Classificazione di eventi rari estremi utilizzando Autoencoder in Keras

In questo post, impareremo come implementare un codificatore automatico per la creazione di un classificatore di eventi rari. Useremo un set di dati di eventi rari del mondo reale da qui [1].

Chitta Ranjan
3 maggio · 10 minuti di lettura

sfondo

Che cos'è un evento raro estremo?

In un problema con eventi rari, abbiamo un set di dati non bilanciato. Ciò significa che abbiamo meno campioni con etichetta positiva rispetto a quelli negativi. In un tipico problema di eventi rari, i dati con etichetta positiva sono circa il 5-10% del totale. In un problema di evento raro estremo, abbiamo meno dell'1% di dati con etichetta positiva. Ad esempio, nel set di dati utilizzato qui è circa lo 0,6%.

Tali problemi di eventi rari estremi sono abbastanza comuni nel mondo reale, ad esempio rotture di fogli e guasti alle macchine nella produzione, clic o acquisti in un settore online.

Classificare questi eventi rari è piuttosto impegnativo. Recentemente, il Deep Learning è stato ampiamente utilizzato per la classificazione. Tuttavia, **il numero limitato di campioni con etichetta positiva proibisce l'applicazione Deep Learning** . Non importa quanto siano grandi i dati, l'uso di Deep Learning viene limitato dalla quantità di campioni con etichetta positiva.

Perché dovremmo ancora preoccuparci di usare il Deep Learning?

Questa è una domanda legittima. Perché non dovremmo pensare di usare un altro approccio di Machine Learning?

approccio di Machine Learning?

La risposta è soggettiva Possiamo sempre seguire un approccio di Machine Learning. Per farlo funzionare, possiamo sottocampionare da dati con etichetta negativa per avere un set di dati quasi bilanciato. Poiché abbiamo circa lo 0,6% di dati con etichetta positiva, il sottocampionamento comporterà approssimativamente un set di dati che è circa l'1% della dimensione dei dati originali. Un approccio di Machine Learning, ad esempio SVM o Random Forest, continuerà a funzionare su un set di dati di queste dimensioni. Tuttavia, avrà limiti nella sua precisione. E non utilizzeremo le informazioni presenti nel restante ~ 99% dei dati.

Se i dati sono sufficienti, i metodi di apprendimento profondo sono potenzialmente più capaci. Consente inoltre la flessibilità per il miglioramento del modello utilizzando architetture diverse. Cercheremo quindi di utilizzare i metodi di apprendimento profondo.

. . .

In questo post **impareremo come utilizzare un semplice codificatore automatico di strati densi per creare un classificatore di eventi rari** . Lo scopo di questo post è dimostrare l'implementazione di un Autoencoder per la classificazione di eventi rari estremi. Lasceremo l'esplorazione di diverse architetture e configurazioni dell'Autoencoder sull'utente. Per favore condividi nei commenti se trovi qualcosa di interessante.

Autoencoder per la classificazione

L'approccio del codificatore automatico per la classificazione è simile al **rilevamento di anomalie** . Nel rilevamento delle anomalie apprendiamo lo schema di un normale processo. Tutto ciò che non segue questo schema è classificato come un'anomalia. Per una classificazione binaria di eventi rari, possiamo usare un approccio simile usando gli autoencoder (derivati da qui [2]).

Revisione rapida: cos'è un codificatore automatico?

- Un autoencoder è composto da due moduli: encoder e decoder.
- Il codificatore apprende le caratteristiche sottostanti di un processo. Queste funzionalità sono in genere in una dimensione ridotta.
- Il decodificatore può ricreare i dati originali da queste funzioni sottostanti.

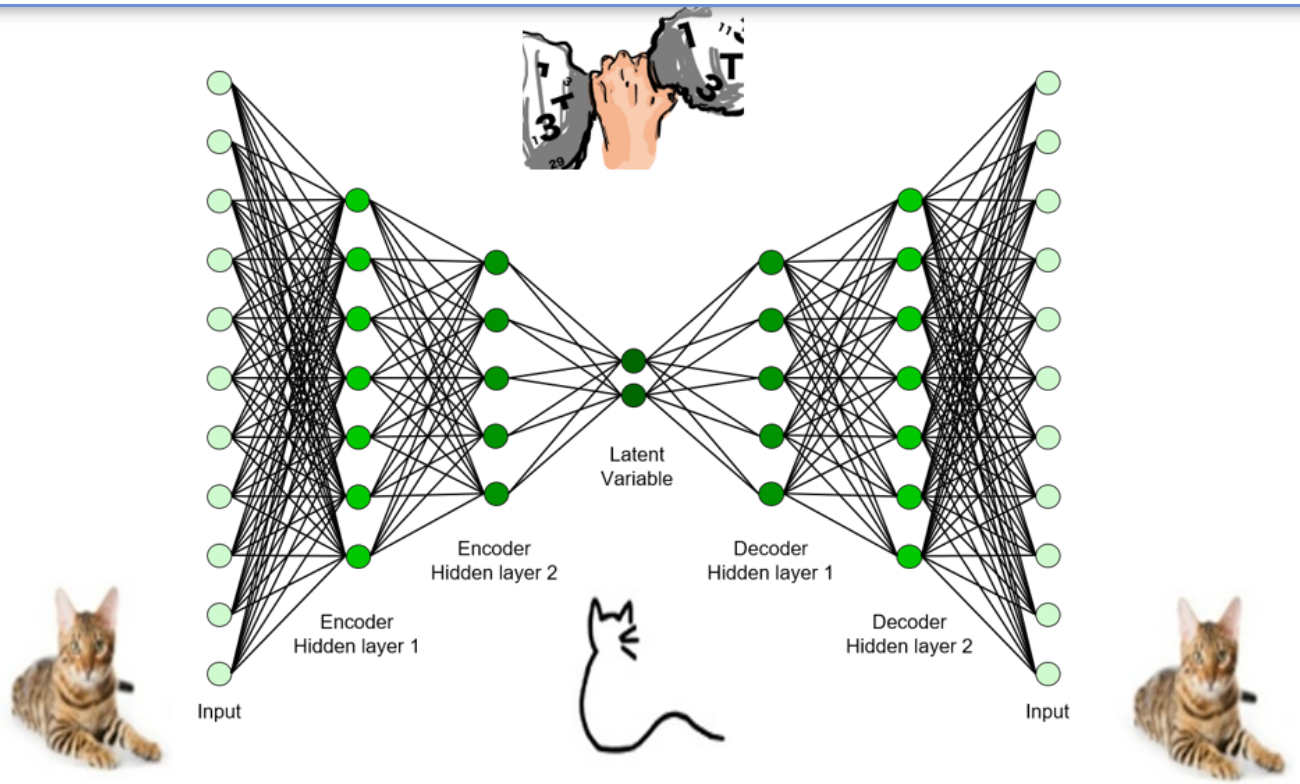


Figura 1. Illustrazione di un codificatore automatico. [Fonte : Autoencoder del Prof. Seungchul Lee iSystems Design Lab]

Come utilizzare una classificazione degli eventi rari di Autoencoder?

- Divideremo i dati in due parti: etichettati positivamente e etichettati negativamente.
- I dati con etichetta negativa vengono trattati come stato *normale* del processo. Uno stato *normale* è quando il processo è privo di eventi.
- Ignoreremo i dati con etichetta positiva e formeremo un codificatore automatico solo sui dati con etichetta negativa.
- Questo codificatore automatico ha ora appreso le caratteristiche del *normale* processo.
- Un Autoencoder ben addestrato predirà qualsiasi nuovo dato proveniente dallo stato *normale* del processo (poiché avrà lo stesso modello o distribuzione).
- Pertanto, l'errore di ricostruzione sarà piccolo.
- Tuttavia, se proviamo a ricostruire un dato da un evento raro, l'Autoencoder farà fatica.
- Ciò renderà elevato l'errore di ricostruzione durante l'evento raro.

- Siamo in grado di rilevare errori di ricostruzione così elevati ed etichettarli come previsione di eventi rari.
- Questa procedura è simile ai metodi di rilevamento delle anomalie.

Implementazione

Dati e problemi

Questo è un dato binario etichettato da una cartiera per rotture di fogli. La rottura dei fogli è un grave problema nella produzione della carta. Una singola rottura del foglio provoca la perdita di diverse migliaia di dollari e i mulini vedono almeno una o più rotture ogni giorno. Ciò causa milioni di dollari di perdite annuali e rischi per il lavoro.

Rilevare un evento di interruzione è impegnativo a causa della natura del processo. Come menzionato in [1], anche una riduzione del 5% nelle pause porterà un vantaggio significativo ai mulini.

I dati che abbiamo contengono circa 18k righe raccolte in 15 giorni. La colonna `y` contiene le etichette binarie, con 1 che indica un'interruzione del foglio. Le colonne rimanenti sono predittori. Esistono circa 124 campioni con etichetta positiva (~ 0,6%).

Scarica qui i dati .

Codice

Importa le librerie desiderate.

```
% matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
import numpy as np
from pylab import rcParams

import tensorflow as tf
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
da regularizers keras importazione

da sklearn.preprocessing importazione StandardScaler
da sklearn.model selection importazione train test split
```

```
da sklearn.metrics importare confusion_matrix,
precision_recall_curve
da sklearn.metrics importare recall_score, classification_report,
AUC, roc_curve
da sklearn.metrics importare precision_recall_fscore_support,
f1_score

da numpy.random importa seme
seed (1)
da tensorflow import set_random_seed
set_random_seed (2)

SEED = 123 #utilizzato per aiutare a selezionare casualmente i punti
dati
DATA_SPLIT_PCT = 0.2

rcParams ['figure.figsize'] = 8, 6
LABELS = ["Normale ","Rompere"]
```

Si noti che stiamo impostando i semi casuali per la riproducibilità del risultato.

Preelaborazione dei dati

Ora leggiamo e prepariamo i dati.

```
df = pd.read_csv ("data / processminer-rare-event-mts - data.csv")
```

L'obiettivo di questo problema di eventi rari è prevedere una rottura del foglio prima che si verifichi. Cercheremo di prevedere la pausa con 4 minuti di anticipo. Per costruire questo modello, sposteremo le etichette di 2 righe in alto (che corrisponde a 4 minuti). Possiamo farlo come `df.y=df.y.shift(-2)` . Tuttavia, in questo problema vorremmo fare lo spostamento come: se la riga *n* è etichettata positivamente,

- Rendi la riga (*n* -2) e (*n* -1) uguale a 1. Ciò consentirà al classificatore di apprendere **fino a** 4 minuti di previsione in anticipo.
- Elimina riga *n* . Perché non vogliamo che il classificatore impari a prevedere una pausa quando si è verificato.

Svilupperemo il seguente UDF per questo spostamento di curva.

```
sign = lambda x: (1, -1) [x <0]
```

```
def curve_shift (df, shift_by):
    ''' '
    Questa funzione sposta le etichette binarie in un frame di dati.
    Lo spostamento della curva sarà rispetto a 1s.
    Ad esempio, se shift è -2,
    si verificherà il seguente processo : se la riga n è etichettata
come 1, allora
    - Crea riga (n + shift_by) :( n + shift_by-1) = 1.
    - Rimuovi riga n.
    cioè le etichette verranno spostate fino a 2 righe in alto.
Input:     df Un frame di dati panda con una colonna con etichetta
binaria.              Questa colonna etichettata dovrebbe essere
denominata 'y'.     shift_by Un numero intero che indica il numero
di righe da spostare.     Produzione
```

```
    df Un frame di dati con le etichette binarie spostate di shift.
    ''' '

     vector = df [' y ']. copy ()
    per s nel range (abs (shift_by)):
        tmp = vector.shift (segno (shift_by))
        tmp = tmp.fillna (0)
        vector + = tmp
    labelcol = 'y'
    # Aggiungi vettore a df
     df.insert (loc = 0, colonna = labelcol + 'tmp', valore =
vettore)
    # Rimuovi le righe con labelcol == 1.
     df = df.drop (df [df [labelcol ] == 1] .index)
    # Rilascia labelcol e rinomina il tmp col come labelcol
     df = df.drop (labelcol, axis = 1)
    df = df.rename (colonne = {labelcol + 'tmp': labelcol})
    # Crea il etichetta binaria
    df.loc [df [labelcol]> 0, labelcol] = 1

    restituisce df
```

Prima di andare avanti, lasceremo cadere il tempo, e anche le colonne categoriche per semplicità.

```
# Rimuovi la colonna del tempo e le colonne categoriali
 df = df.drop (['time', 'x28', 'x61'], axis = 1)
```

Ora dividiamo i dati in set di treni, validi e test. Quindi prenderemo il sottoinsieme di dati con solo 0s per addestrare il codificatore automatico.

```
df_train, df_test = train_test_split (df, test_size =
DATA_SPLIT_PCT, random_state = SEMI)
df_train, df_valid = train_test_split (df_train, test_size =
DATA_SPLIT_PCT, random_state = SEMI)

df_train_0 = df_train.loc [df [ 'y'] == 0]
df_train_1 = df_train.loc [df ['y'] == 1]
df_train_0_x = df_train_0.drop (['y'], axis = 1)
df_train_1_x = df_train_1.drop (['y'], axis = 1)

df_valid_0 = df_valid. loc [df ['y'] == 0]
df_valid_1 = df_valid.loc [df ['y'] == 1]
df_valid_0_x = df_valid_0.drop (['y'], axis = 1)
df_valid_1_x = df_valid_1.drop ( ['y'], axis = 1)

df_test_0 = df_test.loc [df ['y'] == 0]
df_test_1 = df_test.loc [df ['y'] == 1]
df_test_0_x = df_test_0.drop ([' y '],axis = 1)
df_test_1_x = df_test_1.drop (['y'], axis = 1)
```

Standardizzazione

Di solito è meglio usare un dato standardizzato (trasformato in gaussiano, media 0 e varianza 1) per i codificatori automatici.

```
ablatore = StandardScaler (). misura (df_train_0_x)
df_train_0_x_rescaled = scaler.transform (df_train_0_x)
df_valid_0_x_rescaled = scaler.transform (df_valid_0_x)
df_valid_x_rescaled = scaler.transform (df_valid.drop ([ 'y'], asse
= 1))

df_test_0_x_rescaled = ablatore .transform (df_test_0_x)
df_test_x_rescaled = scaler.transform (df_test.drop (['y'], axis =
1))
```

Classificatore di autoencoder

Inizializzazione

Innanzitutto, inizializzeremo l'architettura del codificatore automatico. Stiamo costruendo un semplice codificatore automatico. Architetture più complesse e altre configurazioni dovrebbero essere esplorate.

```
nb_epoch = 200
```

```
batch_size = 128
input_dim = df_train_0_x_rescaled.shape [1] #numero di variabili
predittive,
 encoding_dim = 32
hidden_dim = int (encoding_dim / 2)
learning_rate = 1e-3

input_layer = Input (shape = (input_dim,)
encoder (encoding_dim, activation = "relu", activity_regularizer =
regularizers.l1 (learning_rate)) (input_layer)
encoder = Dense (hidden_dim, activation = "relu") (encoder)
decoder = Dense (hidden_dim, activation = "relu") (encoder )
decodificatore = denso (encoding_dim, attivazione = "relu")
(decodificatore)
decodificatore = denso (input_dim, attivazione = "lineare")
(decodificatore)
autoencoder = modello (input = input_layer, output = decoder)
autoencoder.summary ()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 59)                0
_________________________________________________________________
dense_23 (Dense)             (None, 32)                1920
_________________________________________________________________
dense_24 (Dense)             (None, 16)                528
_________________________________________________________________
dense_25 (Dense)             (None, 16)                272
_________________________________________________________________
dense_26 (Dense)             (None, 32)                544
_________________________________________________________________
dense_27 (Dense)             (None, 59)                1947
=================================================================
Total params: 5,211
Trainable params: 5,211
Non-trainable params: 0
_________________________________________________________________
```

Formazione

Formeremo il modello e lo salveremo in un file. Salvare un modello addestrato è una buona pratica per risparmiare tempo per analisi future.

```
autoencoder.compile (metrics = ['accuratezza'],
                     loss = 'mean_squared_error',
                     optimizer = 'adam')

cp = ModelCheckpoint (filepath = "autoencoder_classifier.h5",
                                 save_best_only = True,
                                 verbose = 0)

tb = TensorBoard (log_dir = './logs',
                  histogram_freq = 0,
                  write_graph = True,
                  write_images = True)
```

```
history = autoencoder.fit (df_train_0_x_rescaled,
df_train_0_x_rescaled,
                    epochs = nb_epoch,
                    batch_size = batch_size,
                    shuffle = True,
                    validation_data = (df_valid_0_x_rescaled,
df_valid_0_x_rescaled),
                    verbose = 1,
                    callbacks = [cp, tb]).
```

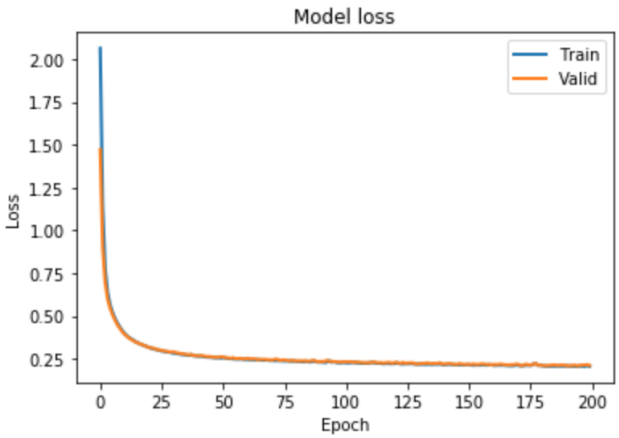


Figura 2. Perdita per la formazione del codificatore automatico.

Classificazione

Di seguito, mostriamo come possiamo usare un errore di ricostruzione del codificatore automatico per la classificazione degli eventi rari.

Come accennato in precedenza, se l'errore di ricostruzione è elevato, lo classificheremo come una rottura del foglio. Dovremo determinare la soglia per questo.

Useremo il set di validazione per identificare la soglia.

```
valid_x_predictions = autoencoder.predict (df_valid_x_rescaled)
mse = np.mean (np.power (df_valid_x_rescaled - valid_x_predictions,
2), asse = 1)
error_df = pd.DataFrame ({ 'Reconstruction_error': MSE,
                           'True_class': df_valid [ 'y ']})

precision_rt, remind_rt ,reshold_rt = precision_recall_curve
(error_df.True_class, error_df.Reconstruction_error)
plt.plot (reshold_rt, precision_rt [1:], label = "Precision",
linewidth = 5)
plt.plot (reshold_rt, richiamo_ 1:], label = "Recall", linewidth =
5)
plt.title ('Precisione e richiamo per valori soglia diversi')
```

```
plt.xlabel ('Soglia')

plt.ylabel ('Precisione / Richiamo')
plt.legend ( )
plt.show ()
```

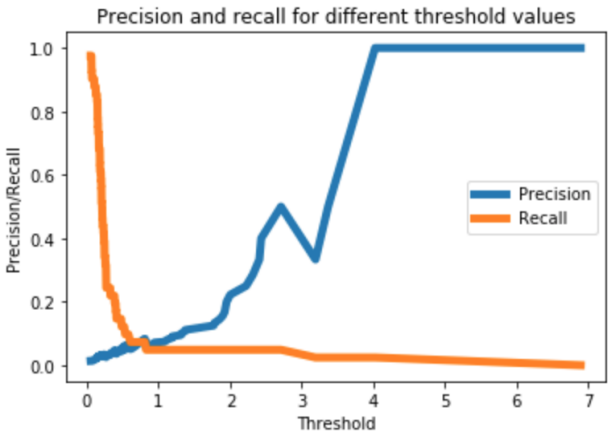


Figura 3. Una soglia di 0,4 dovrebbe fornire un ragionevole compromesso tra precisione e richiamo.

Ora eseguiremo la classificazione sui dati del test.

> ***Non dovremmo stimare la soglia di classificazione dai dati del test. Ciò comporterà un eccesso di adattamento.***

```
test_x_predictions = autoencoder.predict (df_test_x_rescaled)
mse = np.mean (np.power (df_test_x_rescaled - test_x_predictions,
2), asse = 1)
error_df_test = pd.DataFrame ({ 'Reconstruction_error': MSE,
                        'True_class': df_test [ 'y ']})
error_df_test = error_df_test.reset_index ()reshold_fixed

= 0.4
gruppi = error_df_test.groupby (' True_class ')

fig, ax = plt.subplots ()

per nome, gruppo in gruppi:
    ax.plot (group.index, group. Reconstruction_error, marker = 'o',
ms = 3.5, linestyle = '',
            label = "Break" se name == 1 else "Normal")
ax.hlines (soglia_fissa, ax.get_xlim () [0], ax.get_xlim () [1],
colors = "r", zorder = 100, label = 'Soglia ')
ax.legend ()
plt.title ("Errore di ricostruzione per classi diverse")
plt.ylabel ("Errore di ricostruzione")
plt.xlabel ("Indice punti dati")
plt.show ();
```

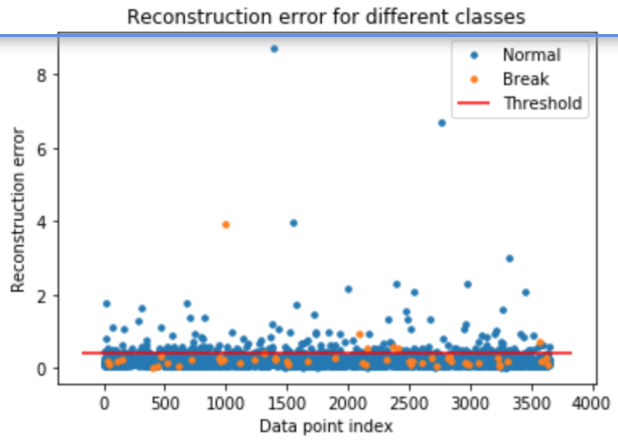


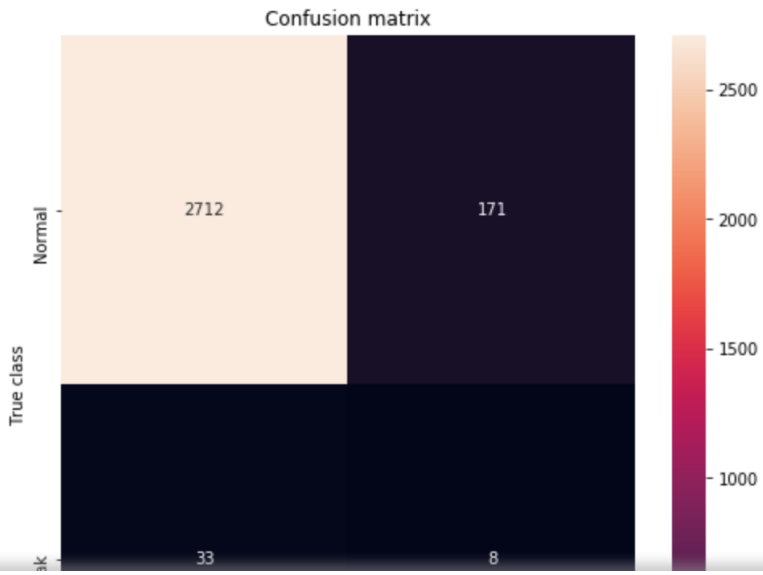
Figura 4. Utilizzo della soglia = 0,4 per la classificazione. I punti arancione e blu sopra la linea di soglia rappresentano rispettivamente il vero positivo e il falso positivo.

Nella Figura 4, il punto arancione e blu sopra la linea di soglia rappresenta rispettivamente il vero positivo e il falso positivo. Come possiamo vedere, abbiamo un buon numero di falsi positivi. Per avere un aspetto migliore, possiamo vedere una matrice di confusione.

```
pred_y = [1 if e> soglia_fissata altro 0 per e in
error_df.Reconstruction_error.values]

conf_matrix = confusion_matrix (error_df.True_class, pred_y)

plt.figure (figsize = (12, 12))
sns.heatmap (conf_matrix, xticklabels = LAB , yticklabels = LABELS,
annot = True, fmt = "d");
plt.title ("Matrice di confusione")
plt.ylabel ('True class')
plt.xlabel ('Classe prevista')
plt.show ()
```



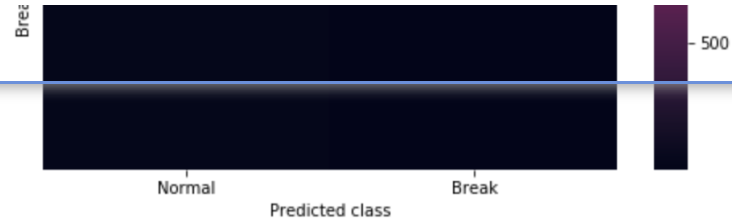


Figura 5. Matrice di confusione sulle previsioni del test.

Potremmo prevedere 8 casi di interruzione su 41. Si noti che queste istanze includono previsioni anticipate di 2 o 4 minuti. Si tratta di circa il 20%, che rappresenta un buon tasso di richiamo per l'industria cartaria. Il tasso di falsi positivi è di circa il 6%. Questo non è l'ideale ma non è terribile per un mulino.

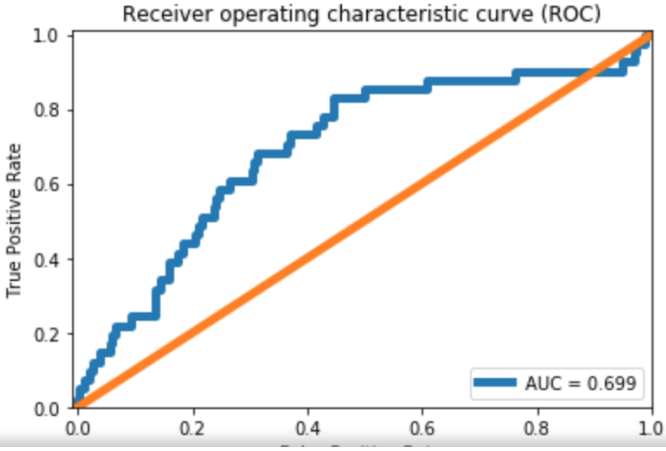
Tuttavia, questo modello può essere ulteriormente migliorato per aumentare la frequenza di richiamo con una frequenza falsa positiva più piccola. Esamineremo l'AUC di seguito e poi parleremo del prossimo approccio per il miglioramento.

Curva ROC e AUC

```
false_pos_rate, true_pos_rate, soglie = roc_curve
(error_df.True_class, error_df.Reconstruction_error)
roc_auc = AUC (false_pos_rate, true_pos_rate,)

plt.plot (false_pos_rate, true_pos_rate, linewidth = 5, label = 'AUC
=% 0.3f' % roc_auc)
plt .plot ([0,1], [0,1], linewidth = 5)

plt.xlim ([- 0.01, 1])
plt.ylim ([0, 1.01])
plt.legend (loc = 'in basso a destra' )
plt.title ("Curva caratteristica operativa del ricevitore (ROC)")
plt.ylabel ("Tasso positivo reale")
plt.xlabel ("Tasso positivo falso")
plt.show ()
```

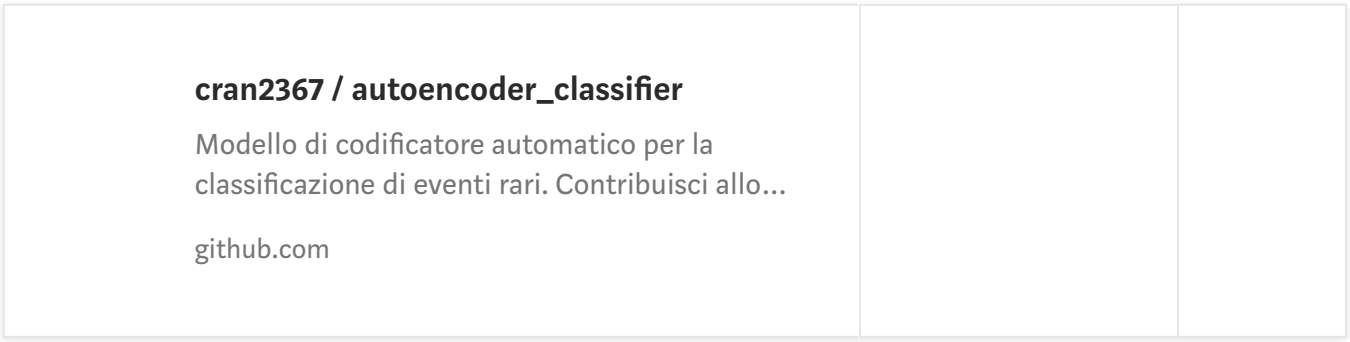


False Positive Rate

L'AUC è 0,69.

Repository Github

L'intero codice con commenti è presente qui .



Cosa si può fare di meglio qui?

Ottimizzazione del codificatore automatico

Gli autoencoder sono un'estensione non lineare di PCA. Tuttavia, il codificatore automatico convenzionale sviluppato sopra non segue i principi della PCA. In Build the Right Autoencoder - Ottimizza e ottimizza utilizzando i principi PCA. Parte I e Parte II , i principi PCA richiesti che dovrebbero essere incorporati in un Autoencoder per l'ottimizzazione sono spiegati e implementati.

Auto-codificatore LSTM

Il problema discusso qui è una serie temporale (multivariata). Tuttavia, nel modello Autoencoder non stiamo prendendo in considerazione le informazioni / i modelli temporali. Nel prossimo post , esploreremo se è possibile con un RNN. Proveremo un autoencoder LSTM .

Conclusione

Abbiamo lavorato su un evento binario estremamente raro etichettato con dati binari da una cartiera per costruire un classificatore di autoencoder. Abbiamo raggiunto una precisione ragionevole. Lo scopo qui era di dimostrare l'uso di un Autoencoder di base per la classificazione di eventi rari. Lavoreremo ulteriormente sullo sviluppo di altri metodi, incluso un codificatore automatico LSTM in grado di estrarre le funzionalità temporali per una migliore precisione.

Il prossimo post su LSTM Autoencoder è qui, LSTM Autoencoder per la classificazione

di eventi rari .

. . .

Letture di follow-up consigliate

- Costruisci il giusto Autoencoder: ottimizza e ottimizza utilizzando i principi PCA. Parte I .
- Costruisci il giusto Autoencoder: ottimizza e ottimizza utilizzando i principi PCA. Parte II .
- Auto-codificatore LSTM per la classificazione di eventi rari estremi in telecamere .

Riferimenti

1. Ranjan, C., Mustonen, M., Paynabar, K., & Pourak, K. (2018). Set di dati: classificazione degli eventi rari nelle serie temporali multivariate. *arXiv prestampa arXiv: 1809.10717* .
2. https://www.datascience.com/blog/fraud-detection-with-tensorflow
3. Repository Github: https://github.com/cran2367/autoencoder_classifier

Dichiarazione di non responsabilità: lo scopo di questo post è limitato a un tutorial per la creazione di un codificatore automatico di livello denso e per l'utilizzo come classificatore di eventi rari. Ci si aspetta che un professionista ottenga risultati migliori per questi dati mediante la messa a punto della rete. Lo scopo di questo articolo è aiutare i data scientist a implementare un codificatore automatico.