



Multivariate and Statistical Learning: Anomaly detection utilizzando gli Autoencoder per la riduzione dimensionale non lineare

Gruppo: Anomaly detectives

Federico Fiorentino

Matteo Gemignani-6284577

Matteo Ghera-7006227

Matteo Marulli-7005652

Corso di laurea magistrale in Informatica 

Curriculum: Data Science

Università degli studi di Firenze





Outline

- **Anomaly detection**

- Anomaly detection: Teoria generale
- Anomaly detection: Riduzione dimensionale
- Anomaly detection: PCA

- **Neural Network: Autoencoder**

- Che cosa sono gli autoencoder
- Autoencoder sparso e Denoising Autoencoder
- Implementazione con Tensorflow 2

- **Fase Sperimentale**

- Sistema di Lorenz
- Generazione dei dati artificiali
- Dataset credit_card
- Risultati

Dataset artificiale

Dataset: Credit card

- **Conclusioni**

- **Riferimenti**



Anomaly detection: aspetti teorici I

Anomaly detection attraverso la riduzione dimensionale

Nel **data mining**, per **anomaly detection** intendiamo un processo che identifica gli elementi o gli eventi che mostrano un comportamento anomalo rispetto all'insieme dei dati osservato.

Applicazioni

Nel campo della **sicurezza informatica**, l'anomaly detection permette all'**analista di cybersecurity** (o ai **data scientist** che hanno come dominio applicativo la cybersecurity) di eseguire diversi task su **database** oppure **data warehouse** come:

- **Fraud detection**
- **Intrusion detection**
- **Fault detection**
- **Malware detection**
- **Network traffic analysis**
- **Monitoring di sistemi informatici**



Gli outlier

Chiamiamo **outlier** (o anomalie) gli esempi che esibiscono un comportamento insolito rispetto alla maggior parte degli esempi presenti nel dataset. Gli outlier si dividono in 3 gruppi:

- In **anomalie puntuale** quando un singolo esempio è estremamente diverso e distante dal resto del dataset.
- In **anomalie contestuali** quando un singolo esempio è considerata un'anomalia solo in specifici contesti.
- In **anomalie collettive** quando un gruppo di esempi assume un comportamento anomalo rispetto al resto del dataset ma non individualmente.

Anomaly detection: aspetti teorici III

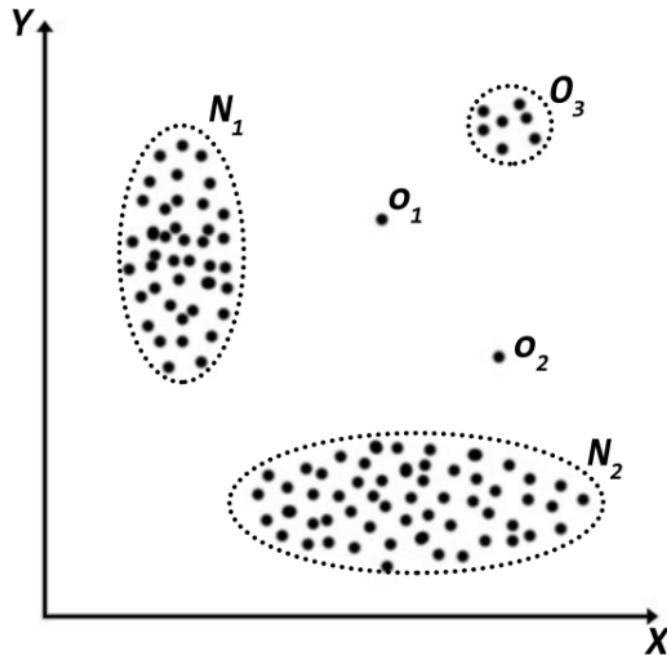


Figure: Esempio di anomalie puntuali

Anomaly detection: aspetti teorici IV

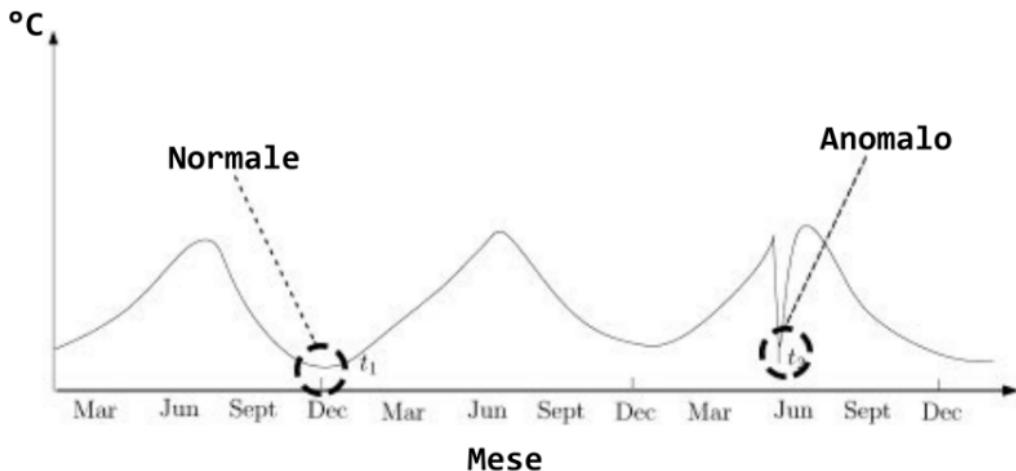


Figure: Esempio di anomalia contestuale

Anomaly detection: aspetti teorici V

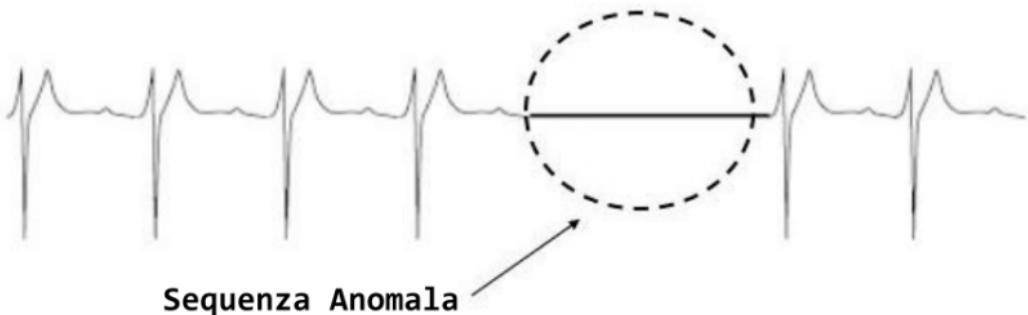


Figure: Esempio di anomalia collettive



Perchè si manifestano gli outlier?

- **Esempi da classi differenti:** un esempio può risultare differente poiché appartenente a una diversa classe.
- **Variazioni naturali:** molti fenomeni possono essere modellati con distribuzioni probabilistiche in cui esiste, anche se molto ridotta, la probabilità che si verifichi un fenomeno con caratteristiche molto differenti dagli altri.
- **Errori di misurazione:** dovuti a errori umani o di sensori

Anomaly detection: aspetti teorici VII

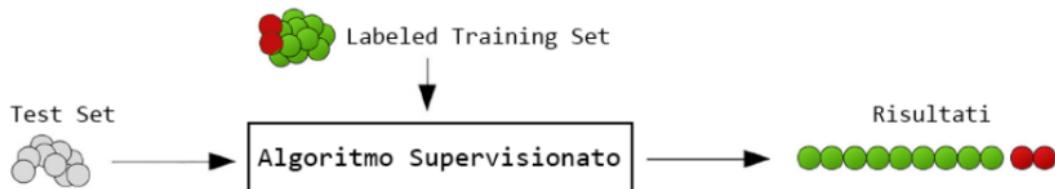


Figure: Anomaly detection supervisionata

- si tratta il problema di anomaly detection con un problema di **classificazioni binaria**.
- possibilità di usare molti algoritmi già visti (es: SVM, regressione logistica, Naive Bayes ecc...) e i **metodi di ensemble** (Adaboost, Random Forest, ecc...)
- Le due **classi** (cioè normale vs anomala) sono tipicamente **sbilanciate**.
- **identificare** il maggior numero di anomalie è molto **più importante** che classificare erroneamente un comportamento normale come anomalia.

Anomaly detection: aspetti teorici VIII

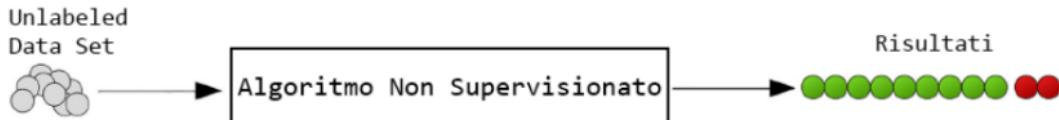


Figure: Anomaly detection non supervisionata

- sono la famiglia di algoritmi di learning più usata per il task di anomaly detection in quanto per definizione **le anomalie sono imprevedibili** e difficilmente si ha un dataset etichettato per lo scopo.
- assumono che gli esempi normali seguono un **pattern ben preciso** rispetto alle anomalie e se tale assunzione non è soddisfatta, l'algoritmo non supervisionato tende a generare un gran numero di **falsi negativi e falsi positivi**.

Anomaly detection: Riduzione dimensionale I



- L'Anomaly detection ottiene il modello acquisendo il comportamento normale dei dati durante il periodo di training.
- Successivamente controlla se i dati del testset possono essere adattati o meno al modello addestrato.
- Se i dati del testset non sono coerenti con il modello ottenuto dal trainingset, vengono considerati anomali.
- Il rilevamento dell'anomalia mediante la riduzione della dimensionalità, si basa sul presupposto che i **dati abbiano variabili correlate** tra loro e possano essere incorporati in un sottospazio dimensionale inferiore, in cui **campioni normali e campioni anomali appaiono significativamente diversi**.
- Nella fase di training, si suppone che $\{\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(m)\}$ siano dati normali del training set e che ciascun campione $\mathbf{x}(i) \in \mathbb{R}^D$ sia rappresentato da un vettore di D variabili differenti.

Anomaly detection: Riduzione dimensionale II



- Comprimendo i dati in sottospazio latente con dimensione inferiore si ottiene l'output $\{\hat{x}(1), \hat{x}(2), \dots, \hat{x}(m)\}$ che **minimizza l'errore di ricostruzione** così definito:

$$MSE(i) = \frac{1}{n} \sum_{j=1}^D (x_j(i) - \hat{x}_j(i))^2$$

- L'errore di ricostruzione precedente è usato come *anomaly score*. Si avranno valori bassi se i campioni del test set sono istanze normali, cioè soddisfano la normale correlazione; l'errore diventa grande con campioni anomali.

Anomaly detection: PCA I



- Esistono diverse tecniche rappresentative di riduzione della dimensionalità come: PCA (lineare) e PCA kernel (non lineare).
- La Kernel PCA esegue la mappatura non lineare mediante una funzione di kernel. La funzione kernel utilizzata è la seguente:

$$\text{Gaussian Kernel } k(\mathbf{x}(i), \mathbf{x}(j)) = \exp\left(-\frac{\|\mathbf{x}(i) - \mathbf{x}(j)\|^2}{\sigma^2}\right).$$

- La kernel PCA richiede un alto costo dal punto di vista computazionale. Pertanto, rispetto al kernel PCA, gli autoencoders hanno un costo di calcolo minore.



- Nella pratica conviene usare algoritmi non supervisionati per l'anomaly detection, una tecnica interessante che fa uso delle reti neurali è l'**Autoencoder**

Autoencoder

Un Autoencoder è una rete neurale capace di apprendere una rappresentazione dell'input chiamata **codings** in modo non supervisionato, esso è composto da:

- un livello di codifica $\phi : \chi \rightarrow \Omega$
 - un livello di decodifica $\psi : \Omega \rightarrow \chi$
-
- Indichiamo con $\{\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(m)\}$ il vettore in input e con $\{\hat{\mathbf{x}}(1), \hat{\mathbf{x}}(2), \dots, \hat{\mathbf{x}}(m)\}$ il vettore in output

Autoencoder II

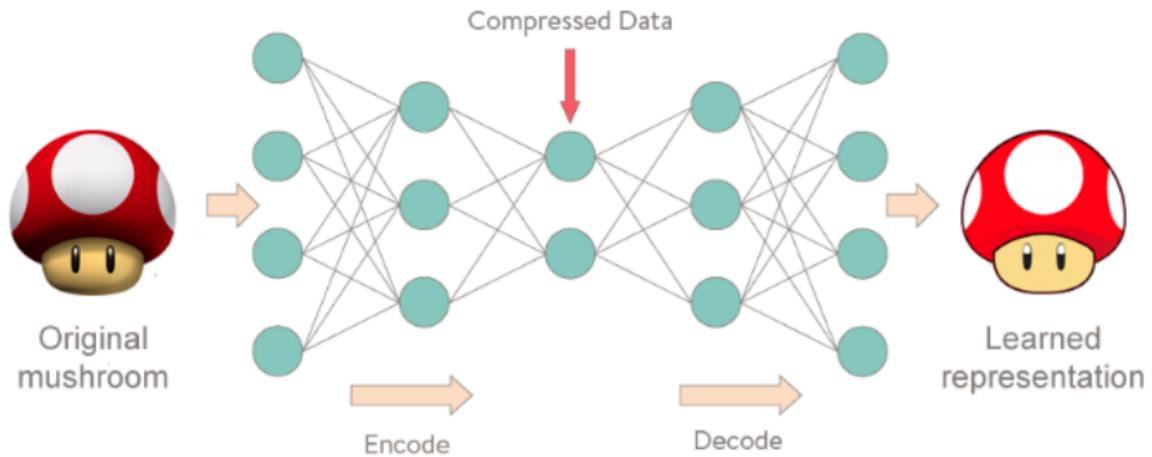


Figure: Struttura schematica di un autoencoder



La loss function

Se l'autoencoder viene usato per eseguire una **riduzione dimensionale** del dataset, come viene fatto con la PCA, la loss function è il **mean square error**:

$$J(W, b) = \frac{1}{n} \sum_{i=1}^n (x(i) - \hat{x}(i))^2$$

Autoencoder sparso

Un autoencoder sparso è un autoencoder **regolarizzato**, cioè la funzione obiettivo da ottimizzare contiene un parametro di regolarizzazione che deve essere minimizzato.

La funzione obiettivo è:

$$J(W, b) = \frac{1}{n} \sum_{i=1}^n (x(i) - \hat{x}(i))^2 + L(\lambda)$$



- Un metodo di ottimizzazione è una procedura che consente di minimizzare una funzione $f(x)$:

$$\min_x f(x)$$

- La discesa stocastica del gradiente è un metodo iterativo per l'ottimizzazione di funzioni differenziabili e il metodo *Adam* è una sua variante
- Adam è un'estensione di RMSprop che tiene conto della media mobile dei momenti primo (m) e secondo (v) del gradiente, usando gli stimatori corretti \hat{m} e \hat{v} per contrastare il bias nelle prime iterazioni.
- Il Momentum serve per avvicinarsi al minimo globale mentre RMSprop risolve il problema del decremento eccessivo del valore del tasso di apprendimento. Se quest'ultimo è troppo piccolo aumentano il numero di passi di convergenza, per questo si utilizza Adam.



Adam

Siano w_t i parametri all'iterazione t e Q la funzione costo da ottimizzare si ha

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla Q(w_t) (\Rightarrow \text{Momentum})$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2) (\nabla Q(w_t))^2 (\Rightarrow \text{RMSprop})$$

$$\hat{m} = \frac{m_{t+1}}{1 - \beta_1^{t+1}}$$

$$\hat{v} = \frac{v_{t+1}}{1 - \beta_2^{t+1}}$$

$$w_{t+1} = w_t - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

dove ϵ è un termine di smoothing aggiunto ai fini della stabilità numerica, mentre β_1 β_2 sono iperparametri. Di solito si utilizza $\epsilon = 10^{-8}$, $\beta_1 = 0.9$ e $\beta_2 = 0.999$

Neural Network: Autoencoder

Autoencoder VI

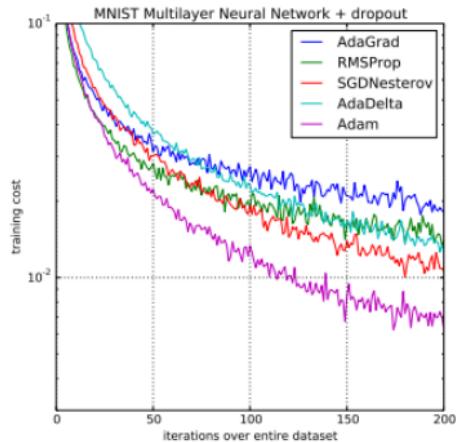


Figure: Confronto fra le prestazioni del metodo Adam con quelle degli altri metodi

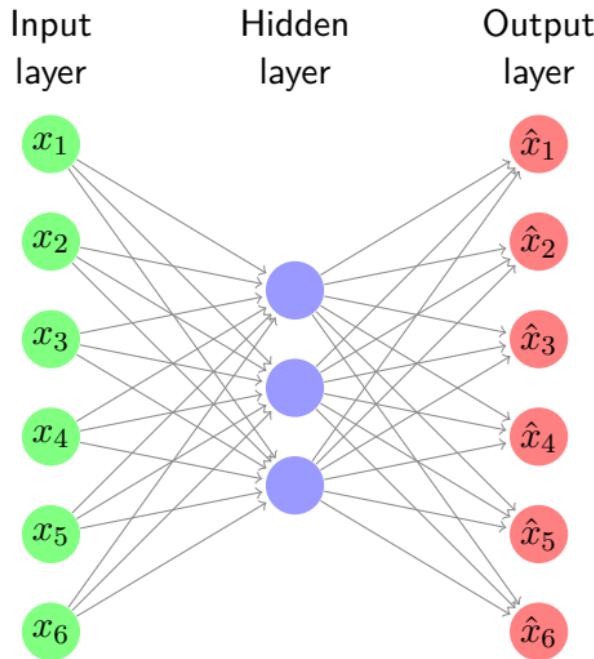
Riduzione dimensionale tramite autoencoder I



- Un autoencoder è composto da più livelli intermedi chiamati *Layer*
- Un autoencoder *undercomplete* è composto da un solo layer che apprende le caratteristiche dell'input, chiamato *Hidden Layer*, e un layer di output, chiamato *Output Layer*
- La riduzione delle dimensioni è effettuata con un autoencoder di tipo undercomplete



Riduzione dimensionale tramite autoencoder II



- L'obiettivo principale è imparare a riprodurre i vettori di input !

Riduzione dimensionale tramite autoencoder III



- Il contributo dell'unità i del layer l sul risultato è dato da:

$$a_i^{(l)} = f \left(\sum_{j=1}^n W_{i,j}^{(l-1)} a_j^{(l-1)} + b_i^{(l-1)} \right)$$

dove \mathbf{W} è la matrice dei pesi calcolata con il metodo Adam e $\mathbf{b} = \mathbf{1}$. Nel primo layer (quello di input) $\mathbf{a}^{(1)} = \mathbf{x}$ mentre nell'ultimo layer (quello di output) $\mathbf{a}^{(1)} = \hat{\mathbf{x}}$.

- La funzione f è chiamata funzione di attivazione. La funzione f da noi utilizzata è detta *ReLU, rectified linear unit*, ed è definita da $f(x) = x^+ = \max(0, x)$
- La funzione $L(\lambda)$ con il parametro di regolarizzazione è definita nel seguente modo:

$$L(\lambda) = \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l-1} \left(W_{ij}^{(l)} \right)^2$$

dove n_l denota il numero di layers nella rete e s_l il numero di unità nell'hidden layer.



Un denoising autoencoder è un'estensione di un autoencoder che consente di catturare con più precisione le strutture e i modelli inerenti ai dati in input. Otteniamo così una rappresentazione compressa e molto dettagliata dell'input con un ristretto numero di neuroni

Denoising autoencoder

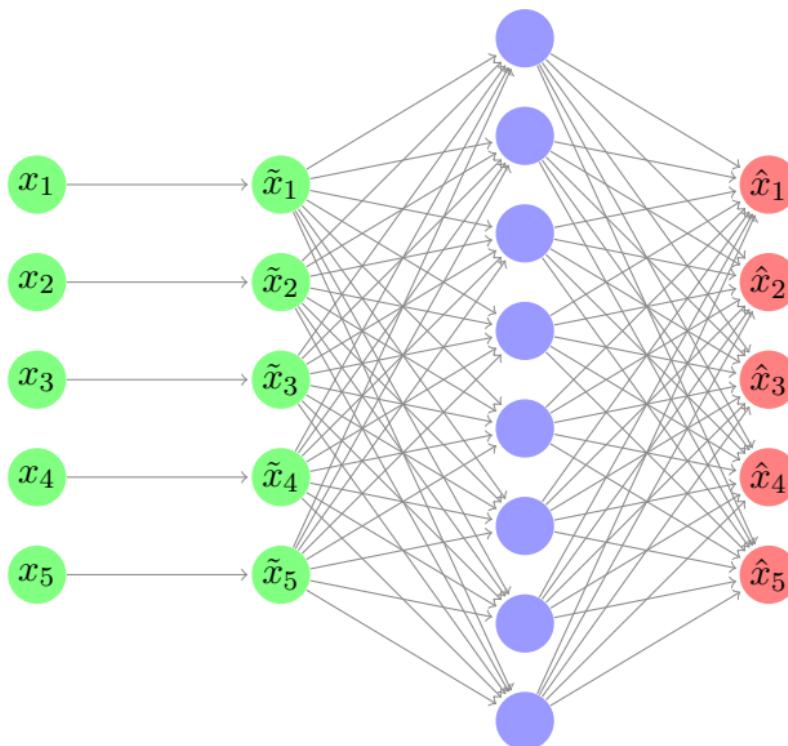
Un denoising autoencoder è un autoencoder con una dimensione dell'*hidden layer* maggiore rispetto a quella dell'input, inoltre i valori in input sono perturbati con una funzione di perturbazione: un buon denoising autoencoder saprà riconoscere le anomalie anche in presenza di rumore.

- La funzione di perturbazione da noi utilizzata è una Gaussiana



Riduzione dimensionale tramite autoencoder V

Input layer Input bias layer Hidden layer Output layer





```
import tensorflow as tf

from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping

import numpy as np

class AutoEncoder:

    def __init__(self, num_features, num_latent_node,
                 activation_fun = 'relu', lamda = 0, min_delta = 0.0004, patience = 5):
        encoder = Dense(units = num_latent_node,
                         input_shape=(num_features,),
                         activation = activation_fun,
                         kernel_regularizer = tf.keras.regularizers.l2(lamda))

        decoder = Dense(units = num_features)

        self.autoEncoder = Sequential([encoder, decoder])

        self.early = EarlyStopping(monitor='mean_squared_error', min_delta = min_delta,
                                  patience = patience)

    def model_settings(self, loss_function='mse', opt_methods='adam'):
```



```
self.autoEncoder.compile(opt_methods, loss = loss_function, metrics = ['mse'])

def setting_train_test_DS(self, X_train, X_test):
    self.X_train = X_train
    self.X_test = X_test

def fit(self, batch_size = 32, epochs = 30, shuffle = True):
    return self.autoEncoder.fit(self.X_train, self.X_train,
        batch_size=batch_size, epochs=epochs, callbacks = [self.early],
        shuffle=shuffle, validation_data=(self.X_test, self.X_test))

def predict(self):
    self.X_test_hat = self.autoEncoder.predict(self.X_test)

def get_error_recustruction_test(self):
    return np.sqrt(np.mean((self.X_test - self.X_test_hat)**2, axis=1))
```



Il sistema di Lorenz è il primo esempio di un sistema di equazioni differenziali a bassa dimensionalità in grado di generare un comportamento caotico nel movimento termico di convezione di un fluido. Venne scoperto da Edward N. Lorenz, del Massachusetts Institute of Technology, nel 1963. Il sistema è composto dalle seguenti equazioni differenziali:

$$\begin{cases} \dot{z}_1(t) = \sigma(z_2(t) - z_1(t)) \\ \dot{z}_2(t) = z_1(t)(\rho - z_3(t)) - z_2(t) \\ \dot{z}_3(t) = z_1(t)z_2(t) - \beta z_3(t) \end{cases}$$

Il sistema dipende da tre parametri σ , ρ e β che rappresentano rispettivamente il numero di Prandtl, il numero di Rayleigh ed una costante posta a 8/3.

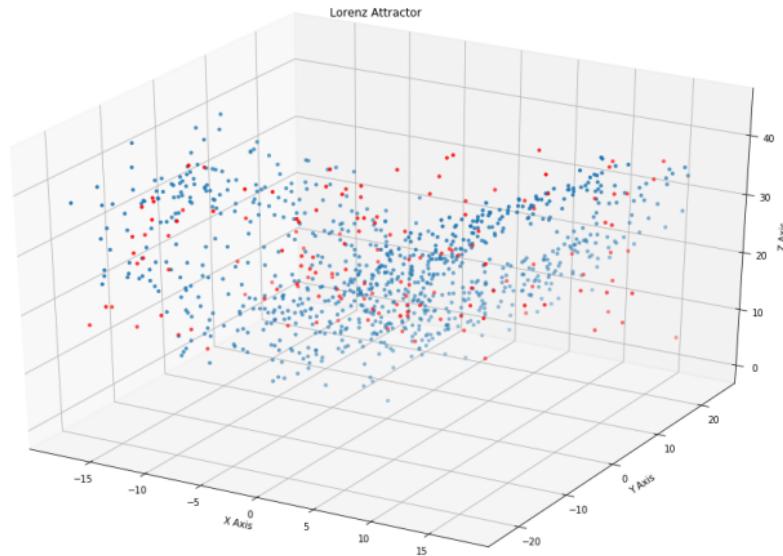


Figure: Dati generati con il sistema di Lorenz con $\sigma = 10$, $\rho = 28$ e $\beta = 8/3$

Generazione dei dati artificiali I



- Sia $\mathbf{z}(t) = (z_1(t), z_2(t), z_3(t))^T$ il punto tridimensionale generato dal sistema di Lorenz al tempo t
- Generiamo il vettore $Z = \{\mathbf{z}(1), \mathbf{z}(2), \dots, \mathbf{z}(1000)\}$ dei punti tridimensionali generati con il sistema di Lorenz (quindi $Z \in \mathbb{R}^{3 \times 1000}$)
- Per creare i dati anomali, gli autori dell'articolo modificano le componenti z_3 dei punti generati dal secondo 850 al secondo 1000 cambiando l'ordine cronologico della componente
- Sia $W \in \mathbb{R}^{25 \times 3}$ una matrice contenente numeri casuali scelti nell'intervallo (-5,5)
- Calcoliamo la matrice $X = W * Z \in \mathbb{R}^{25 \times 1000}$ che è un vettore ad alta dimensionalità
- X è la serie temporale analizzata: i primi 700 secondi costituiscono il training set mentre gli ultimi 300 secondi il test set

Generazione dei dati artificiali II

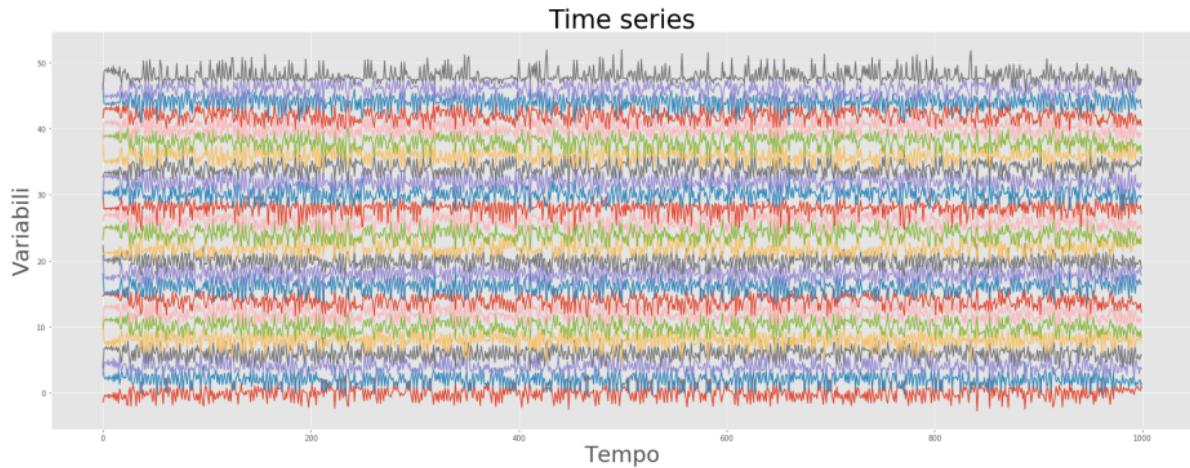
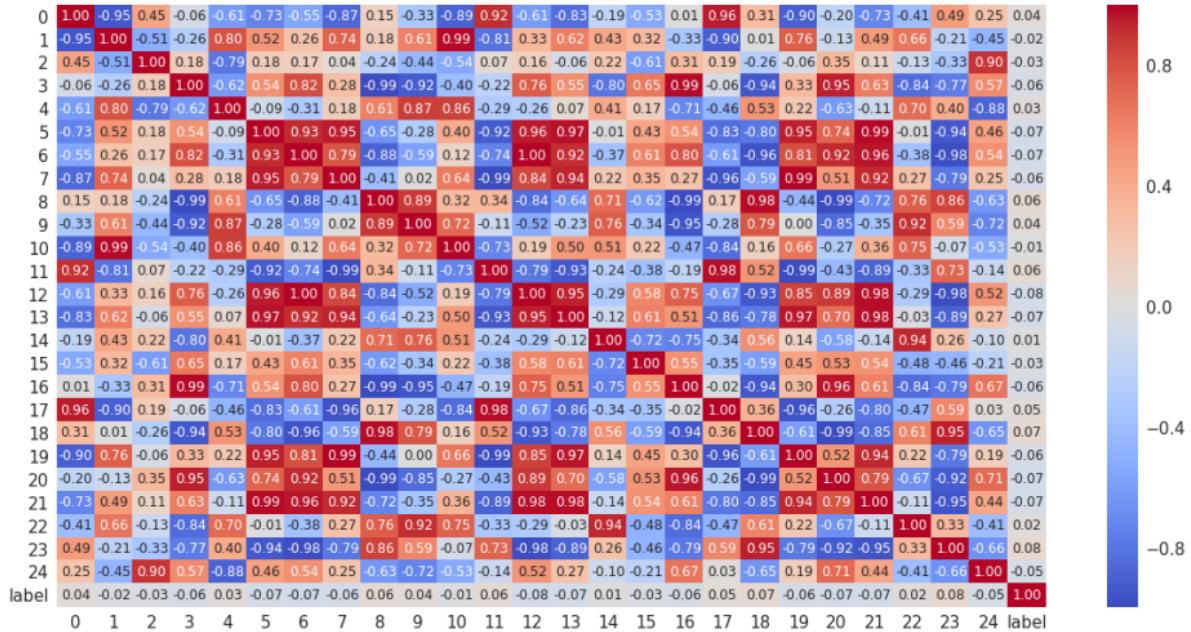


Figure: Dataset

Generazione dei dati artificiali III



Generazione dei dati artificiali IV



- Per rilevare le anomalie, riduciamo la dimensione di ciascun dato utilizzando quattro metodi:
 - ① Linear PCA
 - ② Kernel PCA
 - ③ Autoencoder
 - ④ Denoising Autoencoder
- Per ogni metodo, modifichiamo la dimensione dello spazio latente manualmente e il parametro λ della funzione obiettivo dell'autoencoder è posta uguale a 0.00001.

Dataset credit_card I



- E' uno dei dataset più famosi su Kaggle
- Tolto il tempo, la classe e l'importo, le altre variabili sono il risultato di una PCA
- Viene usato per la **fraud detection**
- Il dataset è molto grande ed è fortemente sbilanciato

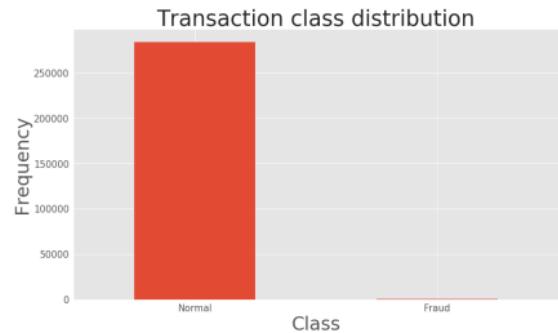
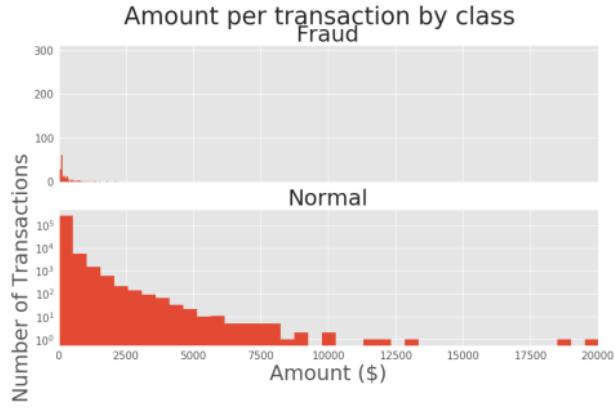
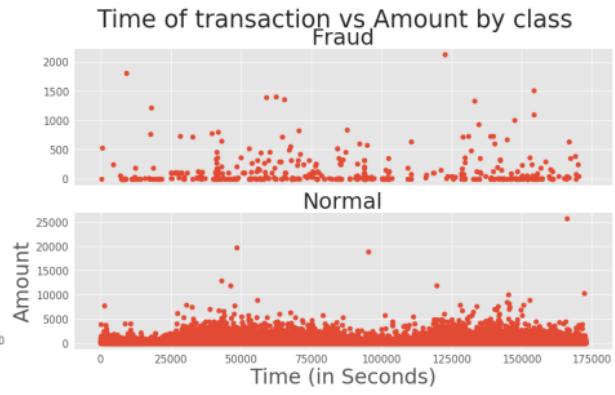


Figure: La maggior parte delle transazioni sono normali, le anomalie sono poche



(a)



(b)

Figure: Nelle transazioni normali gli importi con cifre importanti sono molto rari, invece in quelle anomale gli importi sono rari e modesti questo perchè i truffatori che hanno clonato la carte di credito non vogliono dare nell'occhio

Dataset credit_card III

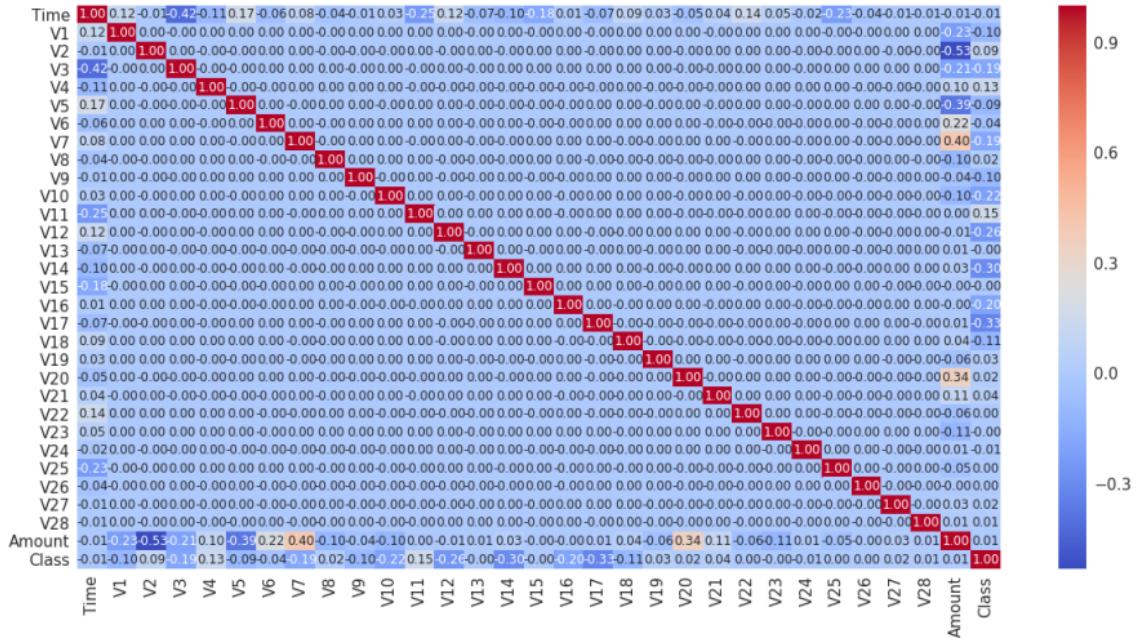


Figure: La matrice di correlazione per credit card, non sono presenti variabili altamente correlate tra di loro

Risultati dataset artificiale I

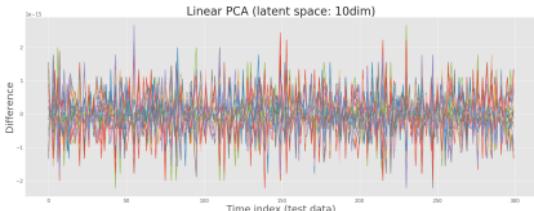


- Per rilevare le anomalie, riduciamo la dimensione di ciascun dato utilizzando quattro metodi:
 - ① Linear PCA
 - ② Kernel PCA
 - ③ Autoencoder
 - ④ Denoising Autoencoder
- Per ogni metodo, modifichiamo la dimensione dello spazio latente manualmente e il parametro λ della funzione obiettivo dell'autoencoder è posta uguale a 0.00001.

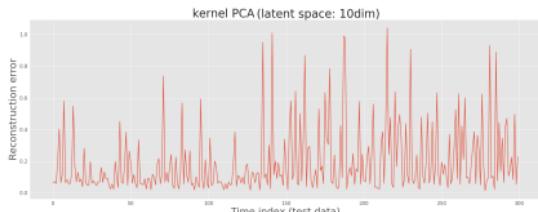
Risultati dataset artificiale II



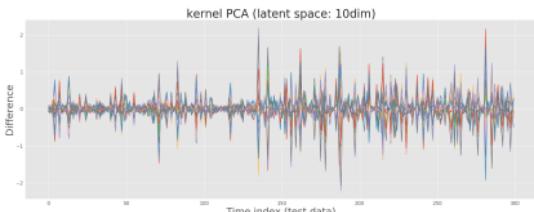
(a)



(b)



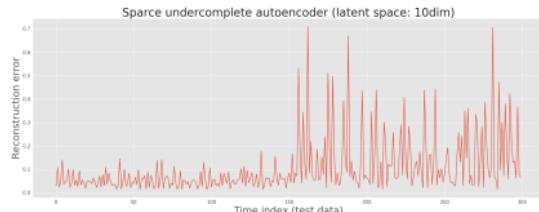
(c)



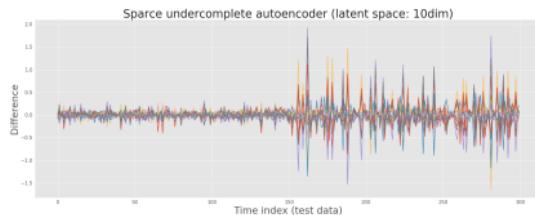
(d)

Figure: L'errore di ricostruzione (sulla sinistra) e la differenza tra i dati originali e quelli ricostruiti (sulla destra) per linear PCA e Kernel PCA

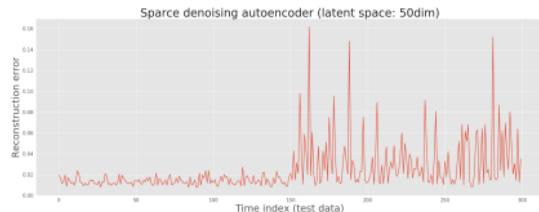
Risultati dataset artificiale III



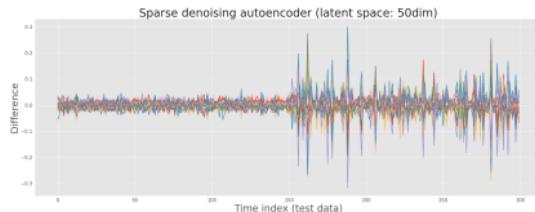
(a)



(b)



(c)



(d)

Figure: L'errore di ricostruzione (sulla sinistra) e la differenza tra i dati originali e quelli ricostruiti (sulla destra) per autoencoder sparsi e denoising autoencoder

Risultati dataset artificiale IV

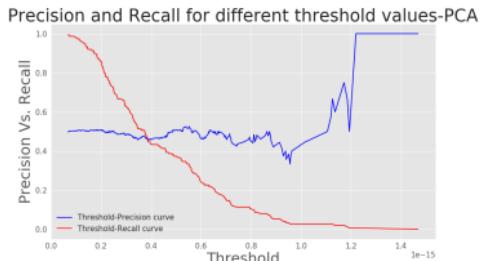
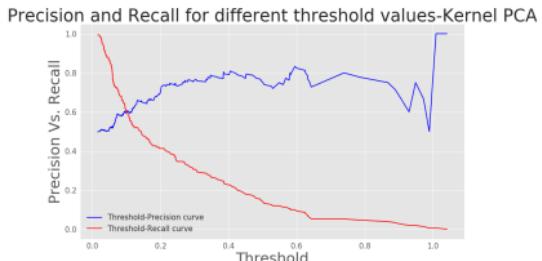
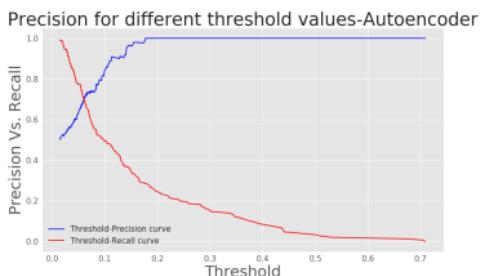
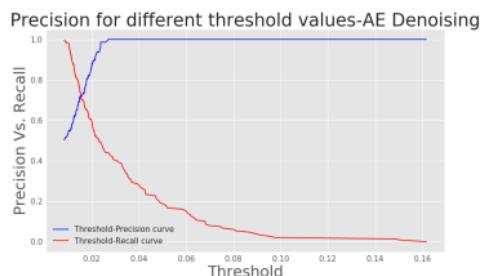
(a) *Linear PCA*(b) *Kernel PCA*(c) *Autoencoder spars*(d) *Denoising autoencoder*

Figure: Confronto tra i valori di Precision e Recall al variare del threshold

Risultati dataset artificiale V

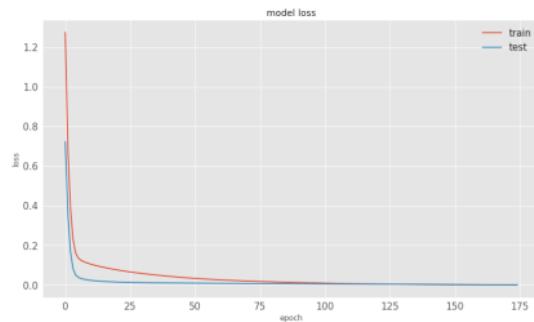
(a) *Autoencoder sparso*(b) *Denoising autoencoder*

Figure: Funzione Loss

Risultati dataset artificiale VI

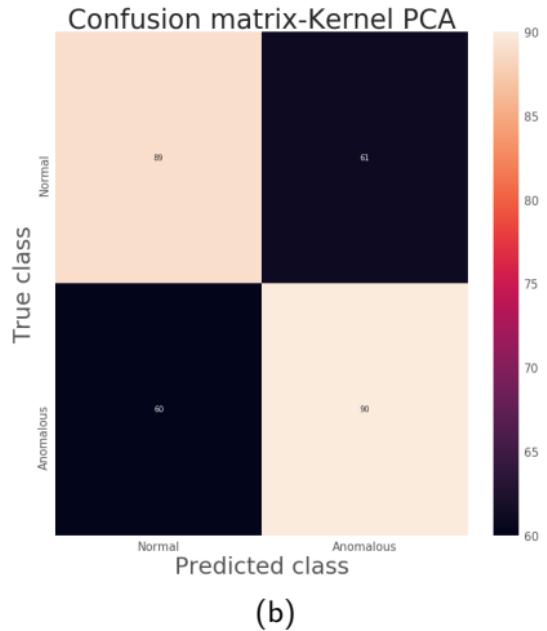
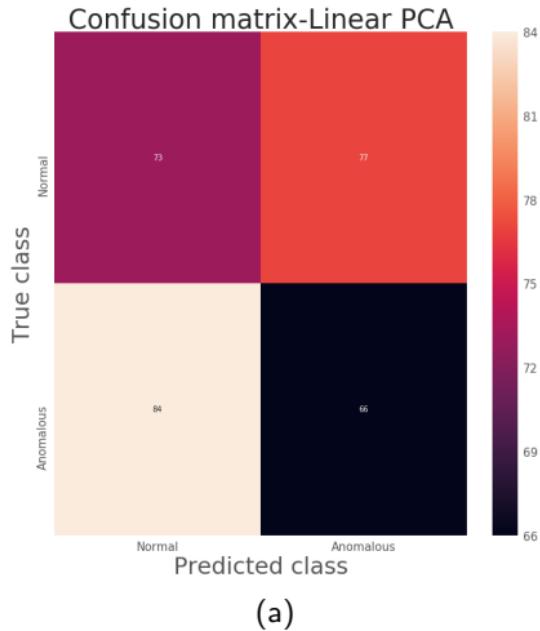


Figure: Matrici di confusione Linear PCA e Kernel PCA

Risultati dataset artificiale VII

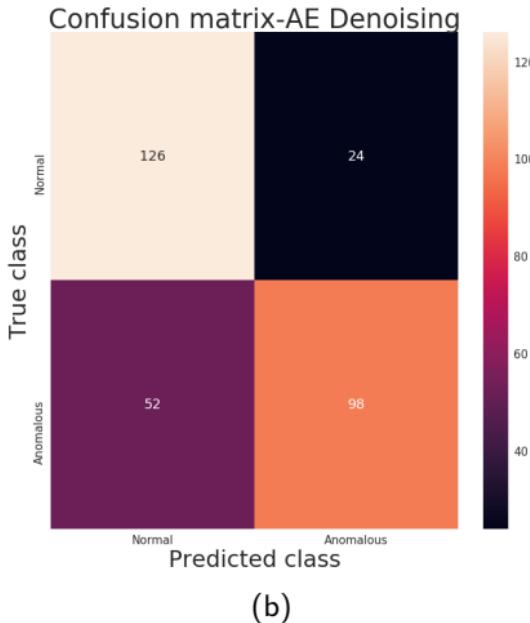
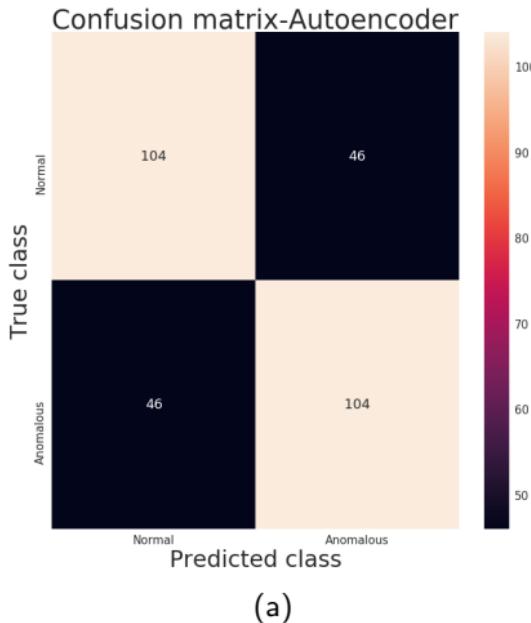


Figure: Matrici di confusione Autoencoder sparso e Denoising Autoencoder

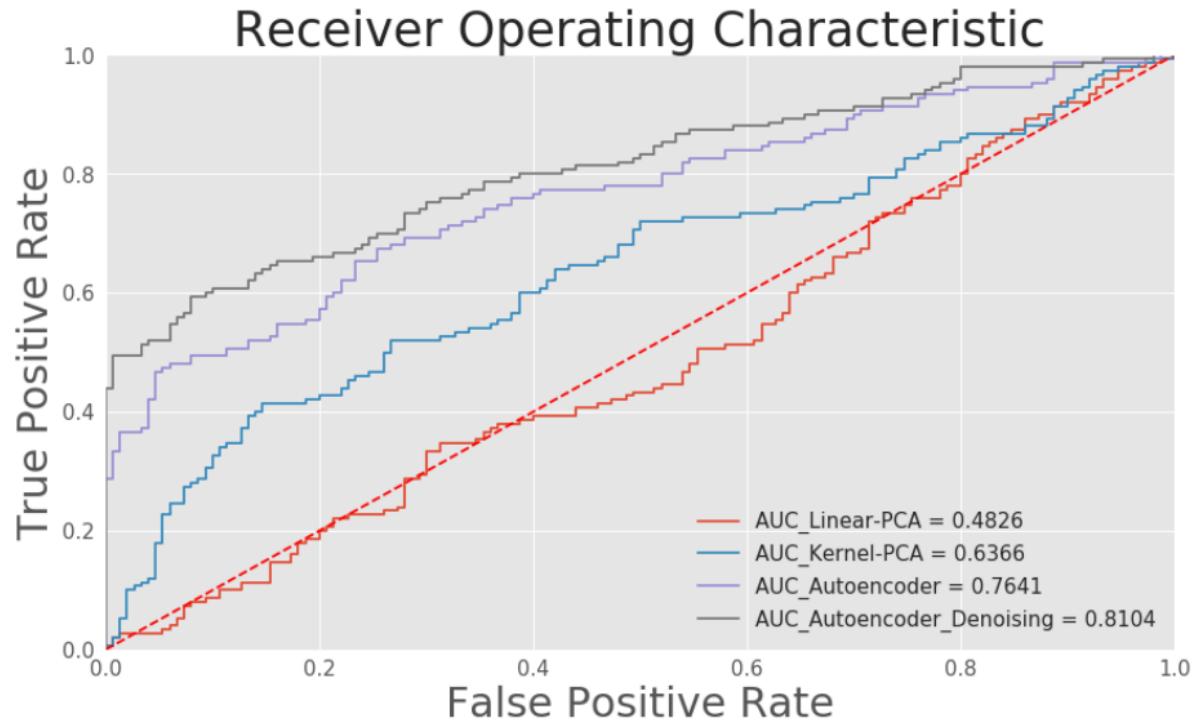
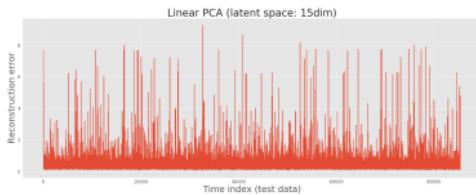
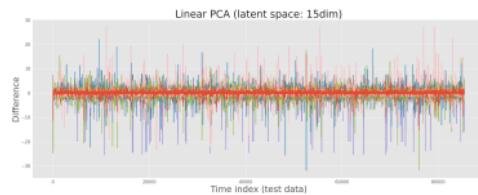


Figure: Curve ROC ottenute utilizzando le tecniche descritte

Risultati dataset credit card I



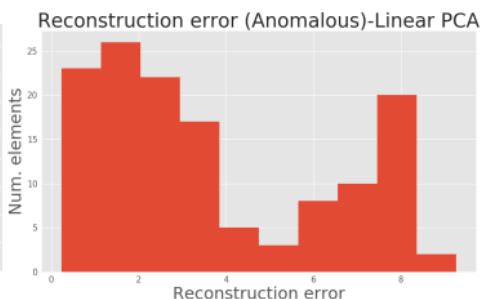
(a)



(b)



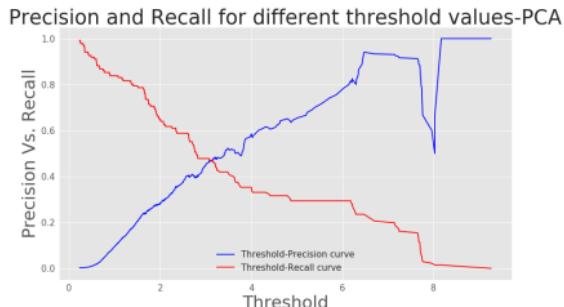
(c)



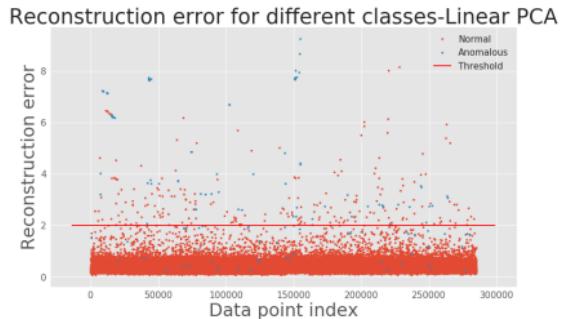
(d)

Figure: Gli istogrammi ci dicono che i dati anomali hanno un elevato errore di ricostruzione

Risultati dataset credit card II



(a) *Precision e recall per PCA*



(b) *Rilevamento delle anomalie*

Figure: Confronto tra i valori di Precision e Recall al variare del threshold e rilevamento delle anomalie

Risultati dataset credit card III

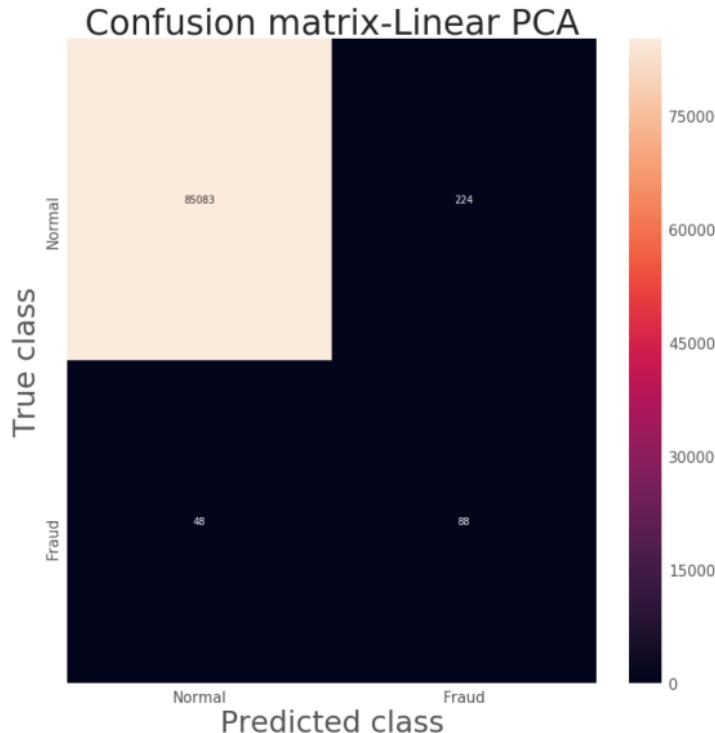
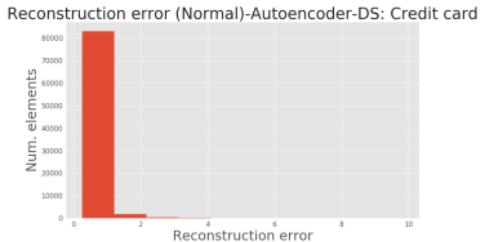
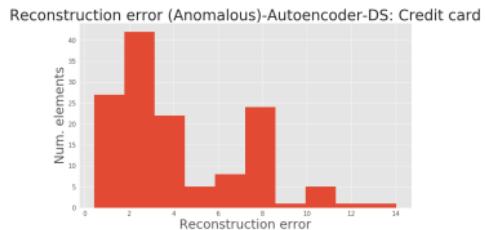


Figure: Matrice di confusione per la PCA su credit card

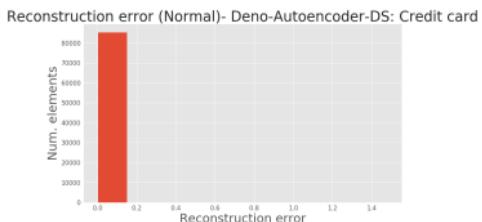
Risultati dataset credit card IV



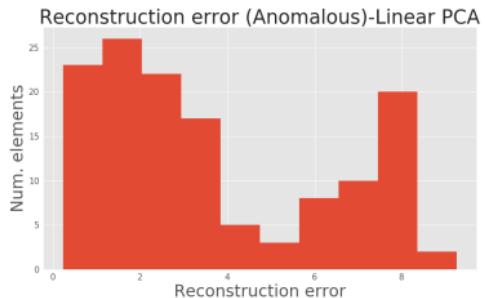
(a) *Errore di ricostruzione per i dati normali AE*



(b) *Errore di ricostruzione per i dati anomali AE*



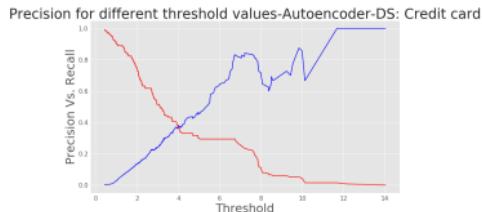
(c) *Errore di ricostruzione per i dati normali DAE*



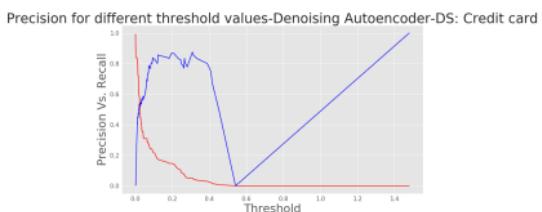
(d) *Errore di ricostruzione per i dati normali DAE*

Figure: L'errore di ricostruzione per gli autoencoders

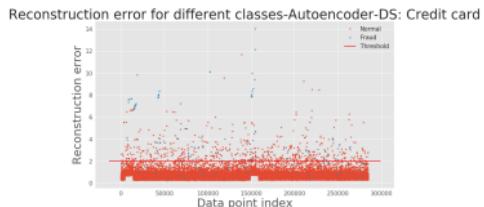
Risultati dataset credit card V



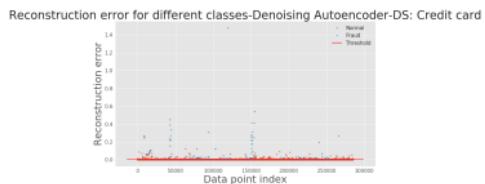
(a) *Precision e Recall per Autoencoder sparso*



(b) *Precision e Recall per Denoising Autoencoder*



(c) *Rilevamento di anomalie per Autoencoder sparso*



(d) *Rilevamento di anomalie per Denoising autoencoder*

Figure: Confronto tra i valori di Precision e Recall al variare del threshold per gli autoencoders e rilevamento delle anomalie

Risultati dataset credit card VI

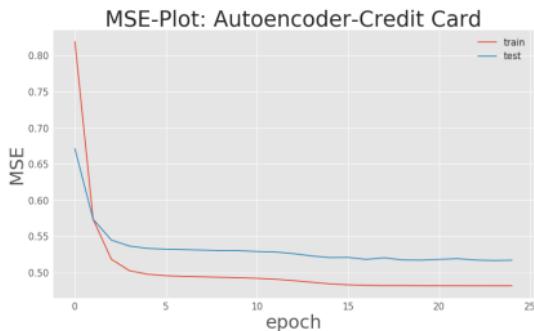
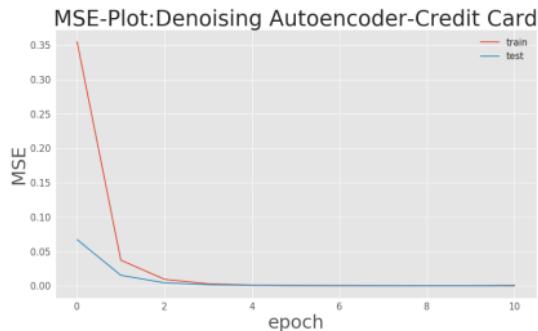
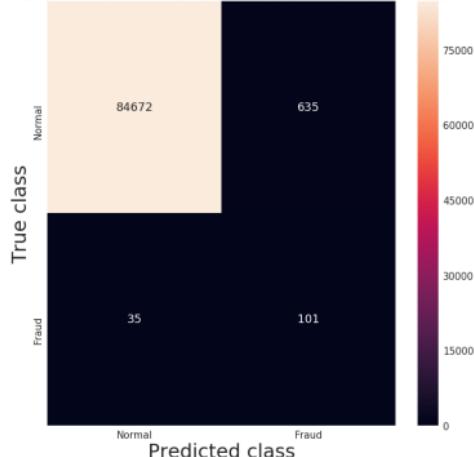
(a) *Autoencoder sparso*(b) *Denoising autoencoder*

Figure: Funzione Loss

Risultati dataset credit card VII

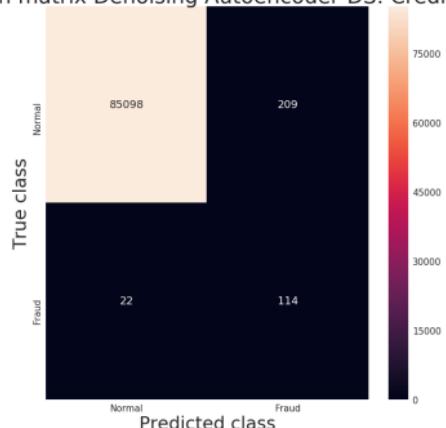


Confusion matrix-Autoencoder-DS: Credit card



(a)

Confusion matrix Denoising Autoencoder-DS: Credit card



(b)

Figure: Matrici di confusione Autoencoder sparso e Denoising Autoencoder

Risultati dataset credit card VIII

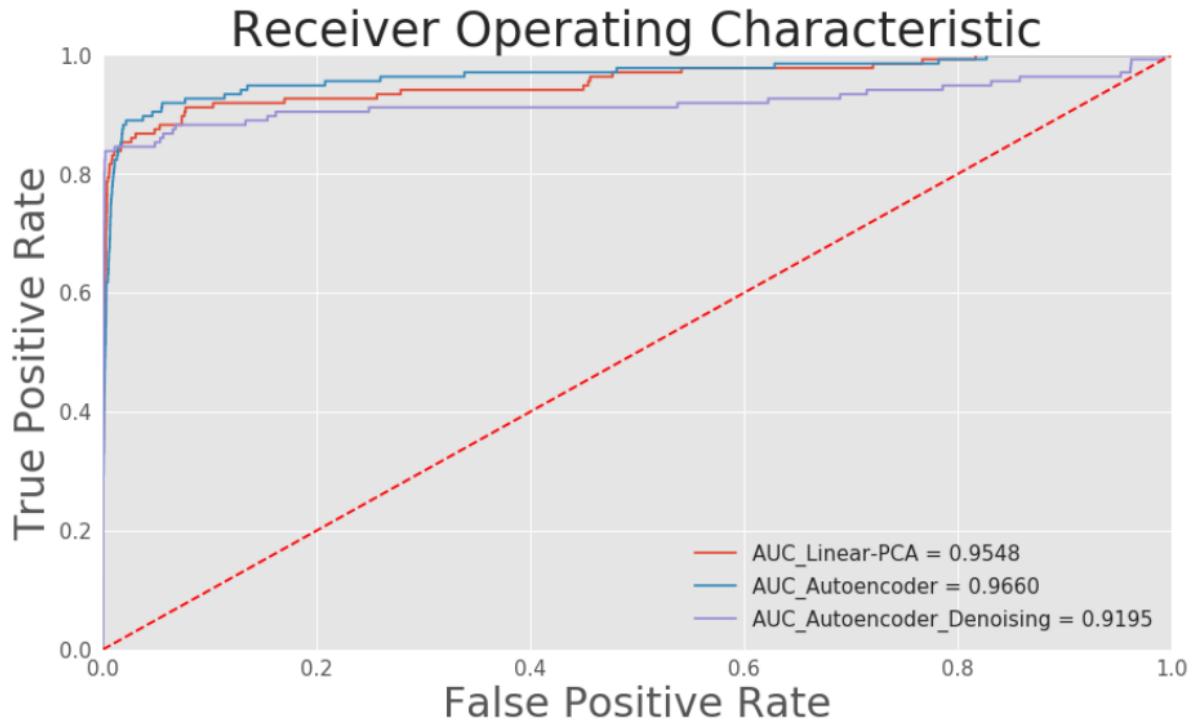


Figure: Curve ROC ottenute utilizzando le tecniche descritte

Conclusioni I



- Dai risultati ottenuti applicando le tecniche di PCA, PCA Kernel, Autoencoder sparso e Denoising autoencoder sui dati generati artificialmente si può osservare che le tecniche non lineari danno buoni risultati; il Denoising Autoencoder è quello con prestazioni migliori.
- Sul dataset credit_card i metodi PCA e Autoencoder sparso si equivalgono, mentre Denoising Autoencoder è quello con prestazioni migliori.
- Sul dataset credit_card la Kernel PCA non è stato possibile utilizzarla per via delle grandi dimensioni del dataset, le altre tecniche sì.
- Se nel dataset sono presenti molte variabili correlate, i metodi non lineari rilevano correttamente la maggior parte delle anomalie mentre la PCA no.

Conclusioni

Le tecniche Autoencoder sono quindi applicabili su dataset molto diversi tra di loro (con dimensioni elevate e/o con la presenza o meno di molte variabili correlate) e dimostrano di essere degli strumenti molto potenti per il riconoscimento di anomalie rispetto a Kernel PCA e PCA.



[1] M. Sakurada and T. Yairi, *Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction*, 2014 ACM

Grazie per l'ascolto!