' Soglia
']
segnato. testa()
```

Loss_mae Threshold Anomaly

	Loss_mae	Threshold	Anomaly
2004-02-13 23:52:39	0.132962	0.3	False
2004-02-14 00:02:39	0.117404	0.3	False
2004-02-14 00:12:39	0.034947	0.3	False
2004-02-14 00:22:39	0.143995	0.3	False
2004-02-14 00:32:39	0.071100	0.3	False

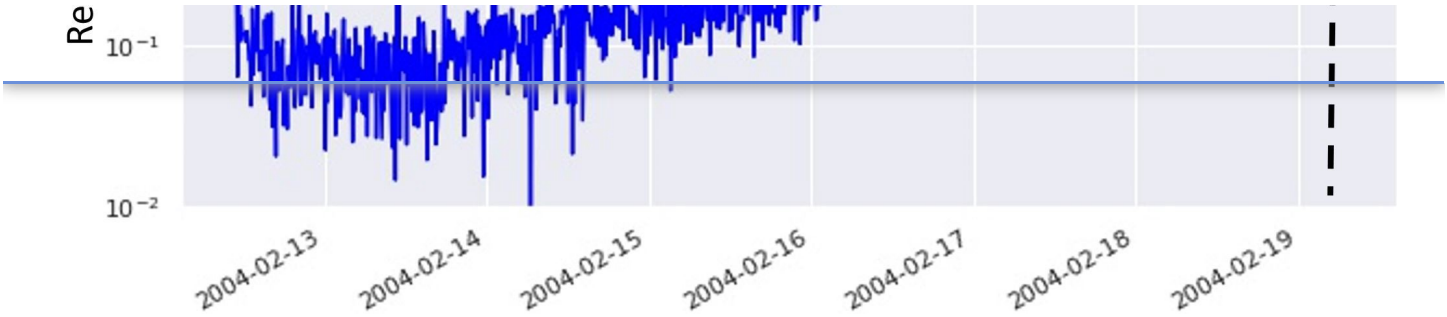
Quindi calcoliamo le stesse metriche anche per il set di formazione e uniamo tutti i dati in un singolo frame di dati:

```
X_pred_train = model.predict (np.array (X_train))
X_pred_train = pd.DataFrame (X_pred_train,
                      colonne = X_train.columns)
X_pred_train.index = X_train.index

score_train =
pd.DataFtrain_exex '] = np.mean (np.abs (X_pred_train-X_train), asse
= 1)
score_train [' Threshold '] = 0.3
score_train [' Anomaly '] = score_train [' Loss_mae ']> score_train
[' Threshold ']

score = pd.concat ([score_train, segnato])
```

Risultati del modello Autoencoder:

Dopo aver calcolato la distribuzione delle perdite e la soglia di anomalia, possiamo visualizzare l'uscita del modello nel tempo che porta al fallimento del cuscinetto:

```
score.plot (logy = True , figsize = (10,6), ylim = [1e-2,1e2], color
= ['blue', 'red'])
```



Sommario:

Entrambi gli approcci di modellazione danno risultati simili, in cui sono in grado di segnalare il malfunzionamento del cuscinetto imminente con largo anticipo rispetto al guasto effettivo. La differenza principale è essenzialmente come definire un valore di soglia adatto per segnalare anomalie, per evitare molti falsi positivi durante le normali condizioni operative.

Spero che questo tutorial ti abbia dato l'ispirazione per provare tu stesso questi modelli di rilevamento delle anomalie. Dopo aver impostato correttamente i modelli, è tempo di iniziare a sperimentare i parametri del modello, ecc. E testare lo stesso approccio su nuovi set di dati. Se ti imbatti in alcuni casi d'uso interessanti, per favore fatemi sapere nei commenti qui sotto.

Divertiti!

Altri articoli:

Se hai trovato interessante questo articolo, potrebbero interessarti anche alcuni dei miei altri articoli:

1. Apprendimento automatico: dall'hype alle applicazioni del mondo reale
2. Il rischio nascosto di AI e Big Data
3. Come utilizzare l'apprendimento automatico per il rilevamento di anomalie e il monitoraggio delle condizioni
4. AI per la gestione della catena logistica: analisi predittiva e previsione della domanda
5. Come (non) utilizzare l'apprendimento automatico per le previsioni sulle serie storiche: evitare le insidie
6. Come utilizzare l'apprendimento automatico per l'ottimizzazione della produzione:

utilizzo dei dati per migliorare le prestazioni

7. Come si insegna la fisica ai sistemi di intelligenza artificiale?

8. Possiamo costruire reti cerebrali artificiali usando magneti su scala nanometrica?

Apprendimento automatico Data Science AI IoT Verso la scienza dei dati

A proposito di AiutoLegale