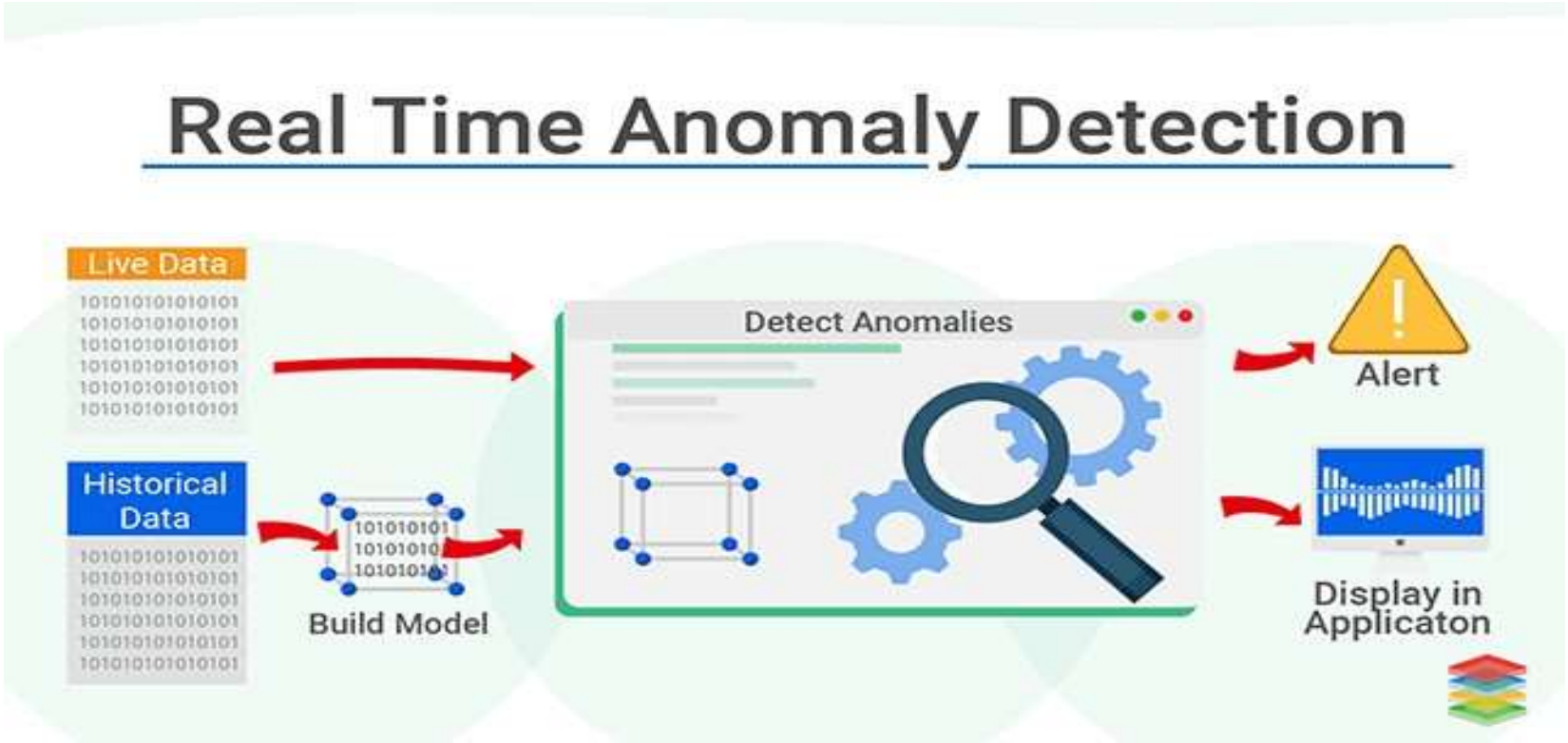


PROGETTO METODI INFORMATICI PER L'ANALISI DEI PROCESSI A.A: 2019/2020

"Analisi degli algoritmi Node2Vec e Graph2Vec per la creazione di modelli"

Studenti
Emilio Casella matr.204898
Davide Costa matr.205167
Antonio Gagliostro matr.207059

Docente
Prof. Guzzo Antonella

L’elaborato è composto dai seguenti punti:

1. Preconcetti;
2. Stato dell’arte sul Graph Embeddings nell’anomaly detection per Processi di Business;
3. Implementazione di Node2vec e Graph2Vec;
4. Confronto;
5. Conclusioni e ulteriori approfondimenti;

Embeddings sui grafi

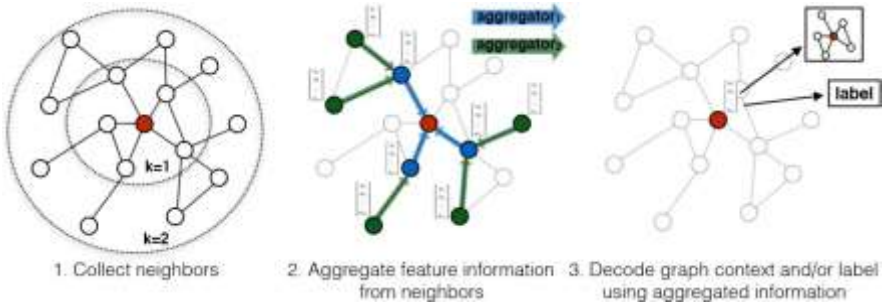
•Vertex embeddings: si codifica ogni vertice (nodo) con la sua rappresentazione vettoriale; questo incorporamento viene usato quando si desidera eseguire la visualizzazione o la previsione a livello di vertice, ad es. visualizzazione di vertici sul piano 2D o la previsione di nuove connessioni basate su similitudini di vertici.

•Graph embeddings: si rappresenta l'intero grafo con un singolo vettore per fare previsioni a livello di grafo o quando si vuole confrontare o visualizzare tutti i grafi, ad es. confronto di strutture chimiche.

Un embeddings, per essere considerato un buon livello di rappresentazione, deve soddisfare i seguenti requisiti:
• Rappresentare necessariamente la topologia del grafo, i suoi nodi, le sue connessioni e i suoi vicini. Le performance di visualizzazione o previsione dipenderanno dal grado di accuratezza;

• La taglia del grafo non deve influenzare le performance del processo di embeddings;

• Bisogna trovare un buon compromesso tra la taglia del grafo e la complessità spaziale causata dalle innumerevoli informazioni da preservare.



Word2vec

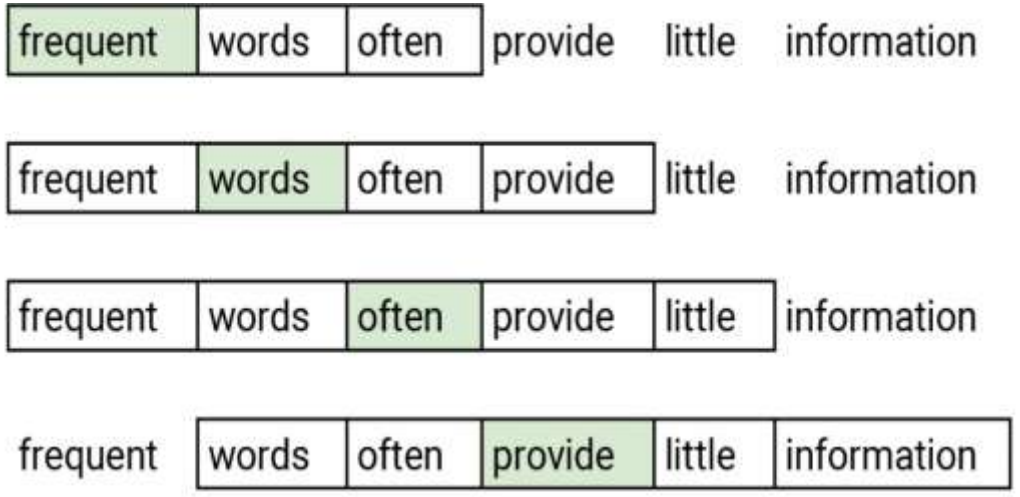
Word2Vec prende come input un grande frammento di testo e costruisce uno spazio vettoriale, tipicamente di diverse centinaia di dimensioni, in cui ogni parola è univocamente assegnata ad un corrispondente vettore nello spazio, seguendo un certo criterio.

Infatti i vettori (rappresentanti le parole) vengono posizionati nello spazio cosicché le parole che risultino "simili" all'interno del frammento di testo, siano collocate vicine tra loro nello spazio stesso.

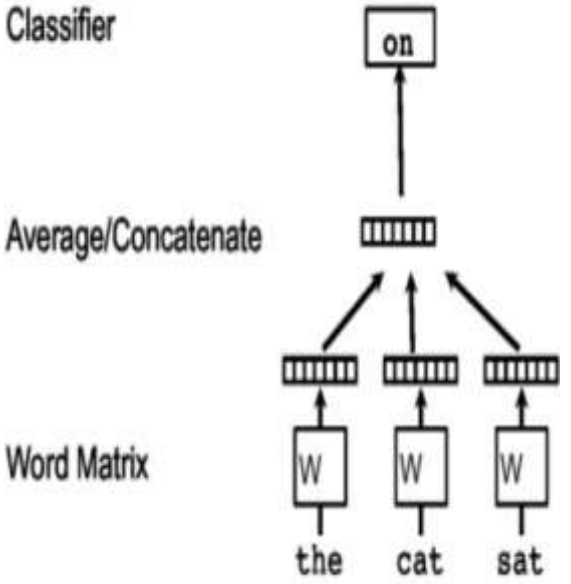
L'idea è quella di riuscire ad incapsulare relazioni differenti tra le parole, come ad esempio sinonimi, contrari o analogie.

•**Skip-Gram model:** in cui si prendono coppie di parole dal testo e si addestra una rete neurale con un livello di nodi di hidden sulla base del finto task in cui, a partire da una parola in input, la rete restituisce la distribuzione di probabilità delle parole vicine (all'interno del testo) all'input. In altre parole, la rete restituirà alte probabilità per le parole che tipicamente compaiono "vicine" alla parola in input. Essendo un finto task però, ad essere realmente importanti saranno i pesi tra i nodi di input e i nodi di hidden che saranno usati come word embedding. Quindi se lo strato di hidden possiede 300 neuroni, la rete restituirà un vettore di dimensione pari a 300, per ciascuna parola.

•**A continuous bag of words (CBOW):** anch'essa fa uso di una rete neurale con un livello di nodi di hidden. Il finto task in questo caso è basato sul predire una parola centrale, a partire da un insieme di parole di input (appartenenti allo stesso contesto); saranno poi i pesi tra il livello di input e di hidden ad essere usati come word embeddings per i termini dati in ingresso alla rete.



The word colored with green is given to the network. It is optimized to predict the word in the neighborhood with higher probability. In this example, we consider words that are the most two places away from the selected words.



Node2Vec

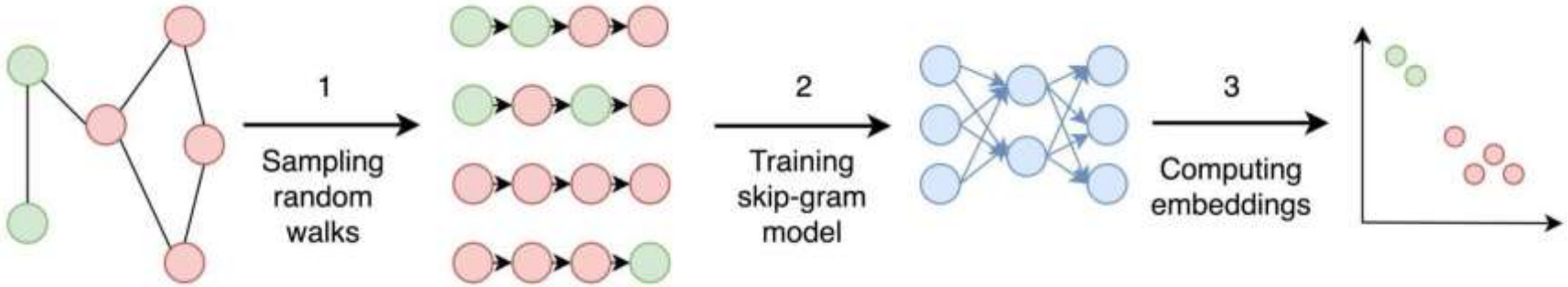
L’algoritmo node2vec appartiene al gruppo di algoritmi vertex embeddings.
La sua base è caratterizzata da un approccio di tipo DeepWalk, ovvero un’applicazione del classico random walk alle reti di apprendimento. Per prima cosa ci si muove selezionando un nodo e ci si sposta su un vicino casuale dal nodo selezionato, in un determinato numero di passi.

Il metodo consiste sostanzialmente in tre passaggi:

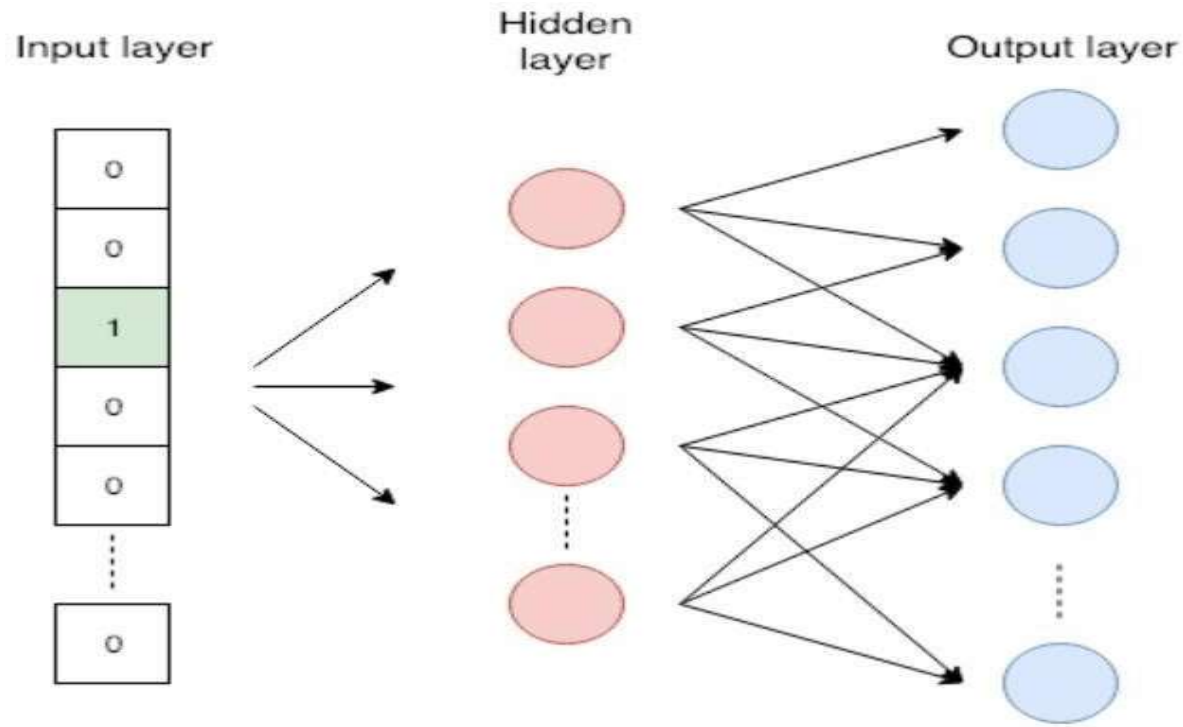
- Campionamento: un grafico viene campionato con passeggiate casuali. Vengono eseguiti pochi passi casuali da ciascun nodo. I buoni cammini random compiono circa 30 passi;

- Allenamento skip-gram: le passeggiate casuali sono paragonabili alle frasi nell'approccio word2vec. La rete skip-gram prende in input un nodo del random walk come un vettore one-hot e massimizza la probabilità di prevedere i nodi vicini.

- Computing embeddings: E’ l'output di un livello nascosto della rete; DeepWalk lo calcola per ciascun nodo nel grafo.



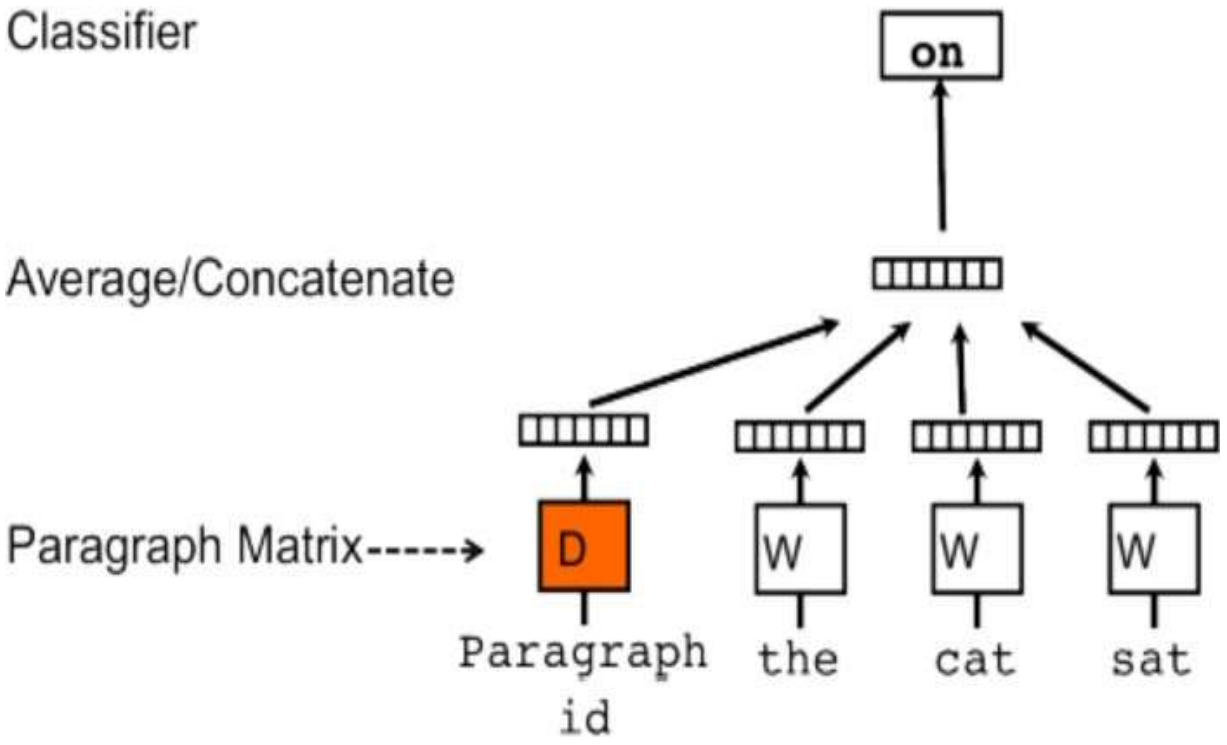
Phases of DeepWalk approach



Skip-gram neural network

Doc2Vec

In maniera analoga agli scopi del Word2Vec, l'obiettivo del Doc2Vec è di creare una rappresentazione numerica di un intero documento (o paragrafo, nel nostro caso di un tweet) a prescindere dalla sua lunghezza. Il principio usato è semplice ed intelligente:
si fa uso del modello word2vec e si aggiunge un altro vettore, detto **Paragraph ID**. Quindi dopo aver addestrato la rete neurale, si avranno oltre che i word vectors (la rappresentazione vettoriale delle parole) anche un document vector (la rappresentazione vettoriale del documento).

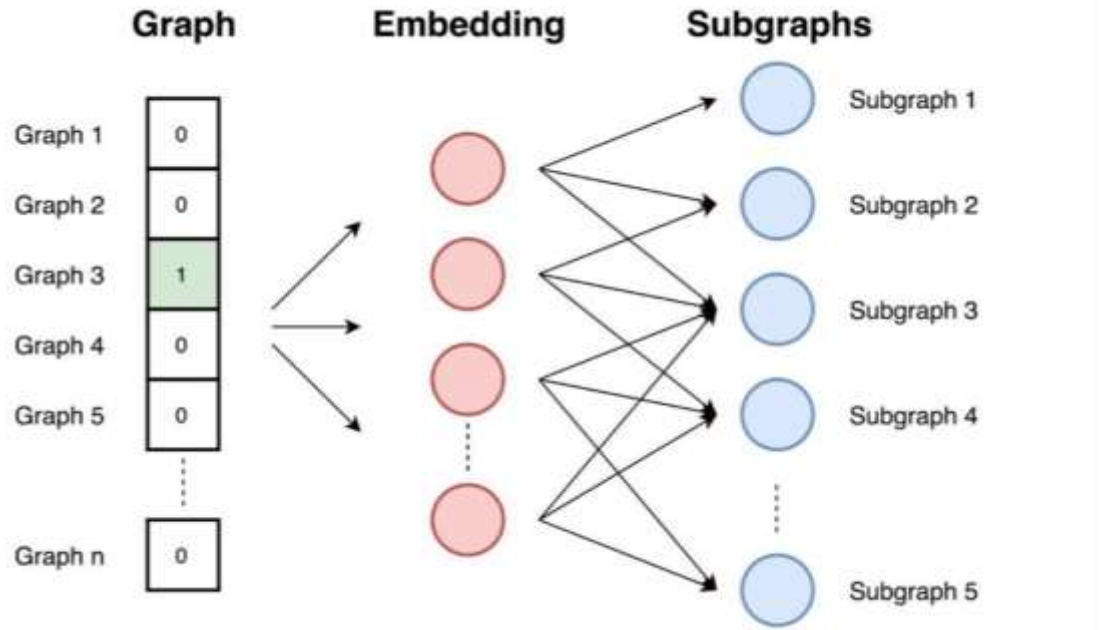


Anche nel caso del Doc2Vec esistono due metodologie analoghe rispettivamente al CBOW e lo Skip-Grim:

- **Distributed Memory version of Paragraph Vector (PV-DM):** che agisce come una memoria che ricorda cosa manca esattamente all'interno del contesto in questione o come l'argomento del paragrafo. Mentre i word vectors rappresentano il concetto di una parola, il document vector intende rappresentare il concetto di un documento;

- **Distributed Bag of Words version of Paragraph Vector (PV-DBOW):**

che risulta essere più veloce e che consuma meno memoria (contrariamente a quanto accade allo skip-gram nel word2vec) in quanto non c'è alcun bisogno di salvarsi i word vectors (rappresentazioni vettoriali delle parole). Dopo l'addestramento infatti, basta fornire il paragraph ID (detto anche tag) per ricevere il document vector.



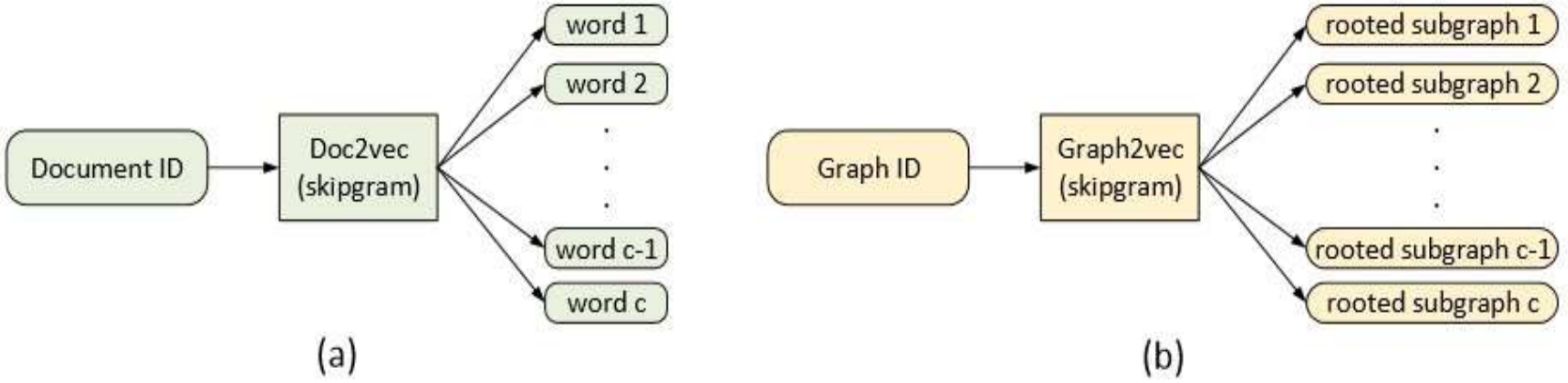
Graph2vec

Graph2vec si basa su doc2vec che utilizza una rete di skip-gram. Ottiene un ID del doc sull'input ed è addestrato per massimizzare la probabilità di prevedere parole casuali tramite il doc stesso.
Gli approcci Graph2vec consistono in tre fasi:

• Campionamento e ri-etichettatura di tutti i sotto-grafi dal grafo: il sotto-grafo è un insieme di nodi che appaiono attorno al nodo selezionato. I nodi nel sotto-grafo non sono superiori al numero selezionato di passi;

•Allenare il modello skip-gram: I grafi sono simili ai doc. Poiché i doc sono un insieme di word, i grafi sono un insieme di sotto-grafi. In questa fase viene addestrato il modello skip-gram e deve massimizzare la probabilità di prevedere un sotto-grafo esistente nel grafo in input. Il grafo di input viene fornito come un vettore one-hot;

•Calcolo di embeddings: Si fornisce un ID tramite un vettore one-hot in input. L'embeddings consiste nel risultato prodotto dal livello nascosto.



Business Process e Machine learning

Rilevare anomalie nei propri processi interni è un compito oneroso per ogni azienda ma necessario, poiché potrebbe essere indicatore di frodi e inefficienze varie.

Le tecniche di process mining classiche forniscono metodologie di rilevazione di anomalie nell'esecuzione di un processo che richiedono l'esistenza di un modello di riferimento.

Tramite l'utilizzo di tecniche di machine learning, se non è disponibile quest'ultimo, si va alla scoperta di un modello di riferimento dal registro eventi stesso; utilizzando una soglia per gestire comportamenti rari, in modo che il modello rilevato sia una buona rappresentazione del normale comportamento del processo. Pertanto, questi può essere utilizzato come modello di riferimento per il controllo di conformità.

Presupposti chiave nel rilevamento di anomalie è che le esecuzioni anomale si verificano meno frequentemente delle esecuzioni normali.

Act2vec

Act2vec è una architettura di apprendimento utilizzata per la rappresentazione delle attività. Supponiamo di avere i dati in input sotto forma di un registro ad eventi.

In linea con l'approccio Word2vec nell'elaborazione del linguaggio naturale, possiamo rappresentarle considerando le attività come word in un corpus, con il corpus che è il registro eventi nel nostro caso;

L'architettura si basa sul principio secondo cui una parola può essere prevista dal suo contesto (cioè le parole che appaiono prima e dopo la parola chiave ;

Le tracce saranno quindi considerate come frasi ed eventi come parole. Pertanto, l'algoritmo apprende rappresentazioni di attività generiche che non sono adattate, ad esempio, alla previsione dell'attività successiva o alla previsione del tempo rimanente delle istanze;

Si noti che, nonostante le tracce in un registro eventi siano di natura sequenziale, si sceglie di definire il contesto di un'attività in base all'insieme non ordinato di attività precedenti e successive; ottenendo un'architettura di apprendimento che può essere utilizzata per comprendere il contesto di due diverse attività. A seconda del loro contesto, la loro rappresentazione sarà simile o molto diversa;

Per accelerare il processo di addestramento della rete neurale, in quanto, in ogni aggiornamento della rete, solo una piccola percentuale dei suoi parametri vengono aggiornati, modificando i pesi in base al campionamento del vettore di output (un vettore one-hot).

L'architettura Act2vec può essere estesa in diversi modi, utilizzando altri attributi correlati che potrebbero includere altri dati disponibili nel registro eventi, nella definizione del contesto di un'attività o nel concetto da prevedere.

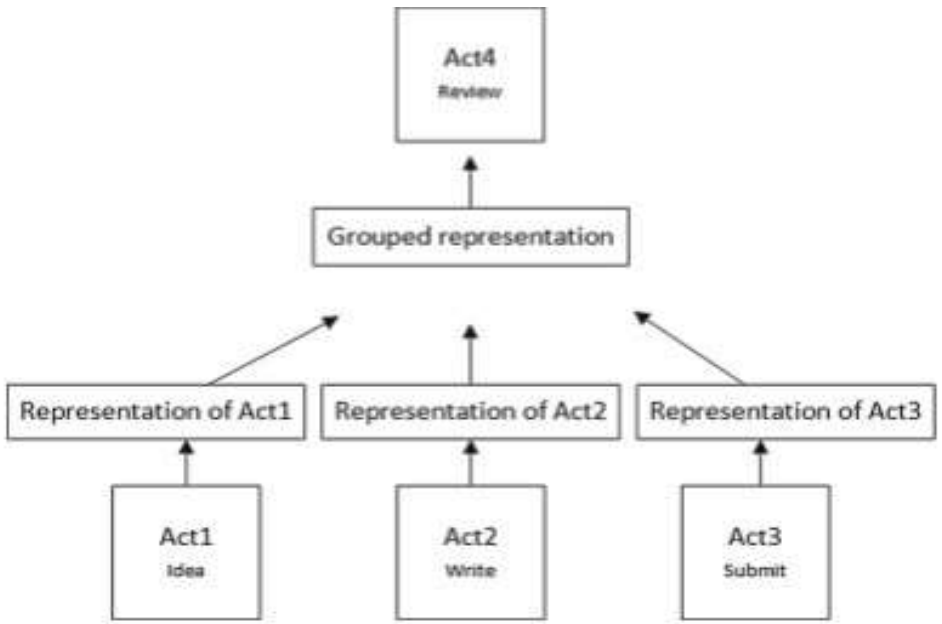


Fig. 1. The *act2vec*-architecture for learning vector representations of activities. The context consisting of activities "idea", "write", and "submit" is used to predict activity "review".

Trace2vec

Seguendo l'analogia tra attività e word, anche le tracce possono essere considerate frasi. L'apprendimento di rappresentazioni distribuite di frasi, paragrafi o documenti è stato introdotto nel dominio dell'elaborazione del linguaggio naturale sotto forma dell'approccio doc2vec, come spiegato precedentemente.
Questa idea può essere adottata anche per le tracce, dando origine all'architettura Trace2vec.
Dato che questa architettura include una rappresentazione di tracce (basata sull'identificatore di tracce), consentirà l'apprendimento congiunto di rappresentazioni di attività e tracce (alle quali siamo principalmente interessati).

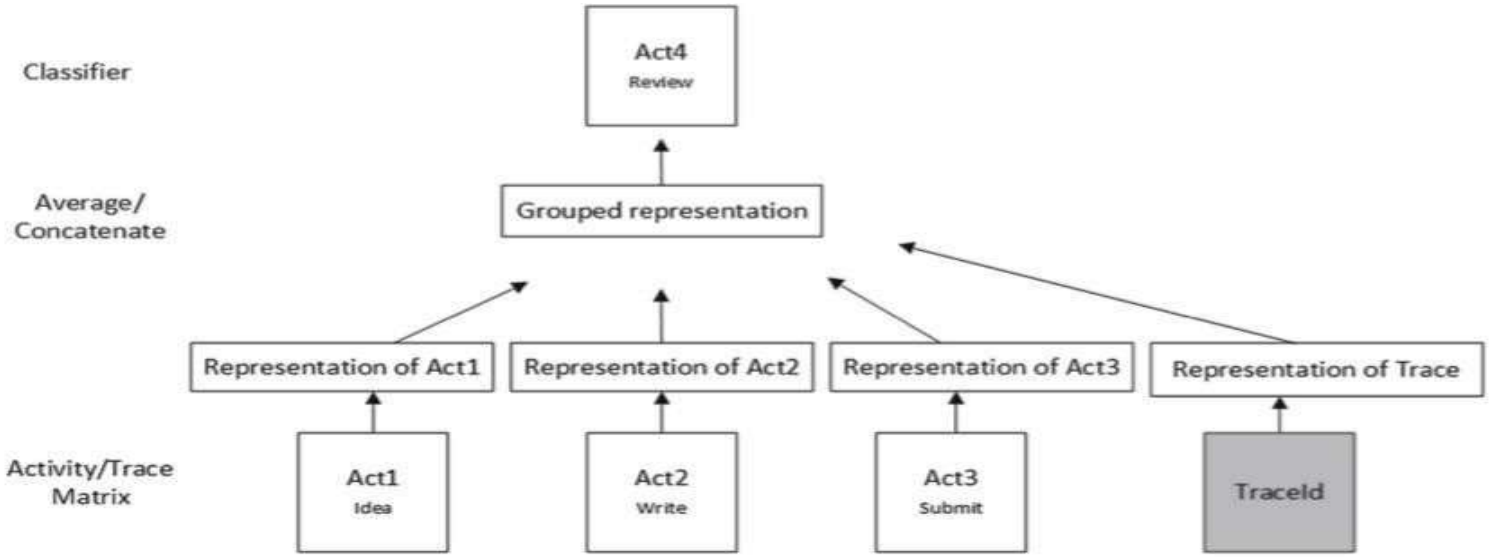


Fig. 2. The *trace2vec*-architecture for learning vector representations of traces. The context consisting of activities "idea", "write", and "submit", as well as the trace-id, is used to predict activity "review".

Log2vec

In linea con Trace2vec, un progetto architetturale può essere concepito per apprendere rappresentazioni distribuite dei registri, sostituendo sostituire la rappresentazione di una traccia con una rappresentazione di un registro.
Utilizziamo l'identificatore delle tracce come prima, tuttavia, dato che i processi aziendali potrebbero condividere varianti di esecuzione simili, proponiamo di includere un identificatore artificiale relativo a istanze di processo distinte nell'architettura (ovvero un identificatore di “variante” di traccia).
Più in particolare, tutte le tracce dei diversi event log in esame vengono unite in un registro ad eventi basato su quell'input, permettendo di calcolare un identificatore di istanza del processo distinto.

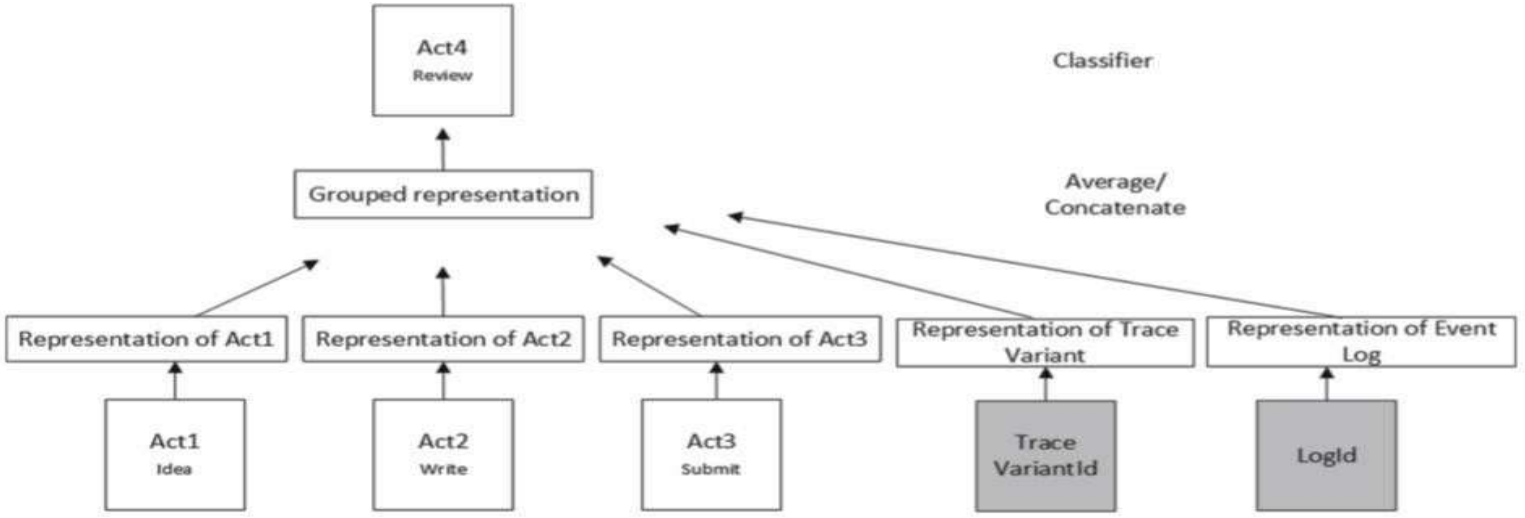


Fig. 3. The *log2vec*-architecture for learning vector representations of logs. The context consisting of activities “idea”, “write”, and “submit”, an identifier for each trace variant, as well as the log-id from which the words are sampled, is used to predict activity “review”.

Model2vec

L'obiettivo è quello di rappresentare "process model" come vettori di piccole dimensioni, basandosi sulle architetture sopra discusse, un'estensione banale potrebbe essere quella di ottenere una rappresentazione generica simulando prima i modelli (tenendo traccia dei dati, degli eventi prodotti durante la simulazione) e applicando successivamente Log2vec, per apprendere una rappresentazione del modello di processo.
Si utilizza un approccio random walk, come illustrato in precedenza, prendendo in considerazione solo indirettamente il comportamento effettivo del modello, ma ponendo maggiormente l'accento sul layout grafico e le relazioni tra gli elementi di modellazione, compresi quelli che non sono direttamente correlati alla rappresentazione di un'attività (come gateway, luoghi o altri costrutti).

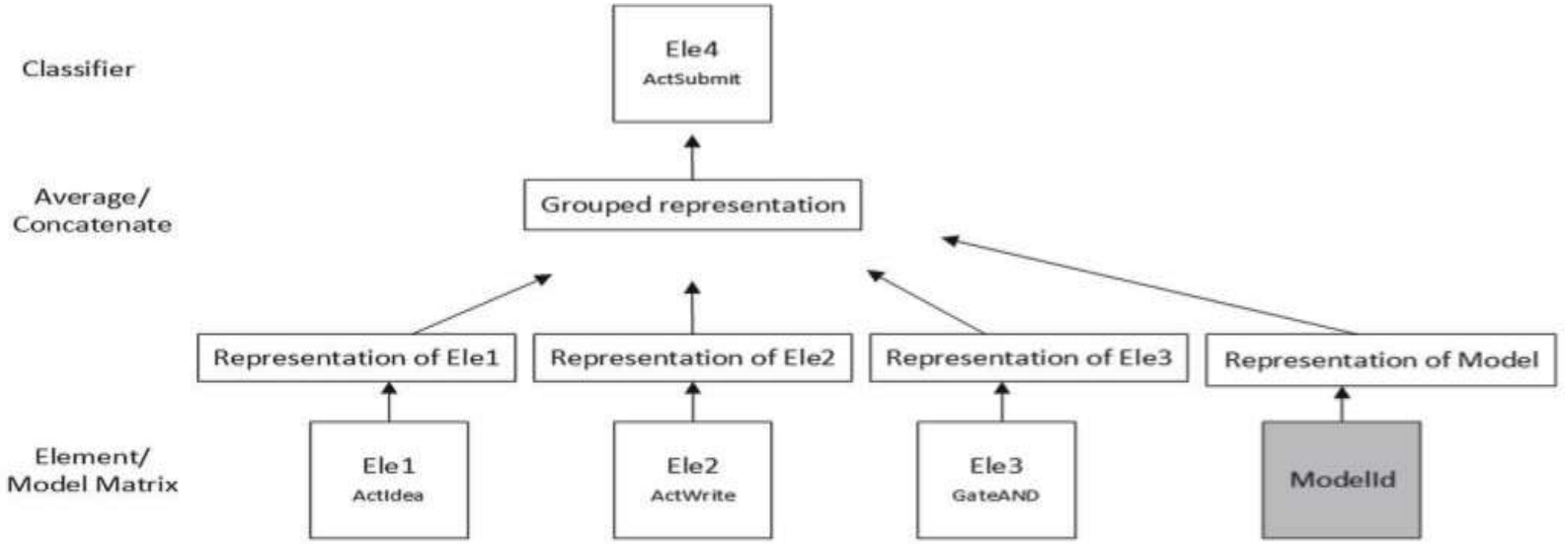


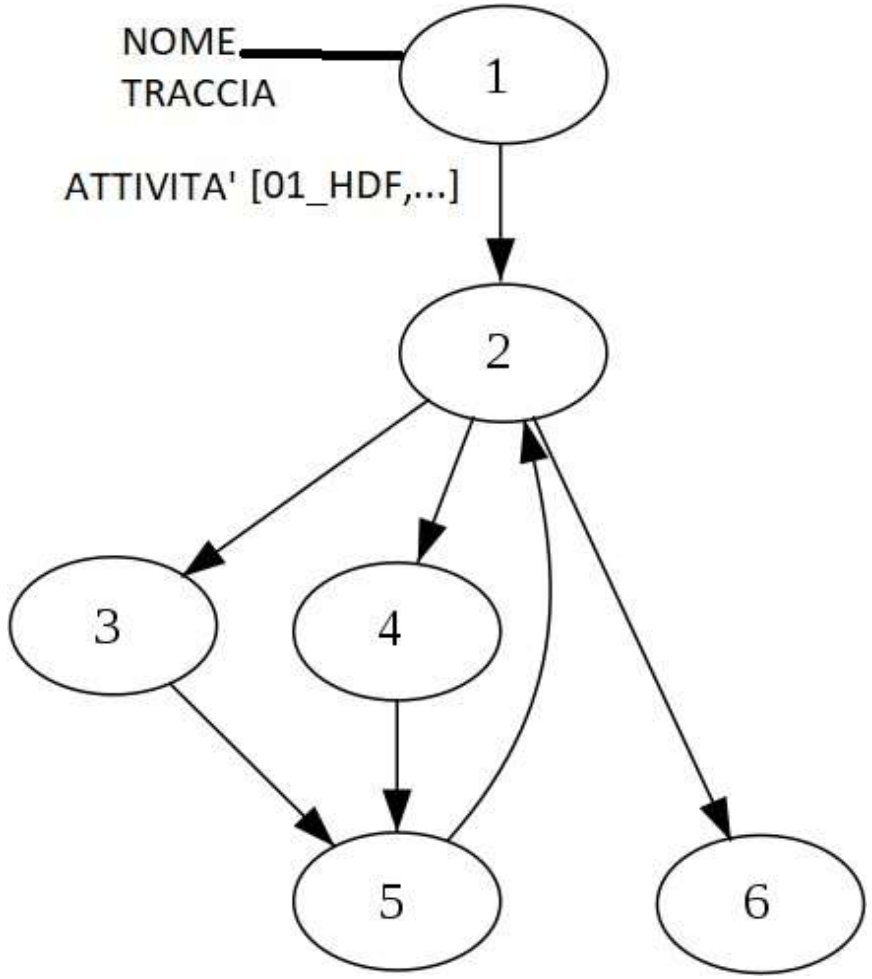
Fig. 4. The *model2vec*-architecture for learning vector representations of logs. The context of elements resulting from either simulation or random walk ("idea", "write", "And-gateway"), as well as the model-id for the model on which the simulation is performed or the walk is sampled, are used to predict activity "submit".

Implementazione Node2Vec

Per l'implementazione di Node2vec si utilizza un pacchetto specifico del linguaggio, basato su un apprendimento di Word2Vec di tipo skip-gram.
Si ottengono 2 modelli:

Il primo, produce un grafo orientato caratterizzato dalla corrispondenza "traccia-insieme di attività che la compongono", ispirandosi all'algoritmo Trace2vec. In particolar modo, ogni nodo conterrà il valore che identifica una traccia e i collegamenti tra i vari nodi, saranno creati sulla base delle attività contenute nella traccia stessa.

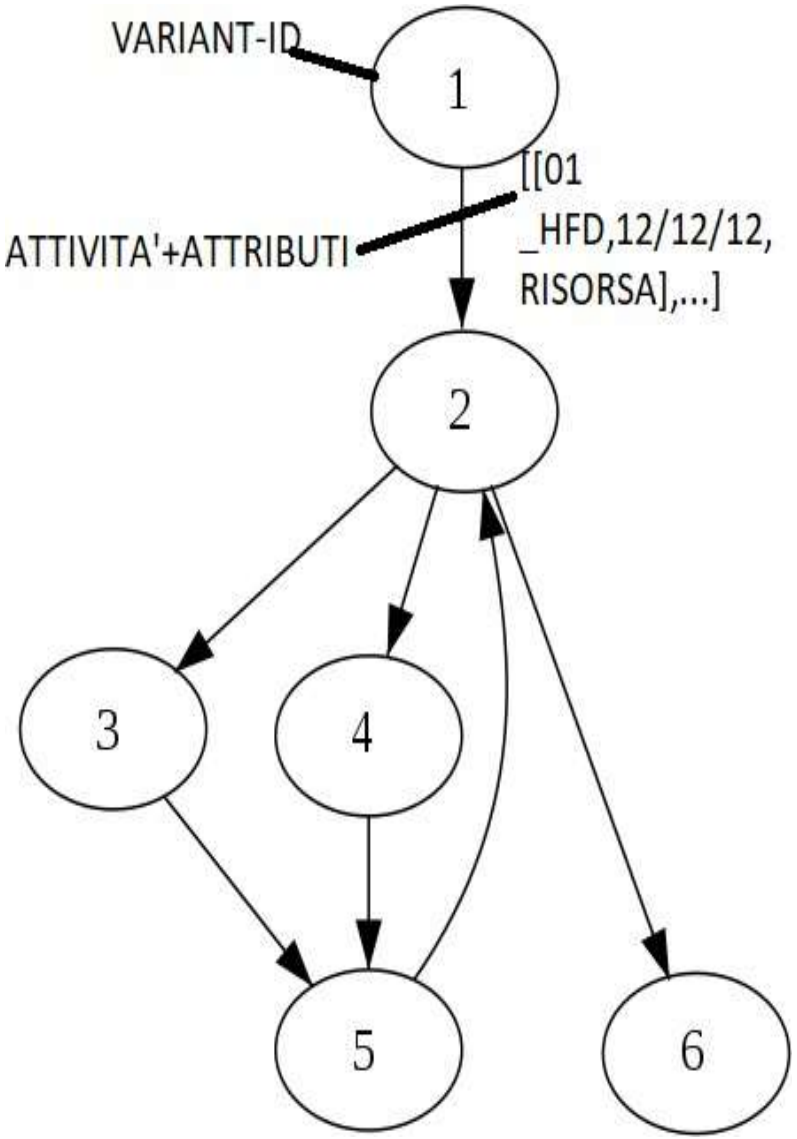
Precisamente, i nodi 1 e 2 saranno collegati tra loro se entrambi in possesso di almeno un'attività in comune. Per limiti di risorse hardware e del pacchetto Node2vec stesso, limitato per grandi reti con molte info, il collegamento tra due nodi sarà unico, anche se i due dovessero avere più attività in comune, con relativa etichettatura.




```
def learnT(logName,vectorsize):
    graph=nx.Graph()
    buildGT(graph,logName)
    node2vec = Node2Vec(graph, dimensions=vectorsize, walk_length=30, num_walks=200,workers=8)
    model = node2vec.fit(window=10, min_count=1, batch_words=4)
    model.save('output/'+'N2VVST'+str(vectorsize)+'.model')
```

Implementazione Node2Vec

Il secondo modello prodotto segue la falsa riga del primo. I nodi, del grafo orientato, conterranno l'id variant della traccia in esame, i collegamenti tra i nodi sono sviluppati in base al valore dell'attività svolta, con attributi relativi alla risorsa utilizzata e al timestamp relativo all'attività, ispirandosi all'algoritmo Act2vec.
Il set di attività-attributi relativi ad ogni id-trace è limitato, così come il set di collegamenti ed etichettatura è limitato, anche qui, a causa dell'hardware e dell'implementazione stessa di Node2vec.




```
def learnV(logName,vectorsize):
    graph=nx.Graph()
    buildG(graph,logName)
    node2vec = Node2Vec(graph, dimensions=16, walk_length=30, num_walks=200,workers=8)
    model = node2vec.fit(window=10, min_count=1, batch_words=4)
    model.save('output/'+'N2VVS'+str(vectorsize)+'.model')
```

```
def buildG(graph,logName):
    max=5000
    attivita=[]
    for element in loadXES.Vget_sentences_XES(logName + ".xes"):
        attivita.append(element)
    NodiAttivita={}
    i=0
    c=0
    for variant in loadXES.get_variant_names(logName+".xes"):
        if(c<=max):
            NodiAttivita[variant]=attivita[i]
            i=i+1
            c=c+1
        else:
            break
    for key in NodiAttivita.keys():
        graph.add_node(key)
    for var1 in NodiAttivita.keys():
        for var2 in NodiAttivita.keys():
            if(var1!=var2):
                x = NodiAttivita[var1]
                y = NodiAttivita[var2]
                for el in x:
                    if(c<max):
                        if(y.__contains__(el)):
                            graph.add_edge(var1,var2,attr=el)
                            c=c+1
                            break
                    else:
                        break
```

si riporta la costruzione dell'albero per il modello "attività+attributi" di Node2Vec.

Implementazione Graph2Vec

L'implementazione del modulo Graph2vec, segue la filosofia precedentemente elencata per Node2Vec.
I grafi orientati prodotti, però, saranno implementati non più attraverso la classe Networkx, ma a livello logico; sotto forma di dizionari contenenti come chiave il trace id variant oppure la traccia stessa, e come valore associato alla chiave, "l'insieme di attività" o "insieme di attività+attributi". Ogni elemento in comune andrà a costituire un collegamento nel grafo, con relativa etichettatura, ma solo e puramente a livello logico, in modo da alleggerire in maniera considerevole, il costo di costruzione dell'albero ed elaborazione del modello.

In questo modo, si può utilizzare molto agevolmente, la classe Doc2Vec implementa in Gensim, permettendo inoltre, una facile lettura del codice. Il nodo sarà quindi una chiave, e due chiavi saranno considerate collegate, se avranno almeno una attività o attributo in comune, come avveniva precedentemente. Tutto questo permetterà, come sarà illustrato in seguito, di ottenere performance nettamente migliori, anche a causa della natura stessa dell'algoritmo implementato (Graph2vec è un Doc2Vec).

```python
def learnT(logName,vectorsize):
    documents=buildGT(logName)
    model = gensim.models.Doc2Vec(documents, dm=0, alpha=0.025, vector_size=vectorsize, window=8, min_alpha=0.025,min_count=0, workers=4)
    nrEpochs=4
    for epoch in range(nrEpochs):
        if epoch % 2 == 0:
            print ('Now training epoch %s'%epoch)
        model.train(documents,total_examples=len(documents), epochs=nrEpochs)
        model.alpha -= 0.002
        model.min_alpha = model.alpha
    model.save('output/'+'G2VVST'+str(vectorsize)+'.model')

def learnV(logName,vectorsize):
    documents=buildG(logName)
    model = gensim.models.Doc2Vec(documents, dm=0, alpha=0.025, vector_size=vectorsize, window=8, min_alpha=0.025,min_count=0, workers=4)
    nrEpochs=4
    for epoch in range(nrEpochs):
        if epoch % 2 == 0:
            print ('Now training epoch %s'%epoch)
        model.train(documents,total_examples=len(documents), epochs=nrEpochs)
        model.alpha -= 0.002
        model.min_alpha = model.alpha
    model.save('output/'+'G2VVS'+str(vectorsize)+'.model')
```

```python
def buildG(logName):
    grafoS=[]
    attivita=[]
    for element in loadXES.Vget_sentences_XES(logName + ".xes"):
        attivita.append(element)
    NodiAttivita={}
    i=0
    for variant in loadXES.get_variant_names(logName+".xes"):
            NodiAttivita[variant]=attivita[i]
            i=i+1
    for key in NodiAttivita:
        t=TaggedDocument((np.concatenate(NodiAttivita[key])),[key])
        grafoS.append(t)
    return grafoS
```

L'analisi dei modelli ottenuti, sarà fatta dal modulo MyModel2Vec che implementa i cluster relativi al Trace2Vec e Act2Vec; andando a formare una parte del modulo Log2Vec, suggerito come analisi per il cluster prodotto da Model2Vec.

Per il cluster sui modelli ottenuti, si utilizzano gli algoritmi KMeans e Hierward. Si mostra, a titolo di esempio, il cluster relativo al variant-id, molto simile a quello prodotto dal cluster sulle tracce.

```
def clusterV(logName,vectorsize,nameM,clusterType):
    corpus = loadXES.get_doc_XES_tagged(logName+'.xes')
    vectors = []
    NUM_CLUSTERS = 5
    conta=[]
    if(nameM=='G2VVS'):
        model= gensim.models.Doc2Vec.load('output/'+nameM+str(vectorsize)+'.model')
        print("inferring vectors")
        for variant in range(len(corpus)):
            if (corpus[variant].words not in conta):
                inferred_vector = model.infer_vector(corpus[variant].words)
                vectors.append(inferred_vector)
                conta.append(corpus[variant].words)
    elif(nameM=='N2VVS'):
        model=gensim.models.KeyedVectors.load('output/'+nameM+str(vectorsize)+'.model')
        print("model vectors")
        for variant in loadXES.get_variant_names(logName+".xes"):
            if(model.wv.__contains__(variant) and variant not in conta):
                model_vector=model.wv.get_vector(variant)
                vectors.append(model_vector)
                conta.append(variant)
    print("done")
```

```
if(clusterType=="KMeans"):
    kclusterer = KMeansClusterer(NUM_CLUSTERS, distance=nltk.cluster.util.cosine_distance, repeats=25)
    assigned_clusters = kclusterer.cluster(vectors, assign_clusters=True)
elif(clusterType=="HierWard"):
    ward = AgglomerativeClustering(n_clusters=NUM_CLUSTERS, linkage='ward').fit(vectors)
    assigned_clusters = ward.labels_
else:
    print(clusterType, " is not a predefined cluster type. Please use 'KMeans' or 'HierWard', or create a definition for ", clusterType)
    return
clusterResult= {}
  if(nameM=='G2VVS'):
      variant_list = loadXES.get_variant_names(logName + ".xes")
      fatte=[]
      for variant in range(len(conta)):
          if(variant_list[variant] not in fatte):
              clusterResult[variant_list[variant]]=assigned_clusters[variant]
              fatte.append(variant_list[variant])
      resultFile= open('output/'+nameM+str(vectorsize)+clusterType+'.csv','w')
      fatte=[]
      for variant in range(len(conta)):
          if(variant_list[variant] not in fatte):
              resultFile.write(variant_list[variant]+','+str(assigned_clusters[variant])+"\n")
              fatte.append(variant_list[variant])
      resultFile.close()
  elif(nameM=='N2VVS'):
      for variant in range(0, len(conta)):
          clusterResult[conta[variant]] = assigned_clusters[variant]
      resultFile = open('output/'+nameM+str(vectorsize)+clusterType +'.csv', 'w')
      for variant in range(0, len(conta)):
          resultFile.write(conta[variant] + ',' + str(assigned_clusters[variant]) + "\n")
      resultFile.close()
  print("done")
```

L'ultimo modulo implementato è MyLog2Vec, che conterrà gli stessi metodi di clustering di Model2Vec.
MyLog2vec, però, non utilizza i modelli prodotti da Node2Vec o Graph2Vec ma analizza il file log in esame, produce un suo modello. Il clustering prodotto, sarà utilizzato come metrica di riferimento per i test dei nostri modelli.

```
def learn(logName,vectorsize):
    documents = loadXES.get_doc_XES_tagged(logName+'.xes')
    model = gensim.models.Doc2Vec(documents, dm = 0, alpha=0.025, vector_size= vectorsize, window=3, min_alpha=0.025, min_count=0)
    nrEpochs= 4
    for epoch in range(nrEpochs):
        if epoch % 2 == 0:
            print ('Now training epoch %s'%epoch)
        model.train(documents,total_examples=len(documents), epochs=nrEpochs)
        model.alpha -= 0.002
        model.min_alpha = model.alpha
    model.save('output/'+'L2VVS'+str(vectorsize) +'.model')

def cluster(logName,vectorsize,clusterType):
    clusterT(logName,vectorsize,clusterType)
    clusterV(logName,vectorsize,clusterType)
```

Testing

Il potere diagnostico di un test è di per sé un concetto multidimensionale, in quanto include la sensibilità, la specificità, il potere predittivo positivo, il potere predittivo negativo e l'accuratezza.
I test effettuati includono:

- NMI, una normalizzazione del punteggio MI per ridimensionare i risultati tra 0 (nessuna informazione reciproca) e 1 (correlazione perfetta). In questa funzione, le informazioni reciproche sono normalizzate da una media generalizzata di H;

- Rand Index, utilizzato per misurare la somiglianza tra 2 cluster di dati; essendo correlato alla precisione. In particolare, verrà utilizzata la versione Adjusted Rand Index che stabilisce una linea di base usando la somiglianza attesa di tutti i confronti a coppie tra cluster raggruppati specificati da un modello casuale; in quanto le premesse del modello di permutazione sono frequentemente violate in molti scenari di clustering, infatti, il numero di cluster o la distribuzione delle dimensioni di tali cluster variano drasticamente, quando, ad esempio, le dimensioni di tali cluster sono dedotte dai dati. Sebbene l'indice Rand possa produrre solo un valore compreso tra 0 e +1, l'indice Rand rettificato può produrre valori negativi se l'indice è inferiore all'indice previsto;

$$AdjustedIndex = \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex}$$

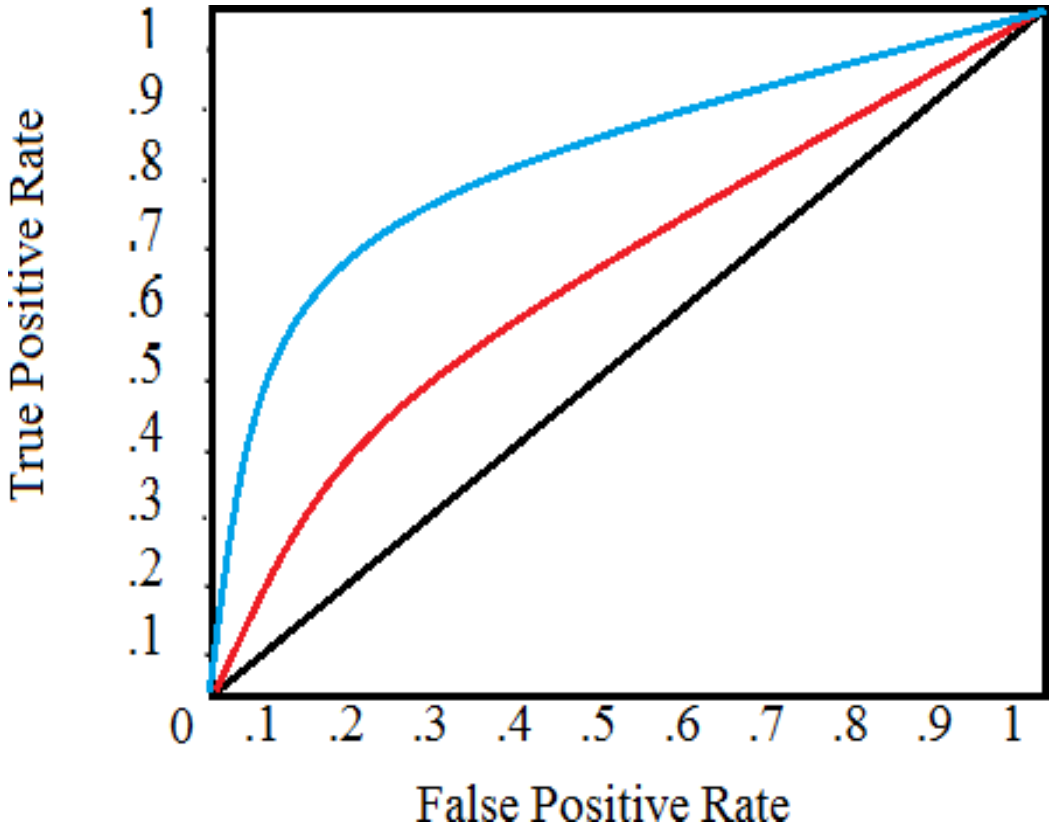
Given the contingency table:

	Y_1	Y_2	$\cdots$	Y_s	$Sums$
X_1	n_{11}	n_{12}	$\cdots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\cdots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\cdots$	n_{rs}	a_r
$Sums$	b_1	b_2	$\cdots$	b_s	

the adjusted index is:

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}$$

- La curva ROC viene costruita considerando tutti i possibili valori del test e, per ognuno di questi, si calcola la proporzione di veri positivi (la sensibilità) e la proporzione di falsi positivi. La proporzione di falsi positivi si calcola con la formula standard: 1 - specificità. Congiungendo i punti che mettono in rapporto la proporzione di veri positivi e di falsi positivi (le cosiddette coordinate) si ottiene una curva chiamata curva ROC. L'area sottostante alla curva ROC (AUC, acronimo dei termini inglesi "Area Under the Curve") è una misura di accuratezza diagnostica. L'area sotto la curva può assumere valori compresi tra 0.5 e 1.0. Tanto maggiore è l'area sotto la curva (cioè tanto più la curva si avvicina al vertice del grafico) tanto maggiore è il potere discriminante del test.



- AUC score,equivale a calcolare la correlazione tra predizioni e target. Dal punto di vista dell'interpretazione, è utile poiché mostra quanto il modello è bravo a classificare le previsioni. In particolare, mostra qual è la probabilità che un'istanza positiva scelta casualmente sia classificata più in alto di un'istanza negativa scelta casualmente. La metrica è utilizzata quanto si vuole dare lo stesso peso ai falsi positivi ed ai falsi negativi (escludendo quindi un'analisi con F1 score che dà più peso alla classe positiva).

Clustering utilizzati

Le tipologie di algoritmi di clustering utilizzate sono due:

1) Il K-Means è un algoritmo di apprendimento non supervisionato che trova un numero fisso di cluster in un insieme di dati. Ognuno di questi cluster raggruppa un particolare insieme di oggetti, che vengono definiti data points, in base a certe caratteristiche che ne determinano la somiglianza. Quando si utilizza un algoritmo K-Means, per ogni cluster si definisce un centroide, ossia un punto (immaginario o reale) al centro di un cluster. L'algoritmo è iterativo, ossia che esegue ripetutamente alcune sue fasi e fondamentalmente si può affermare che è formato dai seguenti step:

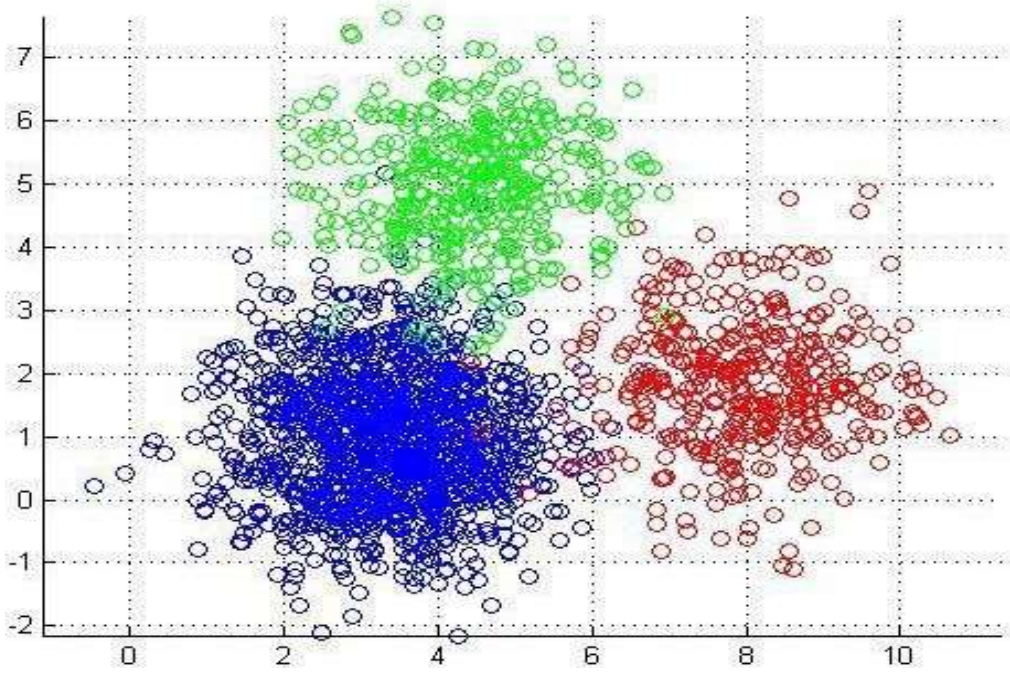
- Inizializzazione: si definiscono i parametri di input per eseguire l'algoritmo;
- Assegnazione del cluster: ogni data points viene assegnato al cluster (o centroide) più vicino;
- Aggiornamento della posizione del centroide: ricalcola il punto esatto del centroide e di conseguenza ne modifica la sua posizione.

VANTAGGI:

- abbastanza veloce, pochi calcoli e poco tempo di elaborazione per calcolare le distanze tra i data points e i centroidi ad ogni iterazione.

SVANTAGGI:

- Bisogna selezionare quanti gruppi si vogliono visualizzare, non sempre banale;
- K-means inizia con una scelta casuale di centroidi e pertanto può produrre risultati di clustering diversi su diverse sequenze dell'algoritmo. I risultati potrebbero non essere ripetibili e mancare di coerenza.



2) HierWard è il clustering gerarchico agglomerativo di Ward. Utilizza un approccio "bottom up" (dal basso verso l'alto) in cui si parte dall'inserimento di ciascun elemento in un cluster differente e si procede quindi all'accorpamento graduale di cluster a due a due. Il metodo utilizza, per calcolare le distanze tra i punti, la somma delle differenze quadrate all'interno dei cluster. E' un approccio che minimizza la varianza, simile alla funzione obiettivo di k-means affrontata con un approccio gerarchico diverso.

PREMESSE

- Nei modelli generati, per ogni algoritmo implementato, la dimensione dei vettori utilizzati per rappresentare le informazioni del modello, varia da 16 a 128 bit;

- I risultati di KMeans e HierWard in base al modello("traccia-attività" oppure "variante traccia-attività+attributi") con Node2Vec o Graph2Vec, sono stati salvati in file di tipo .csv, contenuti insieme ai file .model nella cartella output;

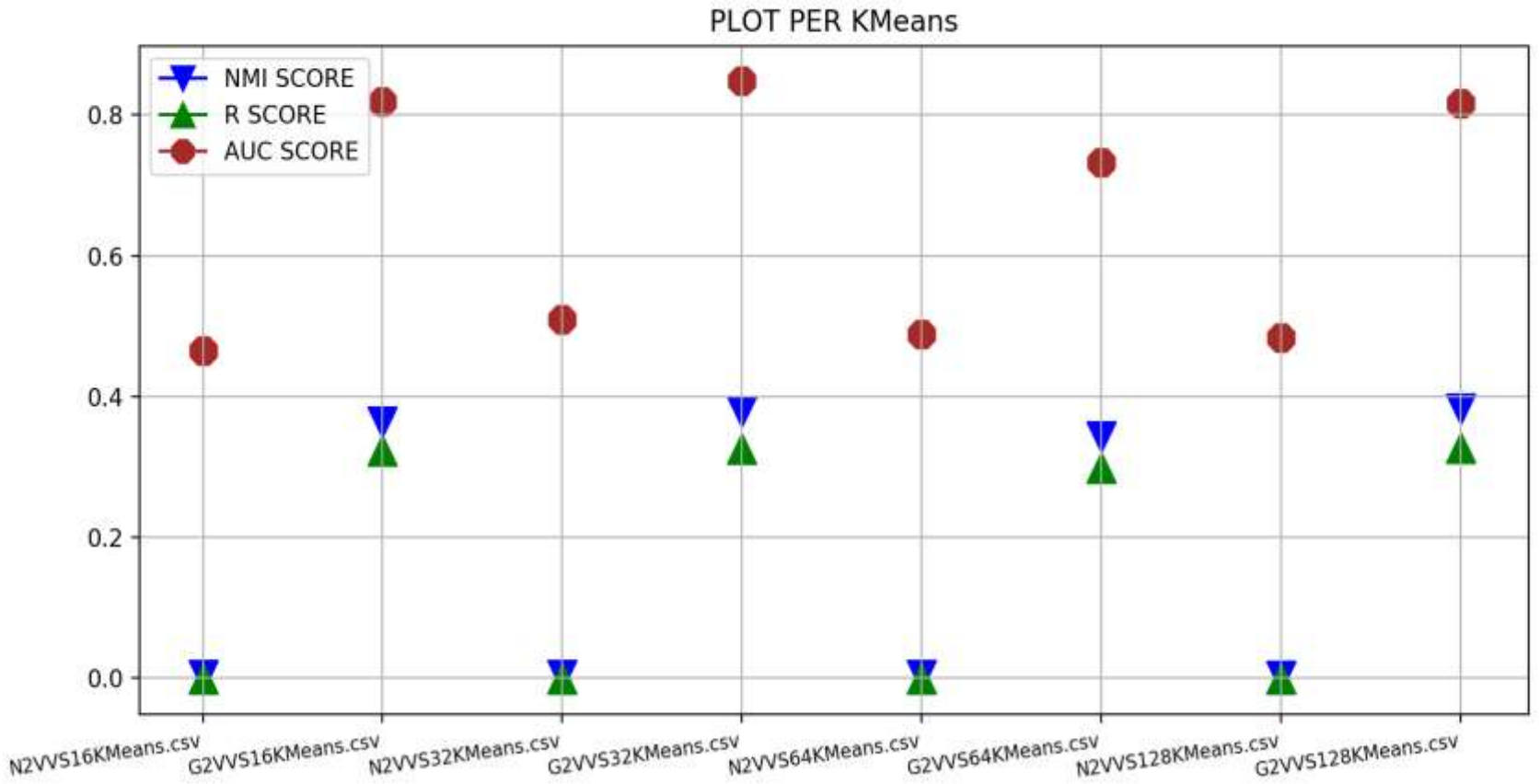
- Le analisi effettuate sui clustering, sono salvate nel file risultati.xlsx in opportune tabelle, di facile lettura ed interpretazione;

- La metrica di riferimento per l'applicazione dei test, è rappresentata dai risultati ottenuti da MyLog2Vec, in grado di analizzare al meglio il file log in esame.

«VariantID-Attività+attributi»

Tabella N2VVSKMeans			
	NMI	RI	AUC
N16	0,004	-0,0007	0,49
N32	0,0003	-0,0009	0,5
N64	0,003	-0,0007	0,507
N128	0,003	-0,0006	0,5

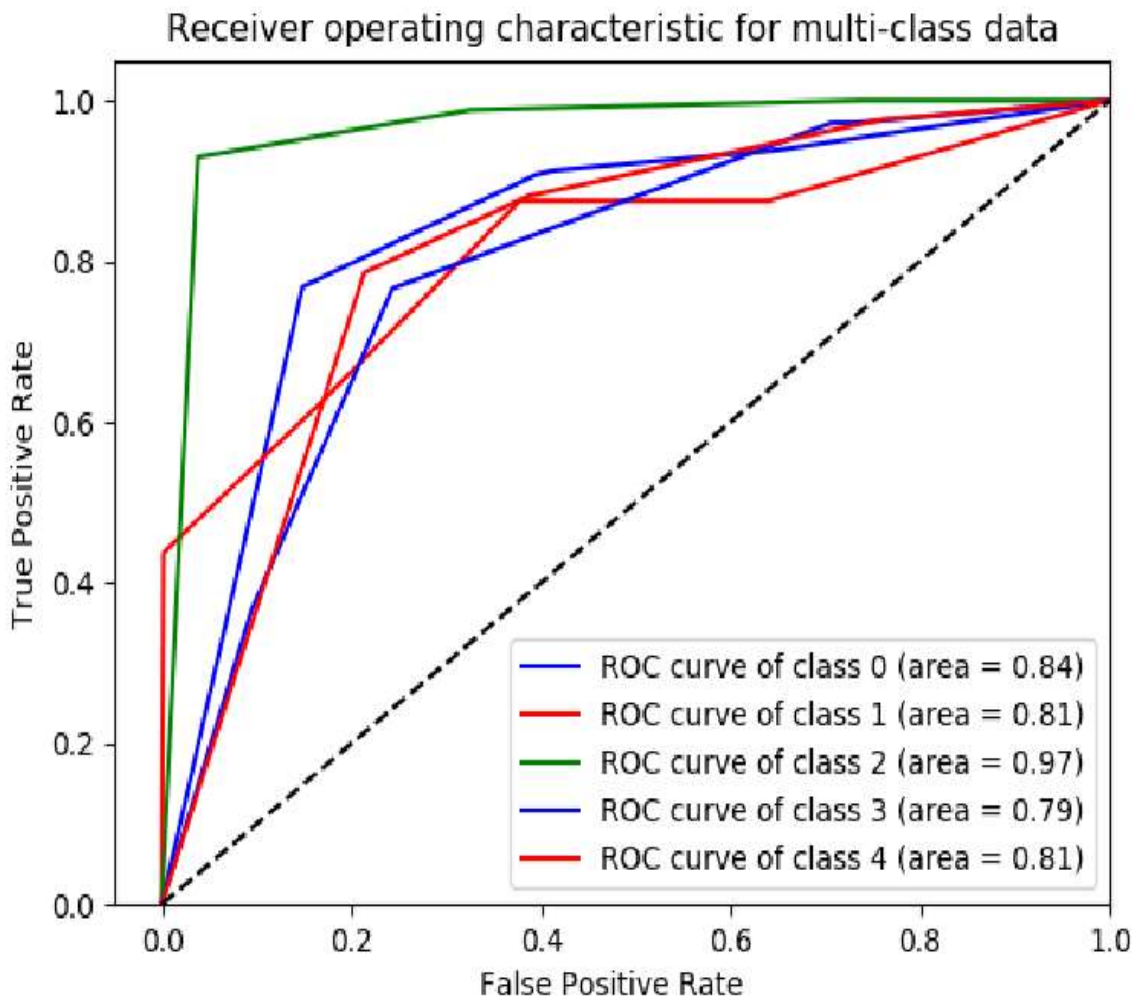
Tabella G2VVSKMeans			
	NMI	RI	AUC
G16	0,36	0,32	0,85
G32	0,38	0,33	0,87
G64	0,35	0,3	0,78
G128	0,38	0,32	0,82



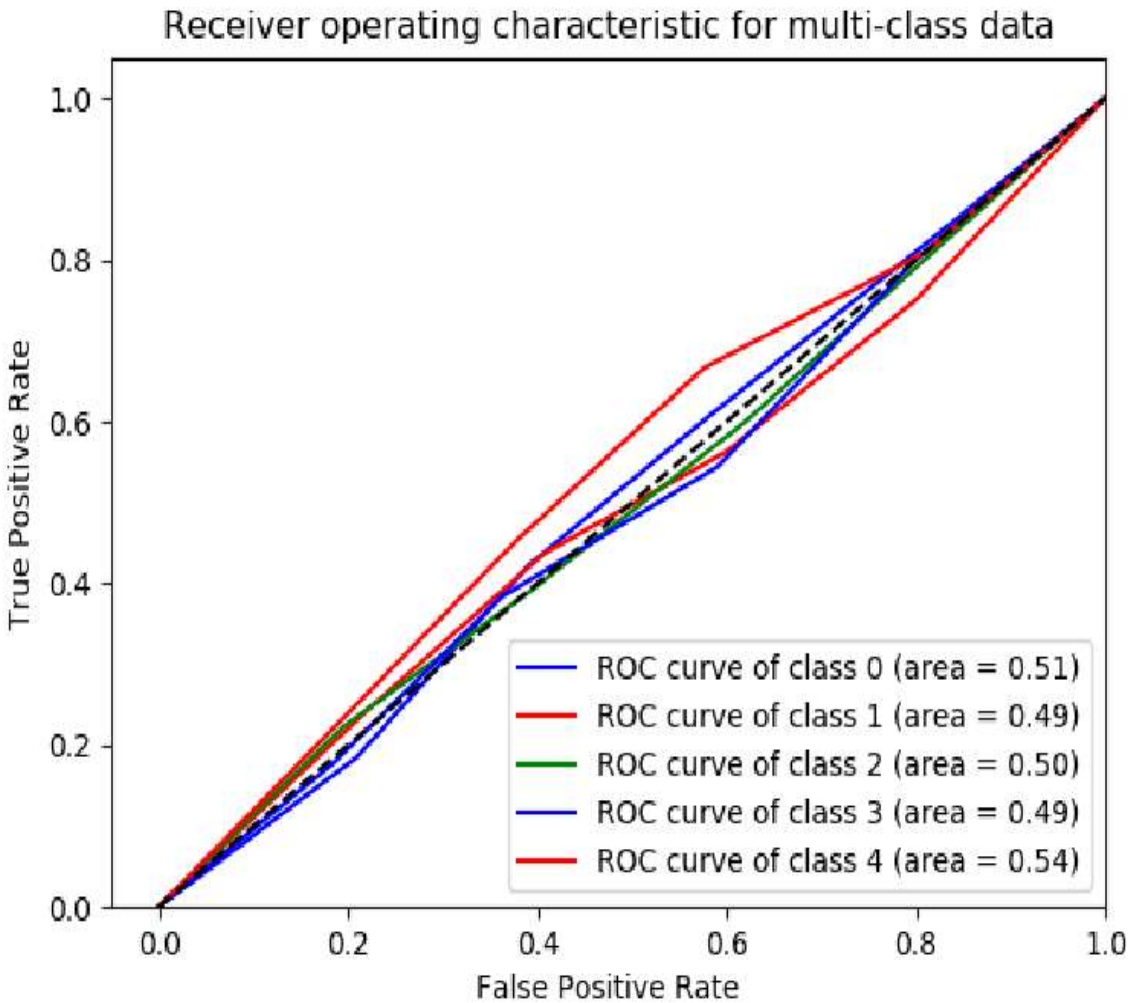
Analizzando i risultati ottenuti da K-Means, è evidente la migliore capacità di Graph2Vec nell'ottenere buoni risultati. L'indice NMI nella configurazione "variante traccia-attività+attributi" ha un range di 0,36<=x<=0,38 contro i 0,0003<=y<=0,003, così come quello RI 0,30<=x<=0,33 contro i -0,0009<=y<=-0,0006, valore negativo che indica un indice diverso da quello atteso per Node2Vec.

Come ultima conferma, si ottengono valori dell'area della curva di ROC per Graph2Vec compresi 0,78<=x<=0,87 contro i 0,49<=y<=0,507 di Node2Vec; indicando l'impossibilità del test di decidere per quest'ultimo modello, contro una buona accuratezza del primo. Ricapitolando, per Node2Vec il test non è informativo, mentre per Graph2Vec è un test accurato.

Graph2vec



Node2Vec

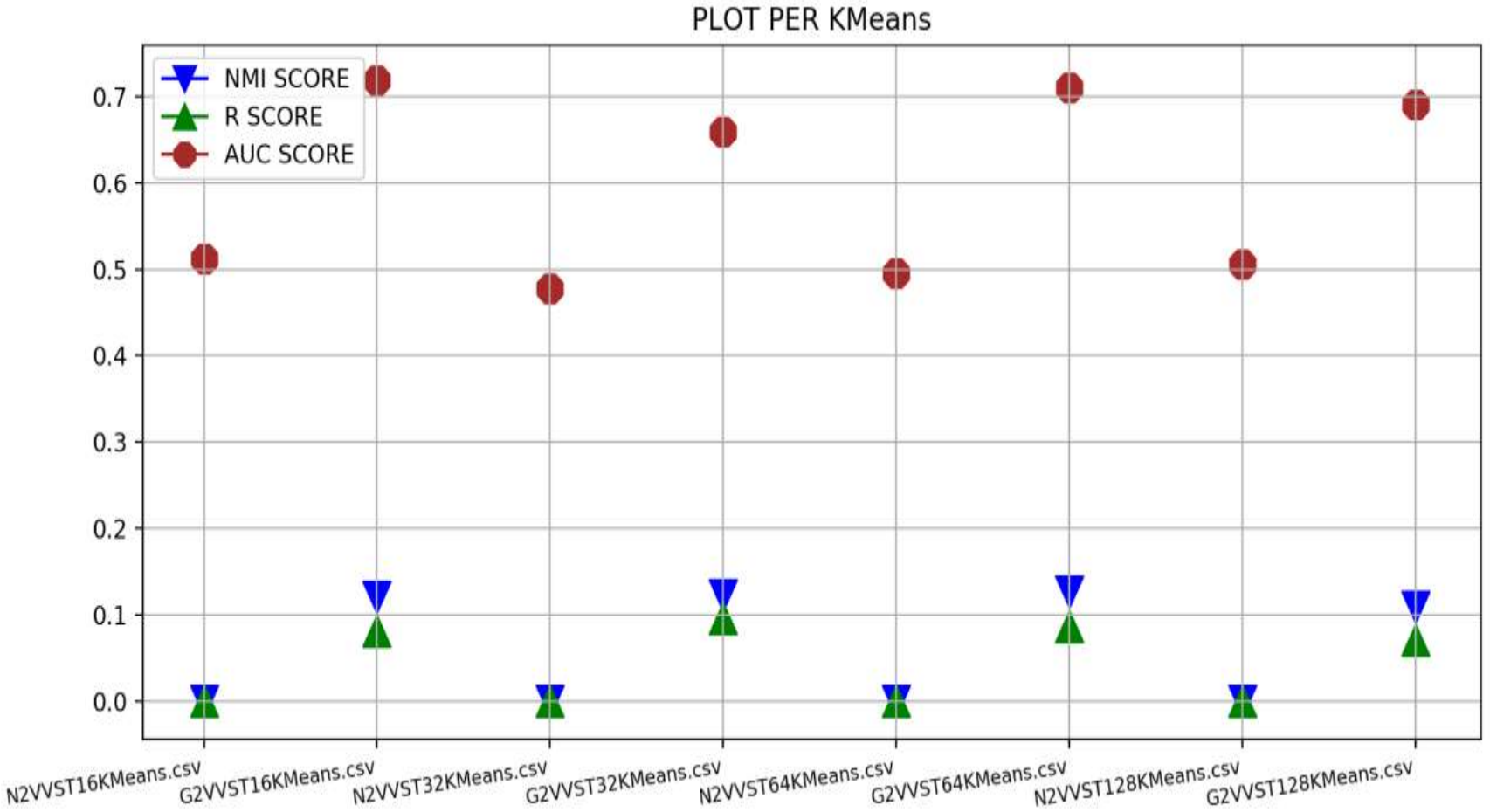


Per Node2Vec, il test della curva di ROC non è informativo, mentre per Graph2Vec è un test moderatamente accurato.

«ID Tracce-attività»

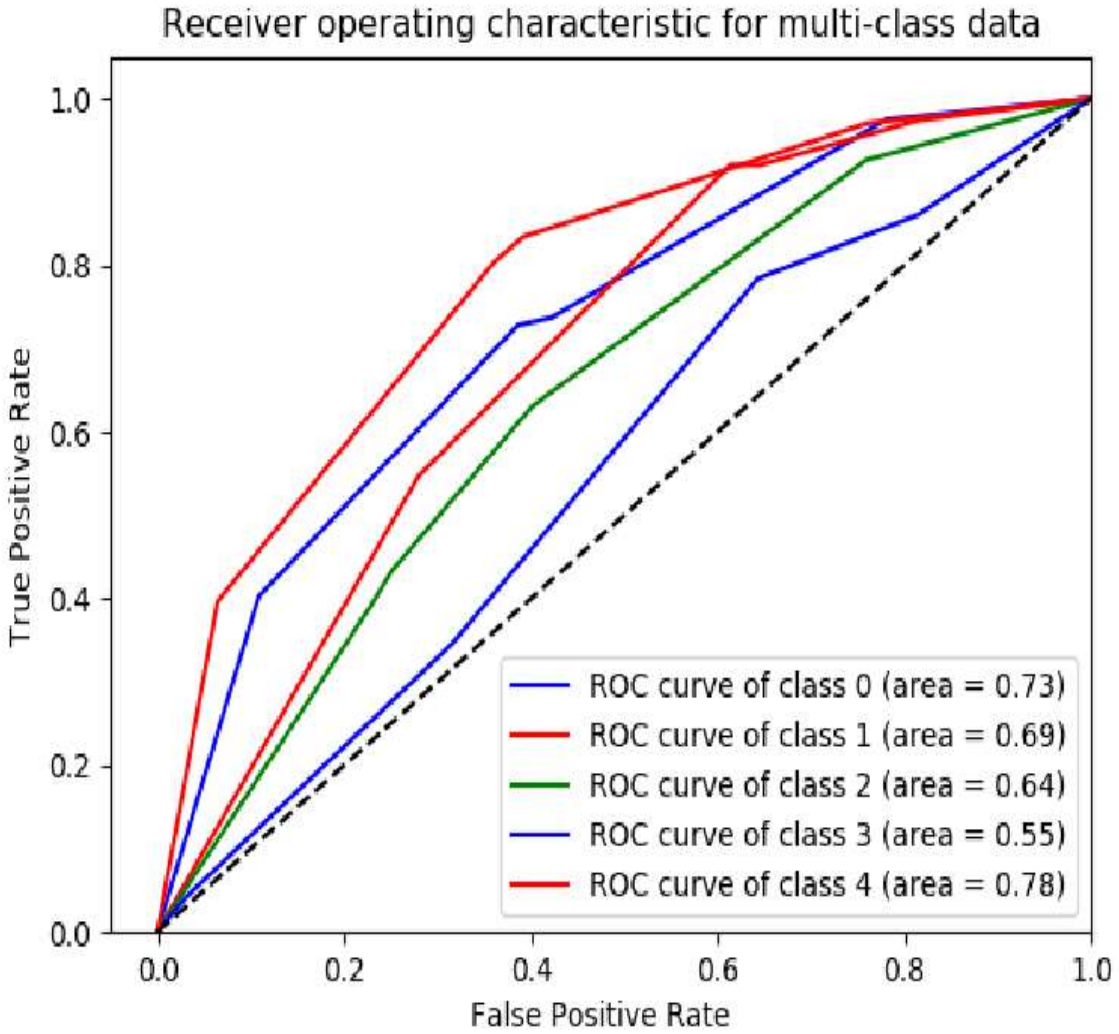
Tabella N2VVSTKMeans			
	NMI	RI	AUC
N16	0,001	0,000087	0,51
N32	0,001	0,0001	0,46
N64	0,0009	0,00008	0,49
N128	0,0011	0,0002	0,5

Tabella G2VVSTKMeans			
	NMI	RI	AUC
G16	0,12	0,08	0,67
G32	0,12	0,09	0,69
G64	0,12	0,08	0,67
G128	0,11	0,08	0,67

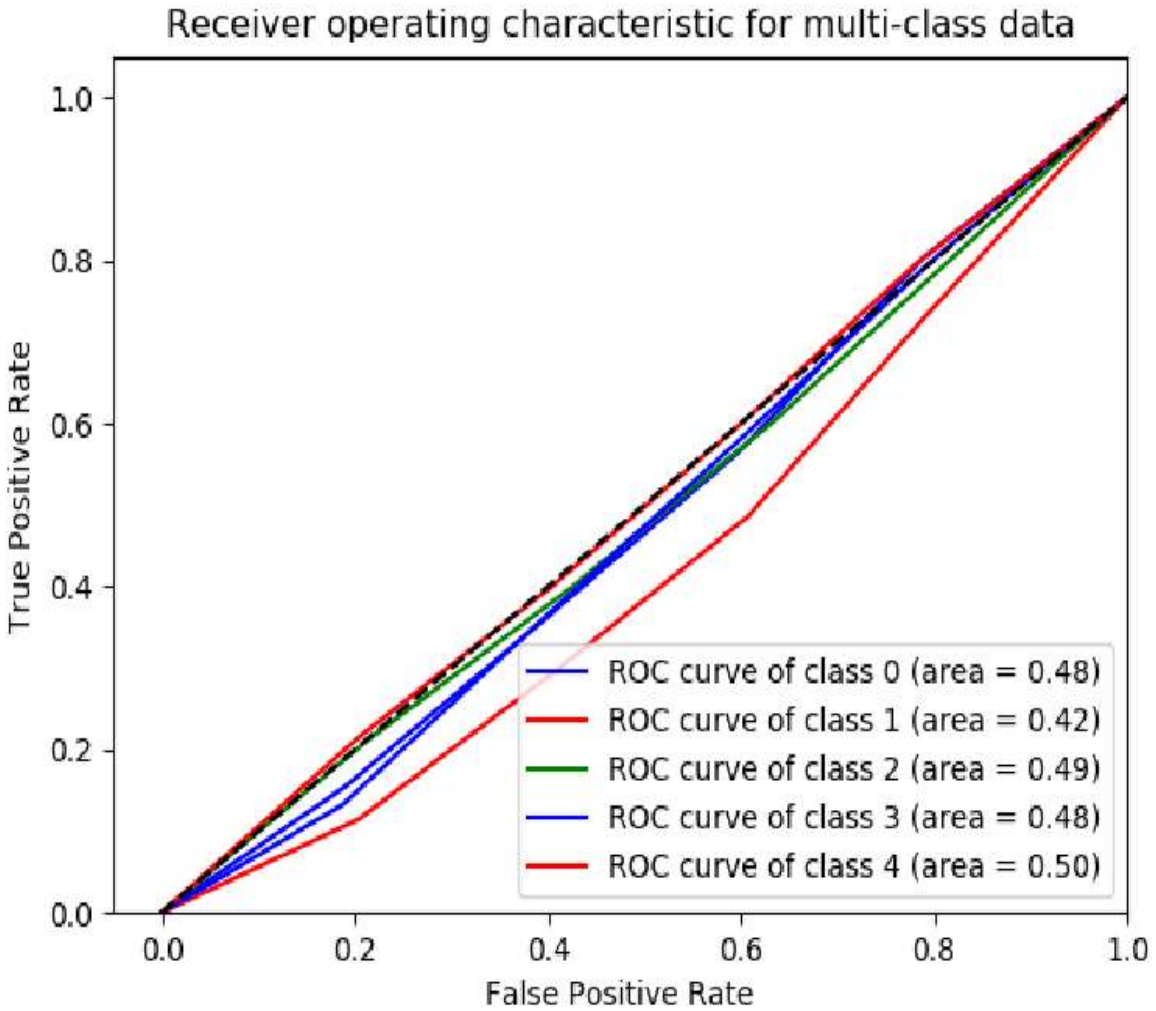


Anche qui Graph2Vec si comporta meglio, nonostante i valori NMI, RI e dell'auc_score siano più bassi.
Il valore di RI qui non è negativo (non ho un valore diverso dall'atteso) per Node2Vec, ma di gran lunga inferiore rispetto a quello di Graph2Vec.

Graph2vec

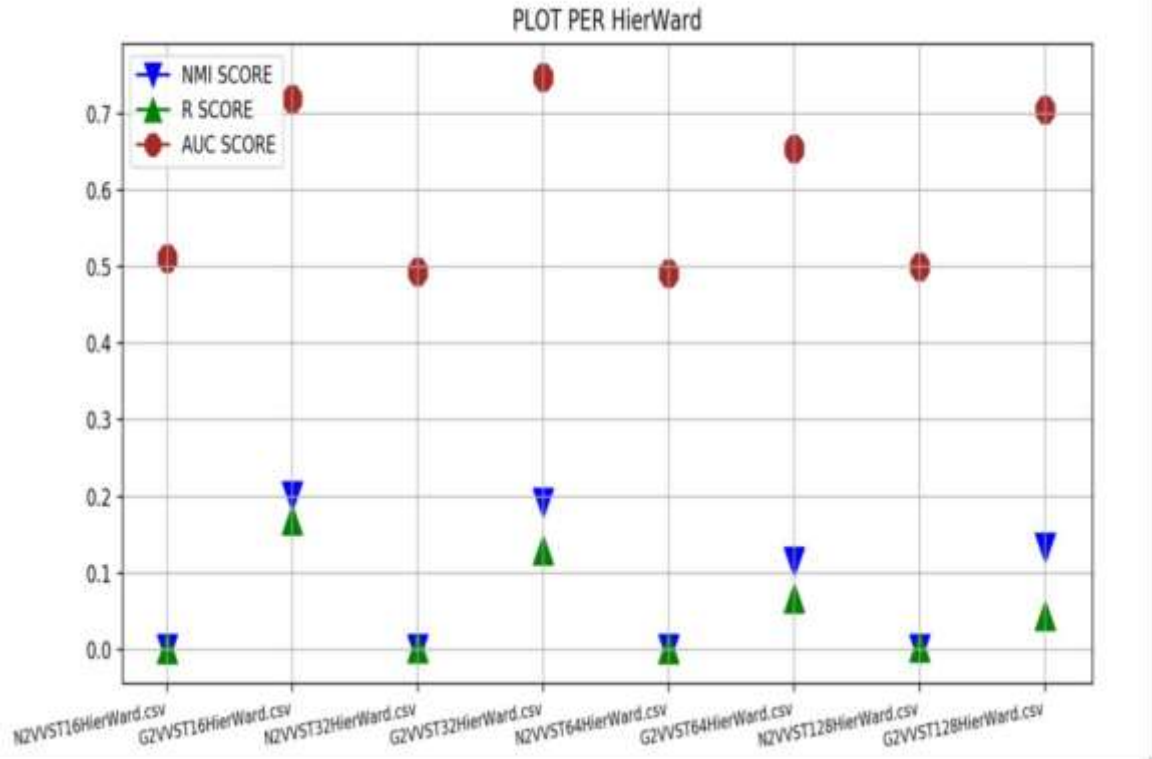


Node2Vec



L'analisi delle curve di ROC, con i risultati AUC tabellati, mostrano chiaramente che questo test è nuovamente non informativo per Node2Vec; mentre è poco accurato per Graph2Vec, producendo comunque un risultato migliore rispetto all'algoritmo precedente.

Passando alla tipologia di clustering gerarchico HierWard, si ottiene la seguente tabella, ricalcano la falsa riga di Kmeans e non necessitando di ulteriori spiegazioni.



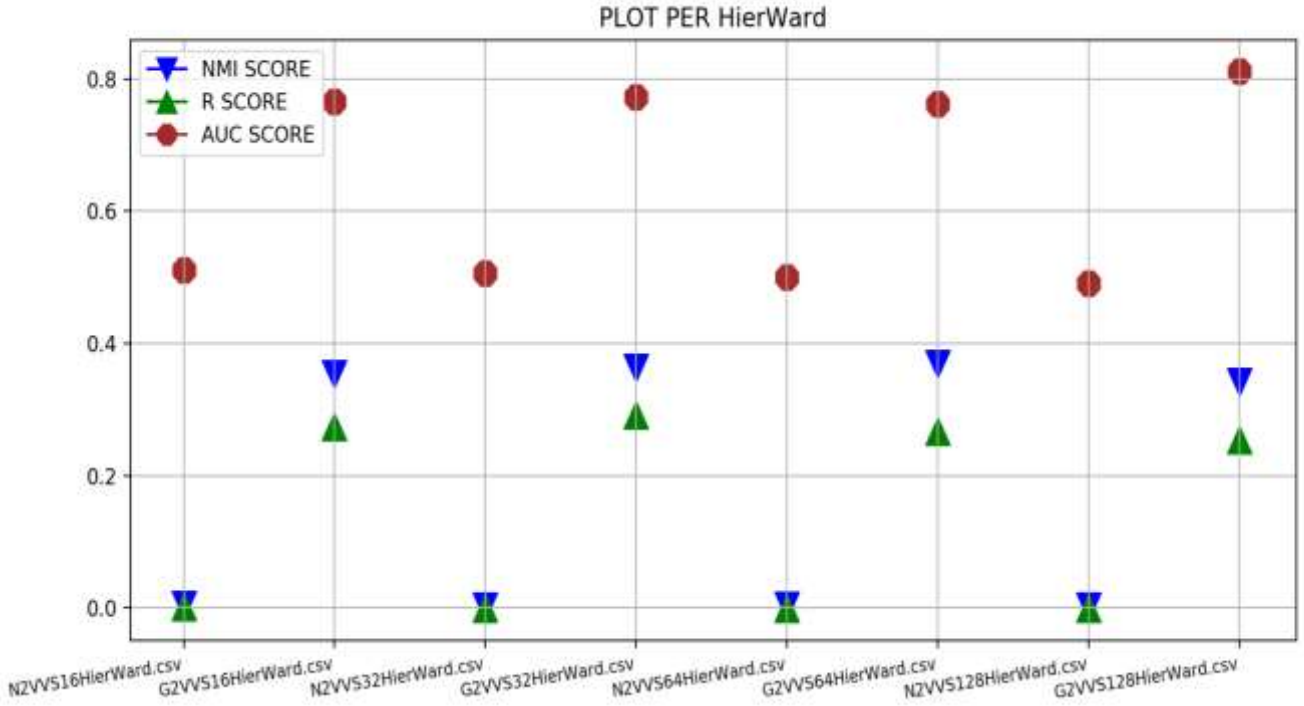


Tabella N2VVSHW

	NMI	RI	AUC
N16	0,005	0,001	0,6
N32	0,003	-0,003	0,49
N64	0,003	-0,001	0,5
N128	0,002	-0,005	0,48

Tabella N2VVSTHW

	NMI	RI	AUC
N16	0,001	0,001	0,51
N32	0,0009	0,001	0,49
N64	0,0008	0,0004	0,49
N128	0,001	0,003	0,51

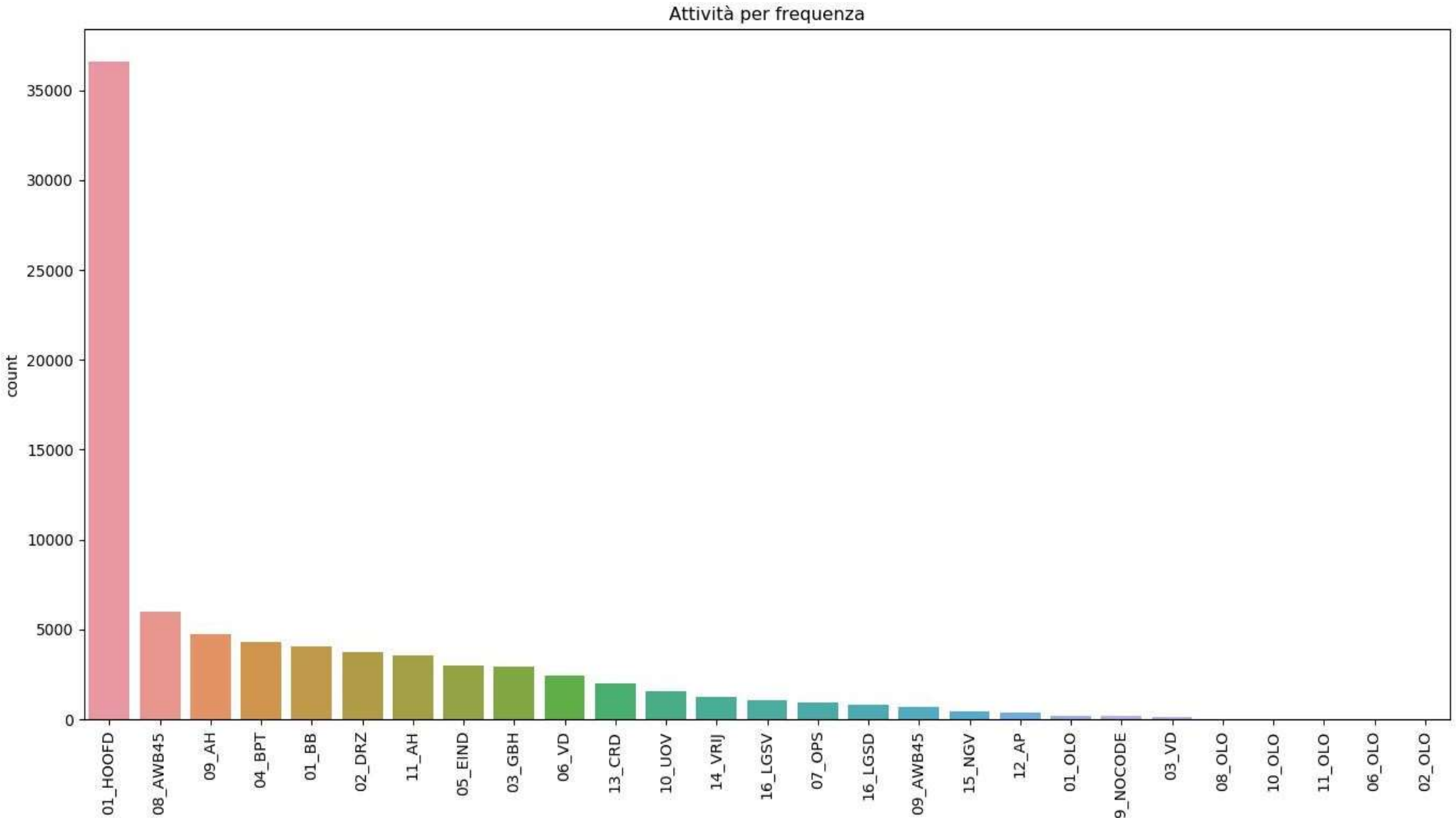
Tabella G2VVSHW

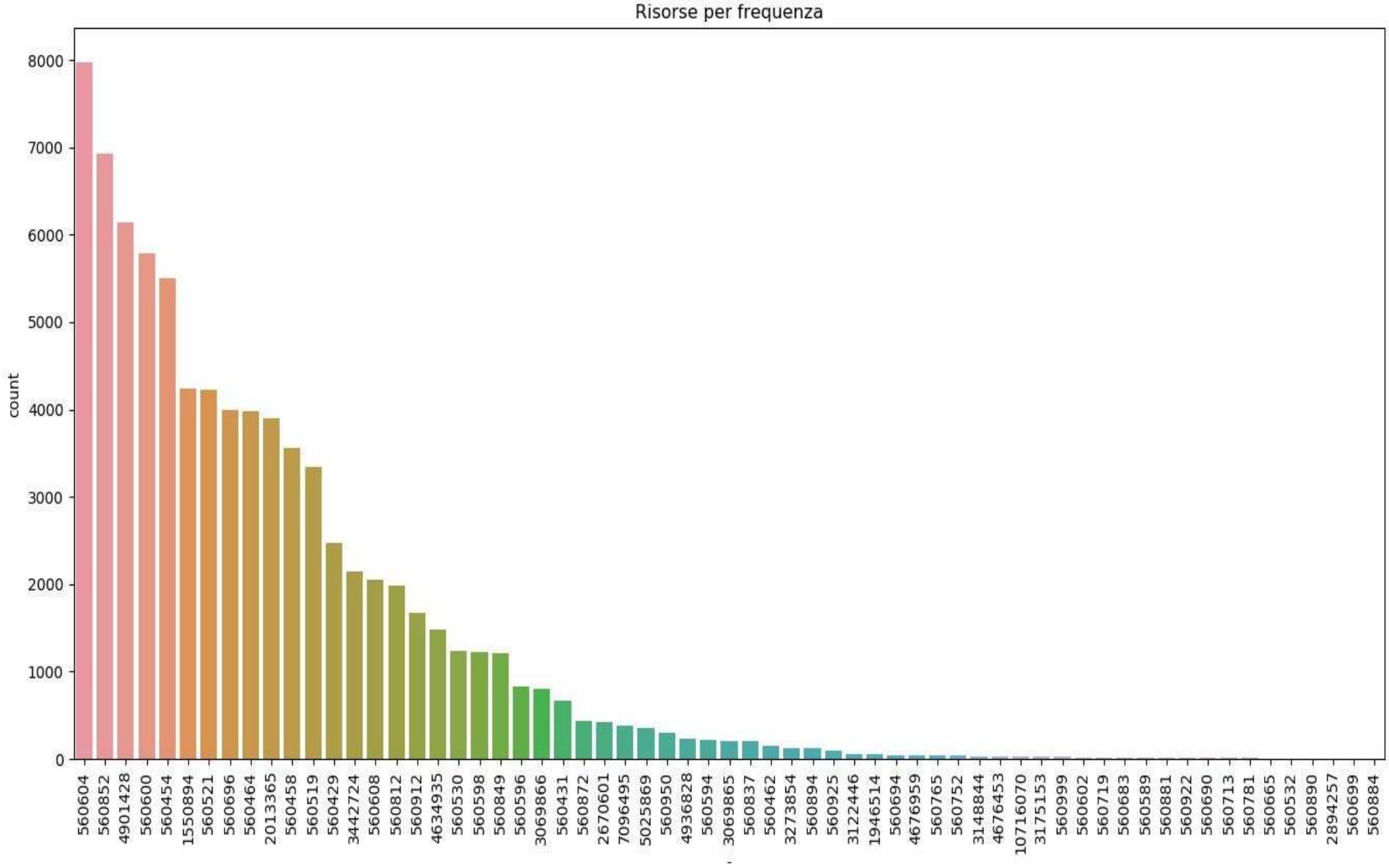
	NMI	RI	AUC
G16	0,35	0,27	0,85
G32	0,36	0,29	0,82
G64	0,37	0,28	0,83
G128	0,34	0,25	0,84

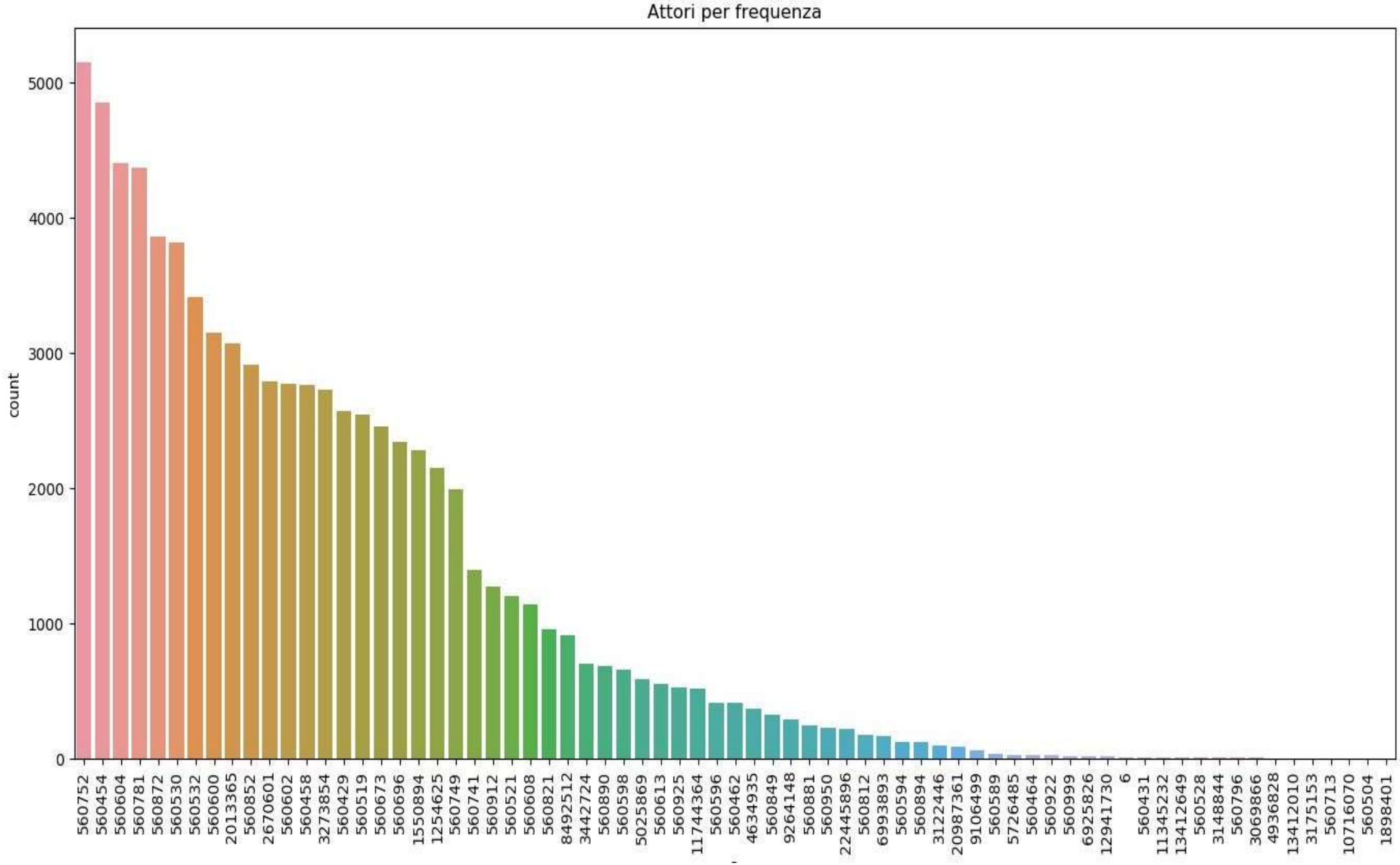
Tabella G2VVSTKHW

	NMI	RI	AUC
G16	0,2	0,17	0,73
G32	0,19	0,13	0,73
G64	0,11	0,07	0,67
G128	0,13	0,04	0,73

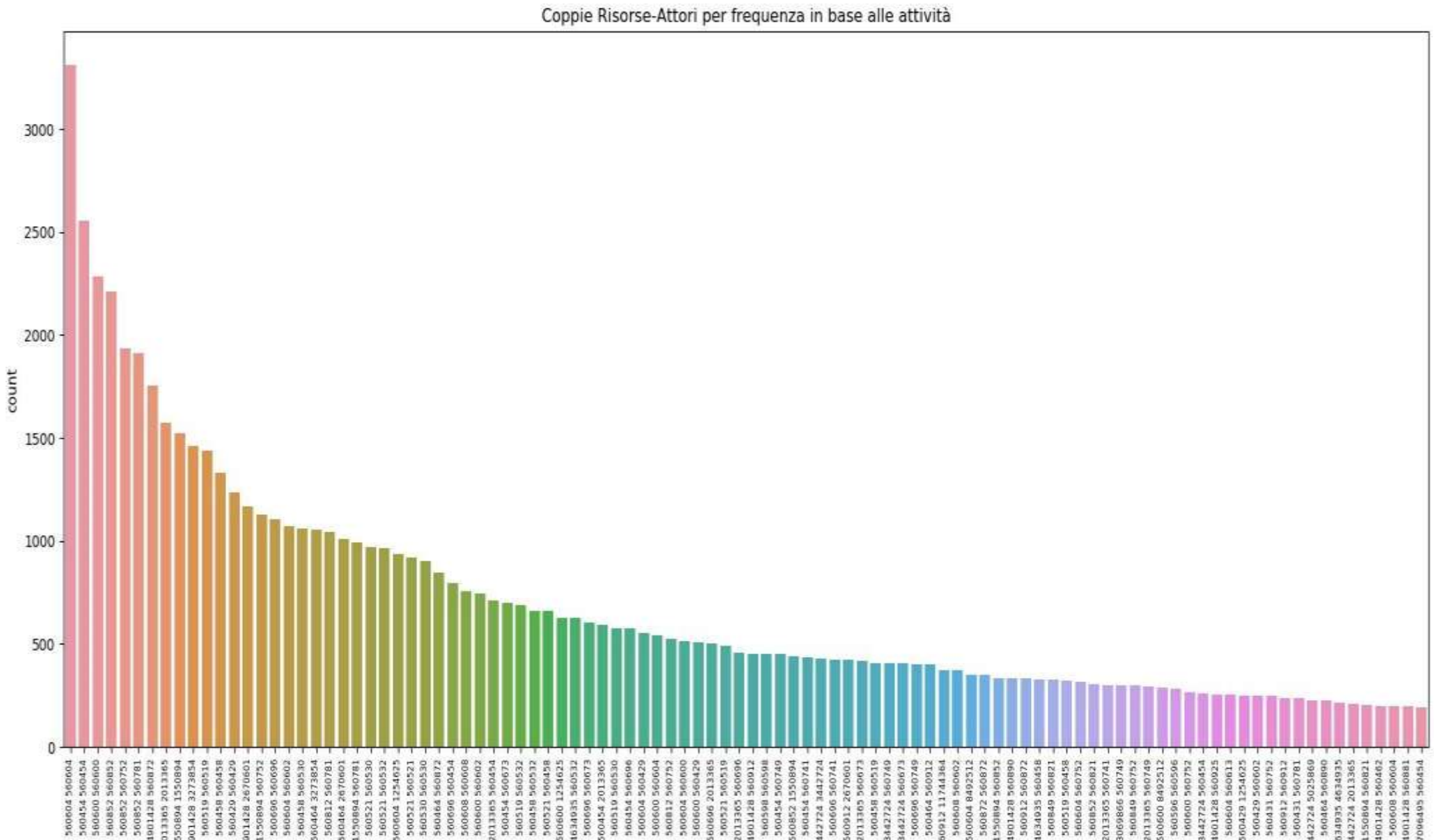
La semplice ma efficace struttura di Graph2Vec, permette di effettuare una valutazione su gli attributi più significativi del file log in esame:




Risorse per frequenza
count
8000
7000
6000
5000
4000
3000
2000
1000
0
560604
560852
4901428
560600
560454
1550894
560521
560696
560464
2013365
560458
560519
560429
3442724
560608
560812
560912
4634935
560530
560598
560849
560596
3069866
560431
560872
2670601
7096495
5025869
560950
4936828
560594
3069865
560837
560462
3273854
560894
560925
3122446
1946514
560694
4676959
560765
560752
3148844
4676453
10716070
3175153
560999
560602
560719
560683
560589
560881
560922
560690
560713
560781
560665
560532
560890
2894257
560699
560884


Attori per frequenza
count
0
1000
2000
3000
4000
5000
560752
560454
560604
560781
560872
560530
560532
560600
2013365
560852
2670601
560602
560458
3273854
560429
560519
560673
560696
1550894
1254625
560749
560741
560912
560521
560608
560821
8492512
3442724
560890
560598
5025869
560613
560925
11744364
560596
560462
4634935
560849
9264148
560881
560950
22445896
560812
6993893
560594
560894
3122446
20987361
9106499
560589
5726485
560464
560922
560999
6925826
12941730
6
560431
11345232
13412649
560528
3148844
560796
3069866
4936828
13412010
3175153
560713
10716070
560504
1898401

Permettendo, inoltre, di studiare la correlazione tra attributi:



Possiamo concludere che la nostra architettura ideale per l'analisi sui grafi deve utilizzare, per la formazione dei modelli, Graph2Vec nettamente migliore di Node2Vec.