# Crédit Agricole Italia - Analisi sugli Instant Feedback

January 20, 2022

Marta Caffagnini

Matricola: 168007

307930@studenti.unimore.it

INDICE

1. Introduzione al problema e alle caratteristiche dei dati ……1
2. Descrizione approccio risolutivo adottato……4
3. Text analytics……5
   3.1 Feature Extraction……13
   3.2 TfidfVectorizer……13
   3.3 N-grammi……13
   3.4 Feature Selection……14
   3.5 Addestramento……14
4. Graph Analytics……19
   4.1 Importazione dati su Neo4j……19
   4.2 Obiettivo 1……24
   4.3 Obiettivo 2……34
5. Conclusioni……41

## 1 Introduzione al problema e alle caratteristiche dei dati

Per migliorare la "Customer Experience" si analizzano le risposte ai questionari Instant Feedback mediante App Crédit Agricole Italia e la rete delle relazioni tra clienti, conti correnti e filiali. Queste analisi hanno l'obiettivo di quantificare il grado di soddisfazione dei clienti calcolando uno score per ogni utente e la loro capacità di influenza. Infine vengono individuate le filiali che si distinguono per un maggiore gradimento negativo, o positivo.

Per operare questa analisi sono stati forniti due dataset:

- Dataset_1 contiene le risposte fornite dai clienti alle domande dei questionari. Ogni cliente ha il suo codice identificativo univoco (ID_CLIENTE). Ogni questionario ha il

suo ID (ID_QUESTIONARIO) e il suo nome (DESC_QUESTIONARIO). Ogni questionario ha le sue specifiche domande con ID univoco (ID_DOMANDA) e testo della domanda (DESC_DOMANDA). Le risposte fornite da un cliente a un certo questionario in una certa data (DATA_COMPILAZIONE) hanno lo stesso codice ID di sessione (ID_SESSIONE_QUESTIONARIO). Il campo TIPO_RISPOSTA indica il tipo di risposta alla domanda. Se la risposta è chiusa a scelta singola il valore di TIPO_RISPOSTA sarà inputradio o inputmatrix; se è chiusa a scelta multipla sarà inputmulticheckb; se è una data inputdate; se aperta, cioè input testuale libero, il valore sarà inputtextarea. 107388 sono gli utenti che hanno risposto ad almeno un questionario. 23 sono i diversi questionari presenti nel dataset. 805943 è il numero di risposte totali.

[1]:
```
#importazione delle librerie necessarie
import pandas as pd
#inseriamo il primo dataset in un dataframe con dtype specifichiamo i tipi dei␣
↪valori di ogni colonna
dataset = pd.read_csv('Dataset1_Risposte_Questionari.txt', sep="\t",␣
↪header=0,dtype = {'ID_CLIENTE': int,
                                                                              ␣
↪       'ID_QUESTIONARIO': int,
                                                                              ␣
↪       'DESC_QUESTIONARIO': str,
                                                                              ␣
↪       'ID_SESSIONE_QUESTIONARIO': int,
                                                                              ␣
↪       'DATA_COMPILAZIONE': str,
                                                                              ␣
↪       'ID_DOMANDA': int,
                                                                              ␣
↪       'DESC_DOMANDA': str,
                                                                              ␣
↪       'TIPO_RISPOSTA': str,
                                                                              ␣
↪       'DESC_RISPOSTA': str})
dataset.head(3) #mostriamo le prime righe del dataset_1
```

[1]:
```
   ID_CLIENTE  ID_QUESTIONARIO   DESC_QUESTIONARIO  ID_SESSIONE_QUESTIONARIO  \
0     6149323               17  Apertura cc online                    534695
1     6149323               17  Apertura cc online                    534695
2     6149323               17  Apertura cc online                    534695

  DATA_COMPILAZIONE  ID_DOMANDA  \
0        2021-02-03        5275
1        2021-02-03        5276
2        2021-02-03        5277

                                      DESC_DOMANDA TIPO_RISPOSTA  \
0  Come sei venuto a conoscenza di Conto Crédit A…    inputradio
```

```
1  Quanto sei soddisfatto della tua esperienza di…    inputradio
2  Come valuti il tempo impiegato per il processo…    inputradio

                DESC_RISPOSTA
0  in-filiale-Cr-dit-agricole
1                   Abbastanza
2                  soddisfacen
```

- Dataset_2 contiene i collegamenti tra coppie di clienti (ID_CLIENTE_1, ID_CLIENTE_2). Ciascun cliente ha un campo in cui è indicato il suo id, la sua Natura Giuridica (PF = persona fisica, COI=cointestazione, DI=Ditta Individuale), e l'ID della sua filiale (ID_FILIALE). La tipologia dei collegamenti tra due clienti è individuata dal campo COD_COLLEGAMENTO e descritta da DESC_COLLEGAMENTO. I clienti sono 3505346 e le relazioni che intercorrono sono in totale 4089536.

[2]:
```
#inseriamo il secondo dataset in un dataframe
dataset2 = pd.read_csv('Dataset2_Collegamenti_Clienti.txt', sep="\t",␣
↪header=0,dtype = {'ID_CLIENTE': int,
↪                  'COD_NATURA_GIURIDICA_1': str,
↪                  'ID_FILIALE_1': int,
↪                  'COD_COLLEGAMENTO': str,
↪                  'DESC_COLLEGAMENTO': str,
↪                  'ID_CLIENTE_2': int,
↪                  'COD_NATURA_GIURIDICA_2': str,
↪                  'ID_FILIALE_2': int})
#specifichiamo il tipo di ogni campo perchè nel dataset valori della stessa␣
#colonna sono di tipo diverso
dataset2.head(3) #mostriamo le prime righe del dataset_2
```

[2]:
```
   ID_CLIENTE COD_NATURA_GIURIDICA_1  ID_FILIALE_1 COD_COLLEGAMENTO  \
0     6123972                    DI         28438              R01
1     6123981                    DI         28638              R01
2     6123989                    PF         45955              I85

   DESC_COLLEGAMENTO  ID_CLIENTE_2 COD_NATURA_GIURIDICA_2  ID_FILIALE_2
0  TIT. DITTA INDIV.       6482323                    PF         28438
1  TIT. DITTA INDIV.       6173394                    PF         28638
2      HA COME EREDE       6756182                    PF         45955
```

## 2 Descrizione approccio risolutivo adottato

Parte 1

Per calcolare lo score di ogni cliente viene effettuata l'analisi di risposte chiuse e risposte aperte contenute nel Dataset_1. Dopo aver eliminato le domande non rilevanti per il calcolo di soddisfazione, a ogni risposta chiusa viene associato un certo score definito in modo statico e arbitrario. Verrà così aggiunta una colonna al dataset con lo score corrispondente a ogni riga, cioè a ogni risposta. Le categorie di classificazione saranno 4: -1.0, -0.5, 0.5, 1. Per quanto riguarda le risposte aperte viene assegnato uno score facendo la media degli score delle risposte chiuse dello stesso questionario del singolo cliente. Poi viene proposta una Sentiment Analysis creando un modello di Machine Learning per classificare le risposte come feedback negativi o positivi. Per l'addestramento del modello i target utilizzati sono gli score calcolati basandosi sulle risposte chiuse.

Prima di creare il modello viene applicata una fase di preprocessing sulle risposte aperte per aiutare il modello ad apprendere meglio dai dati. Questo preprocessing comprende il tokenizing del testo, eliminazione della punteggiatura, delle stopword e delle parole grammaticalmente errate.

I modelli creati sono basati sul classificatore Bernoulli Naive Bayes, oppure il classificatore multinomiale Naive Bayes, o Complement Naive Bayes, infine Random forest. Una volta addestrati e validati i modelli, viene individuato quello con l'accuratezza migliore.

Ora che si hanno gli score per ogni risposta contenuta nel dataset si può calcolare lo score di soddisfazione di ciascun cliente. Prima si raggruppano le risposte date da ciascun cliente basandosi sul suo ID e poi si calcola la media degli score. A ogni cliente ora è associato uno score di soddisfazione. Ordinando i clienti in base al loro score si possono individuare i clienti più soddisfatti e i più insoddisfatti.

Parte 2

Vengono preparati i file csv per importare i dati in un grafo Neo4j. Per calcolare il sottografo di un cliente viene utilizzato l'algoritmo Dijkstra Single-Source, algoritmo di path finding. Questo algoritmo trova i nodi connessi al nodo sorgente e il percorso più breve per raggiungerli. Per calcolare il grado di influenza di ogni nodo del sottografo si prende lo score del nodo sorgente e lo si divide per la lunghezza del percorso che collega il cliente al nodo sorgente. Il grado di influenza prende il segno dallo score del cliente influenzato dal nodo sorgente.

L'approccio alternativo proposto prevede l'utilizzo dell'algoritmo WCC (Weakly Connected Components). Questo algoritmo divide i nodi in gruppi. Nel calcolo del grado di influenza viene sempre usato l'algoritmo Dijkstra Single-Source ma come lunghezza del percorso non viene considerato il numero di archi che separano due nodi, ma il costo delle relazioni che devono essere attraversate. Dopo aver individuato i clienti più soddisfatti e meno soddisfatti in base al loro score, viene calcolato il loro sottografo di influenza.

Per ogni nodo con uno score viene calcolato il suo sottografo di clienti e sui nodi influenzati si scrive il grado di influenza. Se un cliente è influenzato da più nodi, le influenze vengono sommate e poi divise per il numero di nodi da cui è stato influenzato. Per avere un numero che rappresenta la soddisfazione del cliente, si somma lo score al grado di influenza. Questo viene riportato nell'intervallo -1, 1. Ora si ha a disposizione il grado di soddisfazione dei clienti. Per calcolare il numero di clienti di ogni filiale che hanno la property soddisfazione, si utilizza l'algoritmo Degree Centrality che calcola gli archi uscenti da ogni nodo. Per calcolare il livello di soddisfazione di ogni filiale viene fatta una media di soddisfazione considerando tutti i clienti che ne hanno espressa una.

Dopo aver assegnato un valore di soddisfazione a ogni filiale vengono individuate le 3 più virtuose e le 3 più critiche.

## 3 Text analytics

Per calcolare uno score di soddisfazione per ciascun cliente sono necessarie tecniche di text classification da applicare alle risposte fornite da ciascun utente.

Prima di utilizzare questi dati è necessaria una prima fase di data-cleaning. Sono state individuate ed eliminate le domande con risposte non utili o domande non rilevanti per il calcolo del grado di soddisfazione dei clienti.

Sono state eliminate le risposte nulle oppure risposte che contenevano solo un segno di punteggiatura. Due domande sono state inserite in un nuovo questionario per tenere separato il feedback di quell'argomento e facilitare l'analisi successiva. Infine alcune risposte chiuse sono state sostituite con altre per riunire risposte che hanno lo stesso significato.

[3]:
```
#in una lista inseriamo le domande non utili per valutare il livello di␣
 ↪soddisfazione dei clienti
domande_eliminare=['Come sei venuto a conoscenza di Conto Crédit Agricole?',
     'Cosa ti aspetti dal tuo nuovo conto?',
     'Cosa ti ha spinto ad aprire Conto Crédit Agricole?',
     'Cosa ti ha spinto a scegliere Mutuo Agricole',
     'Desideri essere ricontattato?',
     'Cosa ti spinge a utilizzare la funionalità bonifico istantaneo?',
     'Come hai conosciuto questo servizio?',
     'Conosci la funzionalità Instant Payment per inviare denaro in tempo reale?
 ↪',
     'Quante volte sei stato contattato dal tuo Gestore negli ultimi 6 mesi?',
     'Solitamente, quale canale utilizzi per le tue operazioni bancarie?',
     'Come sei venuto a conoscenza di mutuo Crédit Agricole?',
     "Quali tra i seguenti servizi/vantaggi vorresti completassero l'offerta di␣
 ↪Mutuo CA?",
     'Qual è il tuo titolo di studio?',
     'Quale modalità hai utilizzato per effettuare il bonifico?',
     'Come sei venuto a conoscenza del Prestito Crédit Agricole?',
     'Altro_fonticomunicazione',
     'Desideri essere ricontattato?_1',
     'Cosa ti ha spinto a scegliere il Prestito Agos Credit Agricole?',
     'Come sei venuto a conoscenza della polizza?',
     'Cosa ti ha spinto ad acquistare una polizza assicurativa Crédit Agricole?
 ↪',
     'Quale tra le seguenti modalità',
     'Polizze online',
     'Quale tra le seguenti modalità Preventivo',
     'Altro_apprezzati',
     'A che punto sei del tuo percorso universitario?',
     'Stai svolgendo qualche attività lavorativa?',
```

```
    'Come hai attivato la tua Student Card?',
    'Quali sono i tuoi progetti o i tuoi bisogni?',
    'Che studente sei?',
    'Data di laurea?',
    'Argomenti preferiti_1',
    'Altro_argomentipreferiti',
    'Si',
    'No',
    'Quanto utilizzi la tua Student Card',
    'A che punto sei del tuo percorso universitario?',
    'Quali iniziative, in accordo con l'Ateneo, ti potrebbero interessare per␣
↪il futuro?',
    'Quando ha manifestato la volontà di estinguere il mutuo, quali soluzioni/
↪proposte le sono state offerte dal suo gestore',
    'Fra quelli elencati di seguito, qual è il motivo per cui ha estinto il␣
↪mutuo',
    'Ha provato a chiedere una rinegoziazione delle condizioni del mutuo',
    'Hai già utilizzato l'App di Crédit Agricole',
    'Utilizzerai l'App o l'Home Banking per fare il tuo prossimo pagamento?␣
↪(Es. bonifico, ricarica, operazione di compravendita sui mercati, ecc..)',
    'Userai l'App per le tue prossime operazioni?',
    'Altro_2',
    'Altro2',
    'Altro3'  ,
    'Qual è il principale aspetto che cambieresti per migliorare l'esperienza?
↪',
    'Qual è il principale aspetto che cambieresti per migliorare la tua␣
↪esperienza?',
    'Argomentazione_soddisfatti',
    'Argomentazione_insoddisfatti_1',
    'Puoi argomentare la tua valutazione?_positivi',
    'Puoi argomentare la tua valutazione?_negativi',
    'Può argomentare la sua valutazione_POS',
    'Qual è l'aspetto che hai maggiormente apprezzato?',
    'Visualizzare e firmare in digitale i documenti relativi all'operazione␣
↪che ha concluso tramite «CA per Te - la Consulenza Dinamica» è stato',
    "Qual è l'aspetto che hai maggiormente apprezzato?",
    'Per quali motivi ti ritieni soddisfatto della tua Filiale',
    'Per quali motivi non ti ritieni soddisfatto della tua Filiale?',
    'Puoi motivare la tua scelta?'
]

#eliminiamo le domande contenute nella lista e inseriamo le rimanenti nel␣
↪dataframe df
df = dataset[dataset['DESC_DOMANDA'].isin(domande_eliminare)==False]
```

```
#domande aperte da eliminare perchè non hanno risposte utili all'analisi:
df=df.loc[df['ID_DOMANDA']!=6275]
df=df.loc[df['ID_DOMANDA']!=7307]
df=df.loc[df['ID_DOMANDA']!=7310]
df=df.loc[df['ID_DOMANDA']!=7778]
df=df.loc[df['ID_DOMANDA']!=17500]
df=df.loc[df['ID_DOMANDA']!=12277]
df=df.loc[df['ID_DOMANDA']!=7574]

df=df.loc[df['ID_QUESTIONARIO']!=60] #eliminare questionario 60

df=df.dropna() #eliminare tutti i valori nan
df=df.loc[df['DESC_RISPOSTA']!='.'] #eliminare risposte con solo '.'
df=df.loc[df['DESC_RISPOSTA']!='?'] #eliminare risposte con solo '?'
df=df.loc[df['DESC_RISPOSTA']!='!'] #eliminare risposte con solo '!'

#eliminare le risposte 'No' alla domanda 'puoi argomentare'
y=df.loc[df['DESC_DOMANDA']=='Puoi argomentare questa tua valutazione?' ]
df=df.drop(y.loc[y['DESC_RISPOSTA']=='No'].index)

#aggiungere un questionario con solo le due domande  8273, 8274
df.loc[df['ID_DOMANDA']==8273, ['ID_QUESTIONARIO']]= 29
df.loc[df['ID_DOMANDA']==8274, ['ID_QUESTIONARIO']]= 29

#sostituiamo le stringhe passate come primo parametro alla funzione replace con␣
 ↪il secondo argomento.
df=df.replace('Per-niente', 'perniente')
df=df.replace('Perniente', 'perniente')
df=df.replace('Poco', 'poco')
df=df.replace('pocoutile', 'poco')
df=df.replace('abbastanzau', 'abbastanza')
df=df.replace('Abbastanza', 'abbastanza')
df=df.replace('soddisfacen', 'soddisfacente')
df=df.replace('nonadeguata', 'nonadeguato')
df=df.replace('Molto', 'molto')
df=df.replace('nonintuitiva', 'nonintuitivo')
df=df.replace('complessa', 'complesso')
df=df.replace('lenta', 'lento')
df=df.replace('intuitiva', 'intuitivo')
```

Nel dataframe 'chiuse' vengono inserite le risposte chiuse del tipo 'inputradio'. Per ogni riga del dataframe viene calcolato uno score che viene inserito in una lista che costituirà la nuova colonna inserita nel dataframe. Dopo un'analisi delle risposte sono state individuate 4 classi adatte alla classificazione delle stesse. Alle risposte del tipo 'Per niente', 'eccessivo', 'no', 'v0', 'v1', 'v2' è stato associato il punteggio più basso: -1. Se la risposta è 'poco', 'non adeguato', 'difficoltosa, 'v3', 'v4', 'v5' lo score è -0.5. 0.5 è assegnato alle risposte come 'abbastanza, 'soddisfacente', 'v6', 'v7', 'v8'. Il punteggio massimo (1.0) è associato a 'molto', 'eccellente', 'si', 'v9', 'v10'.

Lo stesso procedimento viene applicato alle domande con risposta multipla. Nel dataframe non è presente neanche una domanda del tipo 'inputmatrix'. Alle risposte 'altro' viene assegnato un punteggio 0 e poi verranno eliminate perchè 'altro' verrà esplicitato in una domanda aperta che verrà valutata quindi non è necessario assegnare uno score a questa risposta. Il valore 2 viene assegnato a risposte che non sono inserite nella classificazione perchè non esprimono un giudizio. Anche queste risposte verranno eliminate.

Una volta assegnato lo score a ogni domanda con risposta chiusa, viene creato un dataframe domande_chiuse che unisce tutte queste risposte e verrà utilizzato per calcolare i target delle risposte aperte necessari per l'addestramento del modello.

Prima di utilizzare il dataframe ottenuto, vengono eliminate le domande chiuse non utili per il calcolo del target della domanda aperta perchè esprimono pareri su argomenti diversi pur facendo parte dello stesso questionario. Nel dataframe 'medie' viene calcolato lo score di soddisfazione medio su ogni questionario della specifica sessione dello specifico cliente. Lo score di ogni riga verrà associato a ogni risposta aperta con il corrispondente id questionario, id sessione e id cliente.

Nel dataframe 'aperte' vengono inserite tutte le risposte aperte a cui andrà associato il corrispondente score. Per ogni riga di 'aperte' viene selezionato lo score in 'medie' associato a specifico cliente, dello specifico questionario, della specifica sessione. Poi la colonna degli score viene inserita in 'aperte'. Dato che i valori di score sono valori continui, questi devono essere etichettati in due classi: positivi (1.0) e negativi (-1.0). Gli score minori a 0.0 sono classificati come negativi e quelli superiori o uguali a 0.0 come positivi.

Il dataset di training e testing è ora disponibile. Nel dataframe samples vengono inserite solo le colonne con la risposta aperta e lo score associato. In questo modo si ha il text (la domanda aperta) e la label (lo score).

[4]:
```
#inserite le risposte del tipo 'inputradio' nel dataframe chiuse
chiuse=df.loc[(df['TIPO_RISPOSTA']=='inputradio')]
new_col=[]
i=0
for utente in chiuse['ID_CLIENTE']:
    riga=chiuse.iloc[i][:]
    #per ogni risposta viene assegnato lo score corrispondente
    if((riga[8]=='perniente') |( riga[8]=='eccessivo') |(riga[8]=='no' )|␣
 ↪(riga[8]=='v0')|(riga[8]=='v1')| (riga[8]=='v2') ):
        new_col.append(-1.0)
    elif ((riga[8]=='poco') |( riga[8]=='nonadeguato')|(␣
 ↪riga[8]=='difficoltosa')|(riga[8]=='v3' )|(riga[8]=='v4' )| (riga[8]=='v5'␣
 ↪)):
        new_col.append(-0.5)
    elif ((riga[8]=='abbastanza') |( riga[8]=='soddisfacente') |(riga[8]=='v6'␣
 ↪)|(riga[8]=='v7' )| (riga[8]=='v8' )):
        new_col.append(0.5)
    elif ((riga[8]=='molto') |( riga[8]=='eccellente') |(␣
 ↪riga[8]=='si')|(riga[8]=='v9' )| (riga[8]=='v10' ) ):
        new_col.append(1)
    i=i+1
```

```
chiuse.insert(9,'SCORE',new_col) #inseriamo la nuova colonna nel dataframe␣
↪'chiuse'

#mettiamo nel dataframe chiuse_multiple le risposte del tipo 'inputmulticheckb'
chiuse_multiple=df.loc[(df['TIPO_RISPOSTA']=='inputmulticheckb')]
new_col_2=[]
i=0
for utente in chiuse_multiple['ID_CLIENTE']:
    riga=chiuse_multiple.iloc[i][:]
    #per ogni risposta viene assegnato uno score che viene memorizzato in␣
↪new_col_2
    if((riga[8]=='lento') |( riga[8]=='complicato') |(riga[8]=='nonintuitivo' )|
       (riga[8]=='complesso')| (riga[8]=='lento')
       |(riga[8]=='problematichedisservizi')
       |(riga[8]=='relazioneinsoddisfacenteconilgestorebanca')
       |(riga[8]=='offertadinuoviprodottinonadeguataalleesigenze')
       | (riga[8]=='servizipocoinnovativi')):
        new_col_2.append(-1.0)
    elif ((riga[8]=='comodo') |( riga[8]=='veloce') |(␣
↪riga[8]=='semplice')|(riga[8]=='sicuro' )|
         (riga[8]=='intuitivo' )|(riga[8]=='aperturaaltrocontocrditagricole')):
         new_col_2.append(1)
    elif ((riga[8]=='altro')):
        new_col_2.append(0)
    else:
          new_col_2.append(2)

    i=i+1

chiuse_multiple.insert(9,'SCORE',new_col_2) #inseriamo la colonna nel dataframe
chiuse_multiple = chiuse_multiple[chiuse_multiple['SCORE']!=2] #eliminiamo le␣
↪risposte con score 2
chiuse_multiple = chiuse_multiple[chiuse_multiple['SCORE']!=0] #eliminiamo le␣
↪risposte con score 0

#uniti i due gruppi di domande chiuse uno sotto l'altro
domande_chiuse=pd.merge(left=chiuse, right=chiuse_multiple,how='outer') # ogni␣
↪risposta chiusa ha il corrispondente score

#eliminare domande chiuse non utili per il calcolo dei target
domande_chiuse=domande_chiuse.loc[domande_chiuse['ID_DOMANDA']!=18429]
domande_chiuse=domande_chiuse.loc[domande_chiuse['ID_DOMANDA']!=8219]
domande_chiuse=domande_chiuse.loc[domande_chiuse['ID_DOMANDA']!=16206]
domande_chiuse=domande_chiuse.loc[domande_chiuse['ID_DOMANDA']!=13766]
domande_chiuse=domande_chiuse.loc[domande_chiuse['ID_DOMANDA']!=19282]
```

```
#raggruppo le domande sul campo dell'utente, sull'id del questionario e id␣
 ↪della sessione. Dopo il raggruppamento viene
#fatta la media sul campo score (colonna 9)
medie=domande_chiuse.
 ↪groupby(['ID_CLIENTE','ID_QUESTIONARIO','ID_SESSIONE_QUESTIONARIO'])['SCORE'].
 ↪mean()
medie=medie.reset_index() # reset degli indici
medie['SCORE']=medie['SCORE'].round(2) # arrotondamento degli score a due cifre␣
 ↪dopo la virgola
medie.head(5)
```

[4]:

| | ID_CLIENTE | ID_QUESTIONARIO | ID_SESSIONE_QUESTIONARIO | SCORE |
|---|---|---|---|---|
| 0 | 6124029 | 29 | 35306 | 1.00 |
| 1 | 6124029 | 37 | 35306 | 0.83 |
| 2 | 6124029 | 51 | 820916 | -0.17 |
| 3 | 6124174 | 58 | 577474 | 0.50 |
| 4 | 6124554 | 29 | 187631 | 1.00 |

[8]:
```
#selezione di tutte le domande a risposta aperta
aperte=df.loc[(df['TIPO_RISPOSTA']=='inputtextarea')]

new_col_3=[] #lista in cui verranno inseriti i nuovi valori di colonna
i=0
for utente in aperte['ID_CLIENTE']: #iterazione su tutte le righe di aperte
    riga=aperte.iloc[i][:] # selezione della riga di aperte a cui associamo lo␣
 ↪score esatto
    id_quest=riga[1] #prendiamo il valore del questionario
    id_sessione=riga[3] #valore della sessione
    domande_rif=medie.loc[(medie['ID_CLIENTE']==utente)
                          &(medie['ID_QUESTIONARIO']==id_quest)
                          &(medie['ID_SESSIONE_QUESTIONARIO']==id_sessione)]
    #selezione in medie del della riga con specifico id cliente, id␣
 ↪questionario e id sessione
    media=domande_rif['SCORE'] # score da associare
    media=media.to_string(index=False) # conversione a stringa
    new_col_3.append(media) #inserimento del valore nella lista
    i=i+1

aperte.insert(9,'SCORE',new_col_3) #colonna inserita
aperte['SCORE']=aperte['SCORE'].astype(float) #conversione dei valori in float
aperte.loc[aperte['SCORE']<0.0,['SCORE']]=-1 #score minori a 0.0 diventano␣
 ↪uguali a -1
aperte.loc[aperte['SCORE']>=0.0,['SCORE']]=1 #score maggiori o uguali di 0.0␣
 ↪diventano uguali a 1.
cols = ['SCORE','DESC_RISPOSTA']
```

```
samples = aperte[cols]
samples.head(3)
```

[8]:

| | SCORE | DESC_RISPOSTA |
|---|---|---|
| 5 | 1.0 | Ho avuto bisogno di aiuto in agenzia |
| 23 | 1.0 | Facile per accedere |
| 29 | 1.0 | Serietà e Professionalità |

Prima di utilizzare il dataset per l'addestramento del modello, si applica una fase di preprocessing sulle risposte aperte per aiutare il modello ad apprendere meglio dai dati.

- Viene fatto il tokenizing del testo. La tokenizzazione è una tecnica utilizzata per suddividere un testo in unità più piccole, come singole parole o termini chiamate token. In questo progetto è stato utilizzato sent_tokenize, cioè un Tokenizer (fornito dal modulo nltk) che mantiene intatte le frasi. Le frasi o parole tokenizzate possono essere trasformate in dataframe e vettorizzate.
- Viene eliminata la punteggiatura e le stopword. Vengono cancellate tutte quelle informazioni che non portano ad una maggior informazione, ma al contrario aggiungono solamente rumore.
- Eliminare le parole grammaticalmente errate. Utilizzando un file txt con una raccolta delle parole in lingua italiana, sono state eliminate le parole che non risultavano in quell'elenco.
- Stemming: un algoritmo che elimina i suffissi delle parole per raggruppare parole che hanno stesso significato, ma suffisso diverso come verbi coniugati oppure parole al femminile o maschile.

[18]:
```
#importate le librerie necessarie al preprocessing
from string import punctuation
import os
import nltk as nltk
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords

tokenize =sent_tokenize #in tokenize viene messa la sent_tokenize

stemmer=nltk.stem.snowball.ItalianStemmer()

#memorizziamo in una variabile l'insieme delle congiunzioni italiane
italian_stopwords = set(stopwords.words('italian'))

#lettura del file con tutte le parole italiane. Tutte le parole contenute del␣
 ↪file vengono memorizzate in un insieme
all_italian_words = set(word.replace("\n", "") for word in open("italian_words.
 ↪txt").readlines())

#funzione per tokenizzare un testo.
def get_tokenized_text(text):
    return " ".join(tokenize(text))
```

```
    #la funzione join() unisce elementi iterabili come una lista e restituisce␣
 ↪una stringa concatenata come output

def get_text_stemmed(text):
    return " ".join([stemmer.stem(w) for w in text.split()])

#funzione che elimina le congiunzioni
def get_text_without_stopwords(text):
    return " ".join([word for word in text.split() if word not in␣
 ↪italian_stopwords])
    #se la parola contenuta in text non fa parte di italian_stopwords questa␣
 ↪non viene aggiunta

#funzione che rimuove la punteggiatura
def get_text_without_punctuation(text):
    return text.translate(str.maketrans('', '', punctuation))
    #come terzo parametro di maketrans viene passata la lista di caratteri che␣
 ↪devono essere eliminati

#funzione che rimuove le parole scritte in modo errato
def get_text_without_uncorrect_words(text):
    return " ".join([word for word in text.split() if word in␣
 ↪all_italian_words])
    #word deve essere contenuto in all_italian_words, altrimenti viene eliminata


def get_normalized_text(text):
    text = get_tokenized_text(text) #testo viene tokenizzato
    text = get_text_without_stopwords(text) #eliminazione stopwords
    text = get_text_without_punctuation(text) #eliminazione punteggiatura
    text = get_text_without_uncorrect_words(text) #eliminazione parole scritte␣
 ↪in modo scorretto
    text = get_text_stemmed(text) #stemming del testo
    return text


def get_label(sample):
    return sample[0] #restituisce la lable di sample


def get_text(sample):
    return sample[1].lower() #ogni lettera della risposta viene trasformata in␣
 ↪lower case e restituisce text di sample


def preprocess_samples(samples):
```

```
    samples = samples.values.tolist() #valori di samples trasformati in una␣
↪lista

    #per ogni sample prendiamo la label e il testo che viene normalizzato
    normalized_samples = [(get_label(sample),␣
↪get_normalized_text(get_text(sample))) for sample in samples]

    #eliminiamo i sample con testo vuoto
    normalized_samples = [sample for sample in normalized_samples if␣
↪get_text(sample) != ""]

    #creiamo dataframe con una colonna label e una text
    normalized_samples = pd.DataFrame(set(normalized_samples),␣
↪columns=["label", "text"])

    return normalized_samples

#preprocess
samples=preprocess_samples(samples)
```

### 3.1 Feature Extraction

Con Feature Extraction possiamo indicare tutte quelle tecniche che a partire da un testo hanno l'obiettivo di trasformarlo in un insieme di feature che possono essere utilizzate da un algoritmo. Più nel dettaglio possiamo dire che queste tecniche dato un testo, lo trasformano in una tabella (chiamata bag of words o Bow) con la quale ogni parola di questo testo (ora diventata una feature) è associata ad un valore numerico. Questa bag of words, alla fine di questo processo non sarà altro che il nostro nuovo dataset.

### 3.2 TfidfVectorizer

All'interno di questo progetto è stato utilizzato un Tf-idf Vectorizer, un algoritmo che trasforma il testo in una bag of word ma con un criterio un po' diverso da un semplice Countvectorizer.

Un TfidfVectorizer produce comunque una bag of words, ma i valori presenti all'interno di essa non sono solo valori dicotomici ma sono generati calcolando il Tf (Term frequency) della parola e moltiplicandolo per l'Idf (Inverse document frequency) del documento.

Il Term frequency indica quante volte una certa parola è presente in un testo. L'Inverse Document frequency ed indica l'inverso della Df (Document frequency) cioè il numero di documenti dove appare una determinata parola. Idf = log (1/Df) con Df = #{testo : testi | parola  testo} Il Tfidf è direttamente proporzionale al numero di volte che una parola è presente in una frase ed inversamente proporzionale a quanti documenti contengono quella parola. Una parola con un TfIdf alto è una parola che ha molto più peso rispetto alle altre.

### 3.3 N-grammi

Gli n-grammi sono sottesequenze di una sequenza, che nel nostro caso è il testo della risposta; un trigramma è un n-gramma formato da 3 elementi/parole. Tutte le parole di tutte le risposte del

dataset vanno a formare gli attributi (le feature, i valori nelle colonne) della nostra bag of words, ma c'è la possibilità di "settare" come feature anche tutti gli n-grammi dei documenti di un dataset, con il fine di cercare di aumentare l'accuratezza e la qualità del nostro modello. Nel modello utilizzato i migliori risultati sono stati ottenuti con n=4.

### 3.4 Feature Selection

La Feature Selection è un processo che prevede di selezionare solo una parte più o meno ampia delle features a nostra disposizione per migliorare l'efficienza ed eliminare feature "rumorose" che non portano grandi informazioni aggiuntive.

Dopo test empirici le feature selezionate per il modello impiegato sono le 113000 feature più importanti, cioè le 113000 feature che hanno ottenuto lo "score" più alto con la funzione di score f_classif (basata sull'ANOVA, Analysis of Variance, ed utilizzabile solo per variabili categoriche)

### 3.5 Addestramento

Per addestrare il modello, il dataset è stato suddiviso in train/test, rispettivamente 80%/20% delle risposte. Gli algoritmi forniti da sklearn (e le loro accuratezze) provati per l'addestramento sono i seguenti:

Multinomial Bayes: 85.01 % BernoulliNB: 88.89 % % Complement Bayes: 76.8 % RandomForestClassifier: 85.01 %

BernoulliNB è adatto per feature booleane o binarie quindi è perfetto per il problema.

Il classificatore multinomiale Naive Bayes classifier è adatto alla classificazione con valori di feature discreti (ad esempio il conteggio di parole per la text classification).

Il classificatore Complement Naive Bayes è stato ideato per correggere assunzioni gravi fatte dal classificatore standard Multinomial Naive Bayes. Inoltre è particolarmente adatto a dataset non bilanciati.

Random forest crea un insieme di alberi di decisione su vari sottoinsiemi del dataset e riduce l'overfitting. La dimensione del sottoinsieme del dataset è definita dal parametro max_samples parameter altrimenti ogni albero viene costruito usando tutto il dataset.

L'accuratezza migliore è ottenuta con il modello BernoulliNB ( 88.89 %).

La sua confusion matrix è la seguente:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| -1 | 0.99 | 0.26 | 0.41 | 927 |
| 1 | 0.88 | 1.00 | 0.94 | 5258 |
| accuracy | | | 0.89 | 6185 |
| macro avg | 0.94 | 0.63 | 0.68 | 6185 |
| weighted avg | 0.90 | 0.89 | 0.86 | 6185 |

Nella confusion matrix sono presenti valori di:

- Accuratezza: percentuale di risposte classificate correttamente (0.89)

- Precision: percentuale di risposte classificate come positive/negative che sono realmente positive/negative. 0.99 per le negative e 0.88 per le positive.
- Recall: percentuale di risposte positive/negative che sono state classificate come positive/negative. 0.26 per le negative (percentuale bassa), 1.0 per le positive.
- f1-score: la media tra precision and recall. Tiene in considerazione sia i falsi positivi che i falsi negativi. 0.41 per le risposte negative e 0.94 per le positive.

[23]:
```
import time
import pandas as pd
import numpy as np
import joblib
import os
from random import shuffle


from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.feature_selection import SelectKBest, chi2, f_classif

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import MultinomialNB, BernoulliNB, ComplementNB
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix

def show_performance_data(Y_test, Y_pred, model_name):
  print(classification_report(Y_test, Y_pred, target_names=['-1','1']))
  tmp_result = classification_report(Y_test, Y_pred, target_names=['-1','1'],␣
 ↪output_dict=True)
  return tmp_result

def train_test_classificators_and_get_the_best_one(classificators, X_train,␣
 ↪X_test, y_train, y_test):
    best_accuracy = -1

    for clf, name in classificators:
        clf.fit(X_train, y_train)
        score = round(clf.score(X_test, y_test) * 100, 2)

        print(name + ": " + str(score) + " %")

        if best_accuracy < score:
            best_accuracy = score
            best_clf = clf
            best_clf_name = name
```

```
    return best_clf, best_clf_name


def get_X_and_y_after_features_extraction(samples, ngram_range):
    vectorizer = TfidfVectorizer(ngram_range=ngram_range) #creato oggetto␣
↪TfidfVectorizer passando le dimensioni del n-gramma

    X = vectorizer.fit_transform(samples["text"].astype(str).tolist())#␣
↪#fit_transform apprende il vocabolario e restituisce le risposte in una␣
↪matrice
    y = samples["label"].tolist()

    return X, y, vectorizer


def get_X_after_feature_selection(X, y, best_feature_number):
    np.seterr(invalid='ignore')

    feature_selector = SelectKBest(score_func=f_classif, k=best_feature_number)␣
↪#selezione delle feature più importanti calcolando lo score con la funzione␣
↪f_classif
                                        #f_classif prenderà due array X e y e␣
↪ritornerà un array con gli score associati a ogni feature.
    selected_feature = feature_selector.fit_transform(X, y) #X viene ridotto␣
↪alle feature selezionate

    print("selected features number " + str(best_feature_number) + "\n")
    return selected_feature, feature_selector


def get_label(samples):
    return samples[0]


def get_balanced_dataframe(samples):
    samples = samples.values.tolist() #samples trasformato in una lista

    shuffle(samples) #prende una lista e cambia l'ordine degli elementi

    good_samples = [sample for sample in samples if get_label(sample) == 1]␣
↪#calcolo del numero di risposte positive
    bad_samples = [sample for sample in samples if get_label(sample) == -1]␣
↪#calcolo numero di risposte negative

    print("total samples number " + str(len(samples)))
```

```
    print("good samples number " + str(len(good_samples)))
    print("bad samples number " + str(len(bad_samples)))

    balanced_number_of_samples = min(len(good_samples), len(bad_samples))

    print("selected samples number for each class " +␣
 ↪str(balanced_number_of_samples))

    balanced_samples = good_samples[:balanced_number_of_samples] + bad_samples[:
 ↪balanced_number_of_samples] #prendo lo stesso numero di risposte positive e␣
 ↪negative

    balanced_samples = pd.DataFrame(balanced_samples, columns=["label", "text"])

    return pd.DataFrame(samples, columns=["label", "text"]) #restituisco␣
 ↪dataset per addestramento


#############################################################
#
#                           | MAIN |
#
#############################################################

test_size = 0.2 #dimensione del dataset di testing
ngram_range = (2, 4) #2,4 #dimensione del n-gramma
best_feature_number = 113000 #113000 #100000 #118000 numero di feature␣
 ↪selezionate

# 1° bilanciamento tra numero di risposte negative e positive e shuffle delle␣
 ↪risposte
samples_1 = get_balanced_dataframe(samples)

# 2° feature extraction
X, y, vectorizer = get_X_and_y_after_features_extraction(samples_1, ngram_range)

# 3° prendiamo solo determinate feature
X, feature_selector = get_X_after_feature_selection(X, y, best_feature_number)

# 4° divisione tra training e test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,␣
 ↪random_state=42)


classificators = [
                    (MultinomialNB(), "multinomial bayes"),
```

```
                    (BernoulliNB(), "BernoulliNB"),
                    (ComplementNB(), "complement bayes"),
                    (RandomForestClassifier(n_estimators=70, oob_score=True,␣
 ↪n_jobs=-1, random_state=101, min_samples_leaf=30),␣
 ↪"RandomForestClassifier"), # lento
                ]

# 6° prende la lista dei classificatori e prende il migliore
best_clf, best_clf_name =␣
 ↪train_test_classificators_and_get_the_best_one(classificators, X_train,␣
 ↪X_test, y_train, y_test)

Y_pred=best_clf.predict(X_test)
ris=show_performance_data(y_test, Y_pred, best_clf_name) #per mostrare la␣
 ↪matrice di confusione del modello migliore
```

```
total samples number 30925
good samples number 26100
bad samples number 4825
selected samples number for each class 4825
selected features number 113000

multinomial bayes: 85.01 %
BernoulliNB: 88.89 %
complement bayes: 76.8 %
RandomForestClassifier: 85.01 %
              precision    recall  f1-score   support

          -1       0.99      0.26      0.41       927
           1       0.88      1.00      0.94      5258

    accuracy                           0.89      6185
   macro avg       0.94      0.63      0.68      6185
weighted avg       0.90      0.89      0.86      6185
```

Ora vengono uniti in un unico dataframe 'domande' tutti i dataframe che contengono le risposte e i loro score. Per ogni cliente si può calcolare il suo score di soddisfazione. Viene prima effettuata una groupby su l'id del cliente e poi calcolata la media sulla colonna 'SCORE'. Il risultato è memorizzato nella variabile 'score_utente'.

Per individuare i tre clienti più soddisfatti viene ordinato il dataframe 'score_utente' in ordine descrescente rispetto allo score. Molti clienti risultano avere lo score massimo (+1), quindi si prende in considerazione anche quanti questionari hanno compilato. I clienti più soddisfatti risultano essere 20200569, 23007684, 13076375.

Con gli stessi criteri vengono trovati i clienti più insoddisfatti, cioè quelli che hanno score -1 e hanno compilato il maggior numero di questionari. Questi sono 7117139, 22996262, 11478292.

[25]:
```
#fare merge verticale di tutte le domande con i loro score
#unire i due gruppi di domande
domande=pd.merge(left=chiuse, right=chiuse_multiple,how='outer')
domande=pd.merge(left=domande, right=aperte,how='outer')

#groupby su ID_CLIENTE e poi calcolata la media su 'SCORE'
score_utente=domande.groupby(['ID_CLIENTE'])['SCORE'].mean()
score_utente=score_utente.reset_index() #resettati gli indici
score_utente.columns=['ID_CLIENTE','SCORE']
score_utente['SCORE']=score_utente['SCORE'].round(2) # arrotondamento degli␣
↪score a due cifre dopo la virgola

utente_ndomande=domande.groupby(['ID_CLIENTE']).count()['ID_QUESTIONARIO']␣
↪#contare quanti questionari per ogni cliente
utente_ndomande=utente_ndomande.reset_index()
utente_ndomande.columns=['ID_CLIENTE', 'N_QUESTIONARI'] #rinominiamo le colonne
utenti_ordinati=pd.merge(left=score_utente, right=utente_ndomande,how='outer')␣
↪#associamo a ogni cliente il numero di questionari svolti
utenti_ordinati_desc=utenti_ordinati.sort_values(by=['SCORE','N_QUESTIONARI'],␣
↪ascending=False) #in ordine descrescente di score e n questionari
print(utenti_ordinati_desc.head(3))
utenti_ordinati_asc=utenti_ordinati.sort_values(by=['SCORE','N_QUESTIONARI'],␣
↪ascending=[True,False]) #in ordine crescente di score e decrescente di␣
↪questionari
print(utenti_ordinati_asc.head(3))
```

```
       ID_CLIENTE  SCORE  N_QUESTIONARI
71092    20200569    1.0             71
89281    23007684    1.0             55
47081    13076375    1.0             45
       ID_CLIENTE  SCORE  N_QUESTIONARI
5692       7117139   -1.0             18
39236    11478292   -1.0             10
89011    22996262   -1.0             10
```

# 4 Graph Analytics

## 4.1 Importazione dati su Neo4j

I dati prodotti devono essere importati in Neo4j, ma prima bisogna produrre dei file csv con i dati strutturati correttamente.

- Viene creato un csv con tutti i clienti che compaiono in dataset2 in modo da avere tutte le informazioni per i nodi Cliente che saranno importati ('clienti_all.csv').
- un altro file con gli score associati ai clienti ('score.csv').
- un file con i collegamenti. Al dataset2 che contiene i collegamenti vengono aggiunte due colonne che attribuiscono un valore a ciascuna relazione. Questi campi sono necessari per

una futura analisi del grafo neo4j. Questi campi vengono presi dal file peso.csv. Il file contiene ogni tipo di relazione a cui è associato un costo e un peso. Il peso rappresenta la forza della relazione, il costo è il costo per attraversare quel tipo di arco del grafo. Le due colonne vengono aggiunte a dataset2 e poi viene creato il file ('dataset_collegamenti.csv').

- un file con le filiali. Per l'obiettivo 2 della graph analysis è necessario avere anche i nodi delle filiali. Creiamo un file con l'elenco degli id delle filiali presenti in dataset2 ('filiali.csv').
- un file con i collegamenti tra clienti e la loro filiale ('coll_cliente_filiale.csv').

[35]:
```
#prendo le colonne con id cliente, id filiale e codice natura giuridica
ID_CLIENTE=dataset2.iloc[:, 0:3]
ID_CLIENTE2=dataset2.iloc[:,5:8]
ID_CLIENTE.columns=['ID_CLIENTE','COD_NATURA_GIURIDICA','ID_FILIALE'] #rinomino␣
 ↪le colonne
ID_CLIENTE2.columns=['ID_CLIENTE','COD_NATURA_GIURIDICA','ID_FILIALE']␣
 ↪#rinomino le colonne
ID_CLIENTE=ID_CLIENTE.merge(ID_CLIENTE2,how='outer') #merge verticale
ID_CLIENTE=ID_CLIENTE.drop_duplicates() #eliminati i clienti duplicati
ID_CLIENTE.to_csv('clienti_all.csv',index=False, header=False) # creato file␣
 ↪csv con tutti i clienti

#DATASET con score
lista=list(ID_CLIENTE['ID_CLIENTE']) #prendo la lista degli id dei clienti
score_utente=score_utente.loc[score_utente['ID_CLIENTE'].isin(lista)] #prendo␣
 ↪gli score (se esistono) di quei clienti
score_utente.to_csv('score.csv',index=False) # creato file csv con tutti gli␣
 ↪score associati all'id cliente

#DATASET collegamenti
pesi = pd.read_csv('peso.csv', sep=",", header=0,dtype = {
    'COD_COLLEGAMENTO': str,
    'DESC_COLLEGAMENTO': str,
    'PESO': int,
    'COSTO': int })
dataset2_pesi=pd.merge(left=dataset2, right=pesi,how='inner')
dataset2_pesi.to_csv('dataset_collegamenti.csv',index=False, header=False) #csv␣
 ↪senza index e header

#DATASET filiali
filiali=ID_CLIENTE['ID_FILIALE'].drop_duplicates() #prendiamo la colonna␣
 ↪filiali e eliminiamo i duplicati
filiali=filiali.astype(int)
filiali.to_csv('filiali.csv',index=False, header=False) # creato file csv con␣
 ↪tutte le filiali

#DATASET collegamenti clienti-filiali
```

```
coll_cliente_filiale=ID_CLIENTE.loc[:,['ID_CLIENTE','ID_FILIALE']] #prendiamo␣
↪le colonne clienti e filiali
coll_cliente_filiale.to_csv('coll_cliente_filiale.csv',index=False,␣
↪header=False) #csv senza index e header
```

Dovendo importare una grande mole di dati è stato utilizzato il bulk importing effettuato da shell con il comando:

```
bin\neo4j-admin import --database neo4j --id-type INTEGER
 --nodes=Cliente="clienti_header_1.csv,clienti_all.csv"
 --nodes=Filiale="filiali_header.csv,filiali.csv"
 --relationships=COLLEGAMENTO="collegamenti_header.csv,dataset_collegamenti.csv"
 --relationships=FILIALE_DI="coll_filiale_header.csv,coll_cliente_filiale.csv"
 --trim-strings=true
```

Sono presenti degli ‘a capo’ per favorire la leggibilità.
Per ogni importazione di nodi o relazioni è stato creato un header file dove è indicato il tipo e il nome di ogni property.

Per aggiungere la property score ad alcuni clienti è stato utilizzato il codice Cypher:

```
// prima è stato creato un indice sulla property id_cliente per velocizzare l'importazione
CREATE INDEX index_id IF NOT EXISTS
FOR (n:Cliente)
ON (n.id_cliente)


:auto USING PERIODIC COMMIT //periodicamente viene fatto commit per evitare di occupare tutta la memoria a disposizione
LOAD CSV WITH HEADERS FROM 'file:///score.csv' AS row //row contiene una riga del file
MATCH (c:Cliente {id_cliente: toInteger(row.ID_CLIENTE) }) //prendiamo il nodo cliente indicato dal file
SET c.score=toFloat(row.SCORE) //settiamo lo score del cliente con il valore preso dal file
```

Il grafo ottenuto ha nodi di tipo Cliente e Filiale.

Il nodo Cliente ha le seguenti property:
- "id_cliente"

- "id_filiale"

- "cod_natura_giuridica"

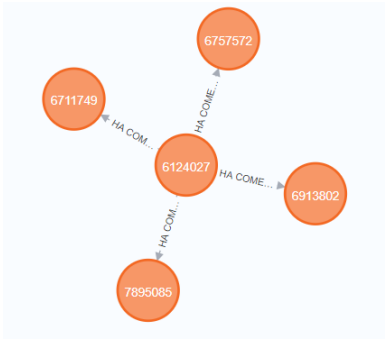
Il nodo Filiale ha solo la property id_filiale.

I clienti sono connessi tra loro dalla relazione di tipo COLLEGAMENTO che ha come property:
- "desc_collegamento": descrizione del collegamento

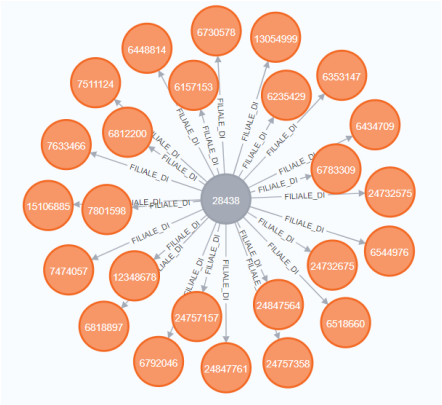
- "cod_collegamento": codice del collegamento

- "peso": forza della relazione

- "costo ": costo di attraversamento dell'arco




Le filiali sono collegate ai clienti con la relazione FILIALE_DI.



## 4.2 Obiettivo 1

### Development phase

**Approccio 1**

Fissato un nodo sorgente con un ID_CLIENTE e il suo score di soddisfazione, viene individuato il grafo dei clienti.
Il sottografo dei clienti è composto dal nodo sorgente e da tutti i nodi Cliente raggiungibili partendo dal nodo sorgente.

Inizialmente viene creato l'in-memory graph 'client_graph' che poi verrà utilizzato per l'analisi.
Effettuiamo la memory estimation:

```
//memory estimation del grafo dei clienti
CALL gds.graph.create.cypher.estimate(
"MATCH (c:Cliente)
RETURN id(c) AS id, c.score AS score",
"MATCH (c:Cliente)-[r:COLLEGAMENTO]->(c2:Cliente)
RETURN id(c) AS source, id(c2) AS target")
YIELD requiredMemory,bytesMin, bytesMax,  nodeCount,  relationshipCount
```

| requiredMemory | bytesMin | bytesMax | nodeCount | relationshipCount |
|---|---|---|---|---|
| "[133 MiB ... 147 MiB]" | 140418616 | 154313944 | 3505346 | 4089535 |

Creiamo l'in-memory graph utilizzando la cypher projection:

```
//creazione grafo dei clienti
CALL gds.graph.create.cypher('client_graph', // nome del grafo che creiamo
"MATCH (c:Cliente) //selezioniamo tutti i nodi Cliente
RETURN id(c) AS id, c.score AS score",
"MATCH (c:Cliente)-[r:COLLEGAMENTO]->(c2:Cliente) //selezioniamo le relazioni di tipo COLLEGAMENTO
RETURN id(c) AS source, id(c2) AS target")
```

| graphName | nodeCount | relationshipCount | createMillis |
|---|---|---|---|
| "client_graph" | 3505346 | 4089536 | 19004 |

Le property selezionate sono l'id del nodo Cliente e lo score.
Per quanto riguarda le relationship si restituisce l'id della sorgente e id del target dell'arco.

Definiamo un approccio che consente di individuare la rete di influenza del nodo sorgente 6189679 (preso come esempio). La pipeline è così composta:
- viene selezionato il nodo source di tipo Cliente con la property id_cliente: 6189679

- utilizzando il grafo creato precedentemente, viene invocato l'algoritmo Dijkstra Single-Source per calcolare tutti i path che hanno come sorgente source e come target tutti i nodi raggiungibili. L'algoritmo Dijkstra Shortest Path calcola il percorso più breve tra nodi. La versione Single-Source calcola il percorso più breve tra un nodo sorgente e tutti i nodi raggiungibili da quel nodo. È stato scelto questo algoritmo perché si adattava perfettamente al problema potendo fissare solo il nodo sorgente senza specificare il nodo target e cercando il percorso più breve.
- i valori restituiti richiesti sono il nodo sorgente, i nodi target e il percorso quando il nodo sorgente è diverso dal nodo target.
- viene calcolata la lunghezza dei vari percorsi e chiamata 'length'. In sourceScore e targetScore vengono salvati i valori di score della sorgente e del target.
- come ultimo step vengono restituiti gli id della sorgente e del target e il grado di influenza. Il valore assoluto di questo viene calcolato dividendo lo score del nodo sorgente per la lunghezza del percorso (cioè quanti archi sono stati percorsi); così più il nodo che influenza è lontano meno il nodo connesso viene influenzato. Il segno del grado di influenza è positivo se il nodo target non ha lo score oppure se il suo score è positivo o uguale a 0, altrimenti il segno è negativo.

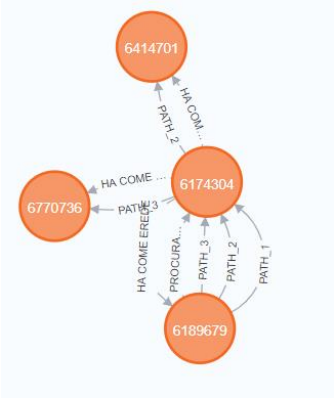
Prima di eseguire l'algoritmo viene effettuata la memory estimation dell'applicazione dell'algoritmo:

```
//memory estimation dell'algoritmo di path finding
MATCH (source:Cliente{id_cliente:6189679})
CALL gds.allShortestPaths.dijkstra.stream.estimate('client_graph', {sourceNode:source})
YIELD requiredMemory,bytesMin, bytesMax
```

| requiredMemory | bytesMin | bytesMax |
|---|---|---|
| "107 MiB" | 113048136 | 113048136 |

Con il seguente codice si ottiene il sottografo del nodo sorgente:

```
//creazione sottografo
MATCH (source:Cliente{id_cliente:6189679})
CALL gds.allShortestPaths.dijkstra.stream('client_graph', {sourceNode:source})
YIELD sourceNode, targetNode, path
RETURN  sourceNode, targetNode, path
```




Questo è il codice completo:

```
//calcolo gradi di influenza
MATCH (source:Cliente{id_cliente:6189679}) //selezioniamo come sorgente il nodo Cliente con l'id fissato
CALL gds.allShortestPaths.dijkstra.stream('client_graph',{sourceNode:source}) //calcolo percorsi partendo
da source
YIELD sourceNode, targetNode, path
WHERE sourceNode<>targetNode // nodo di partenza e di arrivo devono essere diversi
WITH
   sourceNode, targetNode,
   length(path) AS length, // calcolo lunghezza percorso
   nodes(path) AS path, //nodi attraversati
   gds.util.asNode(sourceNode).score AS sourceScore, //score del nodo sorgente
   gds.util.asNode(targetNode).score AS targetScore //score del nodo target
RETURN gds.util.asNode(sourceNode).id_cliente AS sourceNodeId, //id del nodo sorgente
   gds.util.asNode(targetNode).id_cliente AS targetNodeId, //id del nodo target
   sourceScore,
   targetScore,
CASE // se lo score del nodo target è positivo o nullo, il grado di influenza è positivo
   WHEN gds.util.asNode(targetNode).score >=0 OR
      gds.util.asNode(targetNode).score IS NULL THEN sourceScore/length
   ELSE -(sourceScore/length) // altrimenti il grado di influenza è negativo
END AS grado_influenza
```

Di seguito il risultato restituito:

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 6189679 | 6174304 | 0.5 | *null* | 0.5 |
| 6189679 | 6414701 | 0.5 | *null* | 0.25 |
| 6189679 | 6770736 | 0.5 | *null* | 0.25 |

Nessun nodo target ha un valore di score. Lo score del nodo sorgente è 0.5 e la sua distanza dal nodo 6174304 è 1 quindi il grado di influenza coincide con il suo score. Gli altri due nodi hanno grado di influenza 0.25, perché distano 2 dal nodo sorgente.

**Soluzione alternativa**

In questo approccio alternativo oltre all'algoritmo di path finding viene utilizzato l'algoritmo WCC (Weakly Connected Components). Questo algoritmo individua insiemi di nodi connessi in un grafo non orientato dove i nodi dello stesso insieme formano un componente connesso. WCC viene utilizzato per comprendere la struttura del grafo e permette di applicare algoritmi successivi in modo indipendente su un cluster fissato.

Nel file csv, contenente i collegamenti, è già stato associato un intero tra 0 e 2 che rappresenta la forza di quella relazione. 2 è la forza massima.
Se la relazione è debole il peso sarà 1, se forte 2. A relazioni considerate non rilevanti è stato assegnato 0.

In questa versione il grado di influenza dipende anche dalla forza della relazione, per questo è stata individuata una property 'costo' della relationship COLLEGAMENTO. La property 'peso' assegna un valore maggiore alle relazioni forti. Volendo associare un costo al path in base alla forza, per una relazione forte il

costo deve essere più basso. In questo modo verrà scelto il percorso con le relazioni che determinano una influenza maggiore. Se la forza della relazione è 1 o 0, il costo è 2, invece se è 2 il costo è 1. Così se la forza della relazione è 2 la lunghezza rimane uguale, se è 1 raddoppia. Le relazioni con peso 0 non verranno incluse nel calcolo.

Come primo step viene effettuata una memory estimation dell'in-memory graph che verrà creato.
A differenza di prima, vengono restituite anche le property 'peso' e 'costo' della relationship COLLEGAMENTO.

```
//memory estimation del grafo dei clienti
CALL gds.graph.create.cypher.estimate(
"MATCH (c:Cliente)
RETURN id(c) AS id, c.score AS score",
"MATCH (c:Cliente)-[r:COLLEGAMENTO]->(c2:Cliente)
RETURN id(c) AS source, id(c2) AS target, r.peso as peso, r.costo as costo ")
YIELD requiredMemory,bytesMin, bytesMax, nodeCount,  relationshipCount
```

| requiredMemory | bytesMin | bytesMax | nodeCount | relationshipCount |
|---|---|---|---|---|
| "[347 MiB ... 361 MiB]" | 364821384 | 378716712 | 3505346 | 4089178 |

Viene creato l'in-memory graph 'graph_pesato'.

```
//creazione grafo dei clienti
CALL gds.graph.create.cypher('graph_pesato', // nome del grafo che creiamo
"MATCH (c:Cliente)
  RETURN id(c) AS id, c.score AS score",
"MATCH (c:Cliente)-[r:COLLEGAMENTO]->(c2:Cliente) //selezioniamo le relazioni di tipo COLLEGAMENTO
  RETURN id(c) AS source, id(c2) AS target, r.peso as peso, r.costo as costo")
```

In seguito viene applicato l'algoritmo WCC scrivendo la property 'componentId' sul grafo originale neo4j e usando 'peso' come property per determinare il peso delle relazioni.
Se una relationship non ha un peso, l'algoritmo usa il valore di default zero.
Il valore di threshold indica il peso sopra il quale la relazione viene considerata nel calcolo. In questo caso il threshold è fissato a 0, cioè le relazioni con peso uguale a 0 non vengono considerate nel calcolo.
Prima di eseguire l'algoritmo viene fatta la stima della memoria necessaria.

```
// memory estimation dell'algoritmo di community detection
CALL gds.wcc.write.estimate('graph_pesato', { // nome del grafo utilizzato
  writeProperty: 'componentId', // nome della property creata
  relationshipWeightProperty: 'peso', // nome della property che indica il peso
  threshold: 0  // valore di threshold
})
YIELD requiredMemory,bytesMin, bytesMax,  nodeCount,  relationshipCount
```

| requiredMemory | bytesMin | bytesMax | nodeCount | relationshipCount |
|---|---|---|---|---|
| "26 MiB" | 28042960 | 28042960 | 3505346 | 4089178 |

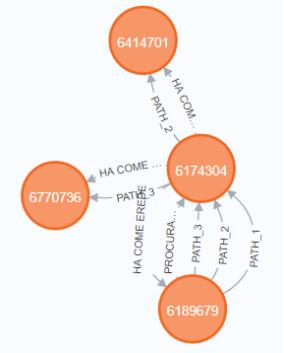
```
// applicazione dell'algoritmo di community detection
CALL gds.wcc.write('graph_pesato', { // nome del grafo utilizzato
  writeProperty: 'componentId', // la property che verrà scritta sui nodi Cliente
  relationshipWeightProperty: 'peso', // nome della property utilizzata come peso
  threshold: 0 // valore di threshold
})
YIELD nodePropertiesWritten, componentCount;
```

Ora i nodi hanno la property 'componentId' che è un valore intero che rappresenta l'id del cluster di cui fa parte il nodo.

Riapplichiamo l'algoritmo di path finding, aggiungendo la condizione che i nodi target devono appartenere allo stesso cluster del nodo sorgente.

Con il seguente codice si ottiene il sottografo del nodo sorgente:

```
//creazione sottografo
MATCH (source:Cliente{id_cliente:6189679})
CALL gds.allShortestPaths.dijkstra.stream('graph_pesato', {sourceNode:source})
YIELD sourceNode, targetNode, path, totalCost
WHERE sourceNode<>targetNode AND (gds.util.asNode(sourceNode).componentId=gds.util.asNode(target
Node).componentId) // nodo sorgente e target devono essere diversi e il loro component id deve essere
uguale
RETURN  sourceNode, targetNode, path, totalCost
```




Il codice completo è uguale alla versione precedente, ma in aggiunta è presente la condizione sulla property componentId e la lunghezza del path è uguale al costo totale del percorso.
Il nome che diamo a relationshipWeightProperty indica la property che verrà usata come peso (in questo caso è 'costo').

Prima di eseguire l'algoritmo viene eseguita una memory estimation:

```
// memory estimation dell'algoritmo di path finding
MATCH (source:Cliente{id_cliente:6189679})
CALL gds.allShortestPaths.dijkstra.stream.estimate('graph_pesato', {
      sourceNode:source,
      relationshipWeightProperty: 'costo' // property utilizzata come peso delle relazioni
})
YIELD requiredMemory,bytesMin, bytesMax
RETURN requiredMemory, bytesMin, bytesMax
```

| requiredMemory | bytesMin | bytesMax |
|---|---|---|
| "75 MiB" | 79552920 | 79552920 |

Il codice completo:

```
//calcolo gradi di influenza

MATCH (source:Cliente{id_cliente:6189679}) //selezioniamo come sorgente il nodo Cliente con l'id fissato
CALL gds.allShortestPaths.dijkstra.stream('graph_pesato', { //applicazione dell'algoritmo di Dijkstra
   sourceNode:source,
   relationshipWeightProperty: 'costo' //property utilizzata come peso
})
YIELD sourceNode, targetNode, path, totalCost
WHERE sourceNode<>targetNode AND (gds.util.asNode(sourceNode).componentId=gds.util.asNode(target
Node).componentId) //nodo sorgente e target devono essere diversi e il loro component id deve essere
uguale
WITH
   sourceNode, targetNode,
   totalCost AS length, //come lunghezza viene selezionato il costo del percorso
   nodes(path) AS path, //lista dei nodi attraversati
   gds.util.asNode(sourceNode).score AS sourceScore, //score del nodo sorgente
   gds.util.asNode(targetNode).score AS targetScore  //score del nodo target
RETURN gds.util.asNode(sourceNode).id_cliente AS sourceNodeId, //id del nodo sorgente
   gds.util.asNode(targetNode).id_cliente AS targetNodeId, //id del nodo target
   sourceScore,
   targetScore,
CASE // se lo score del nodo target è positivo o nullo, il grado di influenza è positivo
   WHEN gds.util.asNode(targetNode).score >=0 OR
        gds.util.asNode(targetNode).score IS NULL THEN sourceScore/length
   ELSE -(sourceScore/length)  // altrimenti il grado di influenza è negativo
END AS grado_influenza
```

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 6189679 | 6174304 | 0.5 | *null* | 0.5 |
| 6189679 | 6414701 | 0.5 | *null* | 0.25 |
| 6189679 | 6770736 | 0.5 | *null* | 0.25 |

L'output restituito è uguale alla versione precedente, perché la forza delle relazioni è 2 e la distanza tra i nodi rimane la stessa.

## Implementation phase

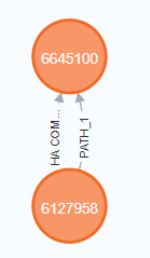
Gli id dei tre clienti più soddisfatti sono:

```
MATCH (c)
WHERE EXISTS(c.score ) // prendiamo clienti che hanno un valore di score
RETURN c.score, c.id_cliente
ORDER BY c.score DESC // ordiniamo i clienti in base allo score partendo dal più alto
LIMIT 3 // restituiamo solo i primi 3
```

| c.score | c.id_cliente |
|---|---|
| 1.0 | 6127958 |
| 1.0 | 6131670 |
| 1.0 | 8586145 |

Iniziamo utilizzando la prima soluzione proposta.
La rete di influenza del cliente 6127958 è la seguente:

```
MATCH (source:Cliente{id_cliente:6127958})
CALL gds.allShortestPaths.dijkstra.stream('client_graph', {sourceNode:source})
YIELD sourceNode, targetNode, path
RETURN  sourceNode, targetNode, path
```



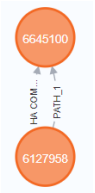

Calcoliamo il grado di influenza del nodo connesso applicando l'algoritmo presentato in precedenza.

```
MATCH (source:Cliente{id_cliente: 6127958})
CALL gds.allShortestPaths.dijkstra.stream('client_graph', {sourceNode:source})
YIELD sourceNode, targetNode, path
WHERE sourceNode<>targetNode
WITH
   sourceNode, targetNode,
   length(path) AS length,
   nodes(path) AS path,
   gds.util.asNode(sourceNode).score AS sourceScore,
   gds.util.asNode(targetNode).score AS targetScore
RETURN gds.util.asNode(sourceNode).id_cliente AS sourceNodeId,
   gds.util.asNode(targetNode).id_cliente AS targetNodeId,
   sourceScore,
   targetScore,
CASE
   WHEN gds.util.asNode(targetNode).score >=0 OR
```

```
        gds.util.asNode(targetNode).score IS NULL THEN sourceScore/length
    ELSE -(sourceScore/length)
END AS grado_influenza
```

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 6127958 | 6645100 | 1.0 | *null* | 1.0 |

Applichiamo l'altra versione.
La rete di influenza del cliente 6127958 è uguale alla precedente:



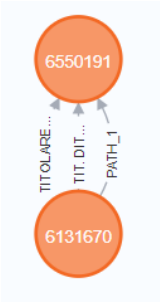

Calcoliamo il grado di influenza del nodo connesso.

```
MATCH (source:Cliente{id_cliente: 6127958})
CALL gds.allShortestPaths.dijkstra.stream('graph_pesato', {
    sourceNode:source,
    relationshipWeightProperty: 'costo'
})
YIELD sourceNode, targetNode, path, totalCost
WHERE sourceNode<>targetNode AND (gds.util.asNode(sourceNode).componentId=gds.util.asNode(target
Node).componentId)
WITH
    sourceNode, targetNode,
    totalCost AS length,
    nodes(path) AS path,
    gds.util.asNode(sourceNode).score AS sourceScore,
    gds.util.asNode(targetNode).score AS targetScore
RETURN gds.util.asNode(sourceNode).id_cliente AS sourceNodeId,
    gds.util.asNode(targetNode).id_cliente AS targetNodeId,
    sourceScore,
    targetScore,
CASE
    WHEN gds.util.asNode(targetNode).score >=0 OR
        gds.util.asNode(targetNode).score IS NULL THEN sourceScore/length
    ELSE -(sourceScore/length)
END AS grado_influenza
```

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 6127958 | 6645100 | 1.0 | *null* | 1.0 |

Il grado di influenza è uguale al precedente, perché la relazione ha peso 2, quindi la distanza tra i due nodi è la stessa.

La rete di influenza del cliente 6131670 è uguale per entrambe le versioni:




Il grado di influenza delle due versioni è lo stesso perché il peso della relazione è 2:

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 6131670 | 6550191 | 1.0 | *null* | 1.0 |

Questo è il sottografo del terzo cliente (id:8586145):


Il grado di influenza delle due versioni è lo stesso perché il peso della relazione è 2:

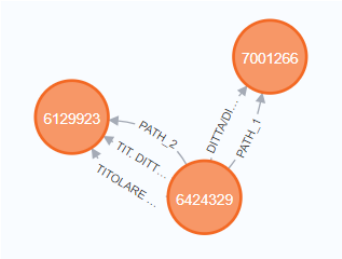
| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 8586145 | 11286028 | 1.0 | *null* | 1.0 |

Visualizzazione dei clienti insoddisfatti:

```
MATCH (c)
WHERE EXISTS(c.score ) // prendiamo clienti che hanno un valore di score
RETURN c.score, c.id_cliente
ORDER BY c.score DESC // ordiniamo i clienti in base allo score partendo dal più basso
LIMIT 3 // restituiamo solo i primi 3
```

| c.score | c.id_cliente |
|---|---|
| -1.0 | 6424329 |
| -1.0 | 10562699 |
| -1.0 | 6437993 |

Questo è il sottografo del cliente 6424329:




Gradi di influenza della prima versione:

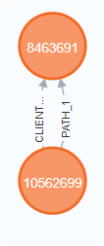
| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 6424329 | 7001266 | -1.0 | *null* | -1.0 |
| 6424329 | 6129923 | -1.0 | *null* | -1.0 |

Gradi di influenza risultanti dalla seconda versione:

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 6424329 | 6129923 | -1.0 | *null* | -1.0 |
| 6424329 | 7001266 | -1.0 | *null* | -0.5 |

Nella seconda versione il nodo 7001266 è influenzato meno dal nodo sorgente perché la loro relazione ha peso 1, quindi la loro distanza raddoppia.

Questo è il sottografo del cliente 10562699:




Gradi di influenza della prima versione:

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 10562699 | 8463691 | -1.0 | *null* | -1.0 |

Grado di influenza risultanti dalla seconda versione:

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 10562699 | 8463691 | -1.0 | *null* | -0.5 |

La relazione ha peso 1, quindi nella seconda versione il nodo è meno influenzato dal nodo sorgente.

Questo è il sottografo del cliente 6437993:


Il grado di influenza risultante è uguale per entrambe le versioni perché la relazione ha peso 2.

| sourceNodeId | targetNodeId | sourceScore | targetScore | grado_influenza |
|---|---|---|---|---|
| 6437993 | 7905569 | -1.0 | *null* | -1.0 |

## 4.3 Obiettivo 2

### Development phase

#### Soluzione 1

Per calcolare il livello di gradimento delle filiali bisogna partire dai livelli di soddisfazione dei clienti e dal grado di influenza.

Applichiamo quindi l'algoritmo di calcolo del grado di influenza, sviluppato in precedenza, considerando come nodi sorgente tutti i nodi che hanno uno score. Se un nodo è influenzato da più nodi sorgente queste influenze verranno sommate e poi divise per il numero di nodi da cui è stato influenzato, cioè viene fatta una media delle influenze.

```
//calcolo grado di influenza di ogni nodo considerando tutti i possibili nodi sorgenti
MATCH (c:Cliente)
WHERE exists (c.score)
WITH DISTINCT collect(c.id_cliente) as ids //mettiamo in una lista gli id
UNWIND ids AS sourceId // per ogni id viene eseguito il codice sotto
MATCH (source:Cliente{id_cliente: sourceId})
CALL gds.allShortestPaths.dijkstra.stream('client_graph', {sourceNode:source})
YIELD sourceNode, targetNode, nodeIds, path
WHERE sourceNode<>targetNode
WITH
   sourceNode, targetNode,
   length(path) AS length, //calcolo lunghezza percorso
   gds.util.asNode(sourceNode).score AS sourceScore,
   gds.util.asNode(targetNode).score AS targetScore

WITH gds.util.asNode(sourceNode).id_cliente AS sourceNodeName,
   gds.util.asNode(targetNode).id_cliente AS targetNodeName,
   sourceScore,
   targetScore,
   targetNode,
```

```
CASE
  WHEN gds.util.asNode(targetNode).grado_influenza IS NULL //se non è presente la property
grado_influenza
  THEN (sourceScore/length)  //lo score del nodo sorgente viene diviso per la distanza tra i due nodi
  ELSE (sourceScore/length)+gds.util.asNode(targetNode).grado_influenza  //se la property è già presente la
nuova influenza viene sommata alle precedenti
END AS grado_inf,

CASE
  WHEN gds.util.asNode(targetNode).grado_influenza IS NULL // se nodo non ha ancora un grado di
influenza
  THEN 1  //numero di nodi che lo influenzano è 1
  ELSE 1+gds.util.asNode(targetNode).influenze  END AS influenze //altrimenti si somma 1 al numero di
influenze

SET gds.util.asNode(targetNode).grado_influenza=grado_inf //viene fissato il valore della property
grado_influenza

SET gds.util.asNode(targetNode).influenze=influenze //viene fissato il valore della property influenze
```

Ora per ogni nodo influenzato abbiamo il grado di influenza totale e il numero di influenze.
Calcoliamo il grado di influenza medio e settiamo il corretto segno.

```
//calcolo media del grado di influenza
MATCH (c:Cliente)
WHERE EXISTS (c.grado_influenza)
WITH
CASE
   WHEN c.score >= 0 OR
       c.score IS NULL THEN c.grado_influenza/c.influenze //se lo score è >=0 o nullo, il valore di influenza è
positivo
   ELSE -(c.grado_influenza/c.influenze) // altrimenti è negativo
END AS grado_influenza, c
SET c.influenza_media=grado_influenza  //viene fissato il valore della property influenza_media
```

L'influenza media viene sommata allo score per ottenere il grado di soddisfazione finale. Lo score di soddisfazione è sempre compreso tra -1 e 1 quindi riportiamo i valori di soddisfazione dentro l'intervallo.

```
//calcolo soddisfazione=score+influenza media
MATCH (c:Cliente)
WHERE EXISTS (c.score) OR EXISTS (c.influenza_media) // prendiamo i nodi che hanno almeno un valore di
score o di influenza media
WITH
CASE
   WHEN c.influenza_media  IS NULL THEN c.score // se è presente solo il valore di score si tiene quello
   WHEN c.score IS NULL THEN  c.influenza_media // se è presente solo il valore di influenza_media si tiene
quello
   ELSE c.score+c.influenza_media //altrimenti score e influenza_media si sommano
END AS soddisfazione, c
SET c.soddisfazione=soddisfazione //viene fissato il valore della property soddisfazione
```

```
// i valori di soddisfazione maggiori di 1 vengono riportati a 1
MATCH (c:Cliente)
WHERE c.soddisfazione>1
SET c.soddisfazione=1

// i valori di soddisfazione minori di -1 vengono riportati a -1
MATCH (c:Cliente)
WHERE c.soddisfazione< -1
SET c.soddisfazione=-1
```

**Soluzione 2**

Scrittura soddisfazione per ogni nodo Cliente, considerando il costo del percorso e non la sua lunghezza.

```
//calcolo grado di influenza di ogni nodo considerando tutti i possibili nodi sorgenti
MATCH (c:Cliente)
WHERE exists (c.score)
WITH DISTINCT collect(c.id_cliente) as ids //mettiamo in una lista gli id
UNWIND ids AS sourceId // per ogni id viene eseguito il codice sotto
MATCH (source:Cliente{id_cliente: sourceId})
CALL gds.allShortestPaths.dijkstra.stream('graph_pesato', { //applicazione dell'algoritmo di Dijkstra Single Source
   sourceNode:source,
   relationshipWeightProperty: 'costo' //property utilizzata come peso
})
YIELD sourceNode, targetNode, totalCost
WHERE sourceNode<>targetNode AND (gds.util.asNode(sourceNode).componentId=gds.util.asNode(targetNode).componentId) //nodo sorgente e target devono essere diversi e il loro component id deve essere uguale
WITH
   sourceNode, targetNode,
   totalCost AS cost, //come lunghezza viene selezionato il costo del percorso
   gds.util.asNode(sourceNode).score AS sourceScore //score del nodo sorgente
WITH
 targetNode,
CASE
  WHEN gds.util.asNode(targetNode).grado_influenza IS NULL //se non è presente la property grado_influenza
  THEN (sourceScore/cost)  //lo score del nodo sorgente viene diviso per la distanza tra i due nodi
  ELSE (sourceScore/cost)+gds.util.asNode(targetNode).grado_influenza  //se la property è già presente la nuova //influenza viene
END AS grado_inf,
CASE
  WHEN gds.util.asNode(targetNode).influenze IS NULL // se nodo non ha ancora un grado di influenza
  THEN 1  //numero di nodi che lo influenzano è 1
  ELSE 1+gds.util.asNode(targetNode).influenze  END AS influenze //altrimenti si somma 1 al numero di influenze
SET gds.util.asNode(targetNode).grado_influenza=grado_inf //viene fissato il valore della property grado_influenza
SET gds.util.asNode(targetNode).influenze=influenze //viene fissato il valore della property influenze
```

```
//calcolo media del grado di influenza
MATCH (c:Cliente)
WHERE EXISTS (c.grado_influenza)
WITH
CASE
   WHEN c.score >= 0 OR
      c.score IS NULL THEN c.grado_influenza/c.influenze //se lo score è >=0 o nullo, il valore di influenza è
positivo
   ELSE -(c.grado_influenza/c.influenze) // altrimenti è negativo
END AS grado_influenza, c
SET c.influenza_media=grado_influenza //viene fissato il valore della property influenza_media
```

L'influenza media viene sommata allo score per ottenere il grado di soddisfazione finale. Lo score di soddisfazione è sempre compreso tra -1 e 1 quindi riportiamo i valori di soddisfazione dentro l'intervallo.

```
//calcolo soddisfazione=score+influenza media
MATCH (c:Cliente)
WHERE EXISTS (c.score) OR EXISTS (c.influenza_media) // prendiamo i nodi che hanno almeno un valore di
score o di influenza media
WITH
CASE
   WHEN c.influenza_media  IS NULL THEN c.score // se è presente solo il valore di score si tiene quello
   WHEN c.score IS NULL THEN  c.influenza_media // se è presente solo il valore di influenza_media si tiene
quello
   ELSE c.score+c.influenza_media  //altrimenti score e influenza_media si sommano
END AS soddisfazione, c
SET c.soddisfazione=soddisfazione //viene fissato il valore della property soddisfazione

//i valori di soddisfazione maggiori di 1 vengono riportati a 1
MATCH (c:Cliente)
WHERE c.soddisfazione>1
SET c.soddisfazione=1


//i valori di soddisfazione minori di -1 vengono riportati a -1
MATCH (c:Cliente)
WHERE c.soddisfazione< -1
SET c.soddisfazione=-1
```

I procedimenti successivi sono stati applicati a entrambe le versioni.
Ora per entrambe le versioni è necessario creare un in-memory graph con i nodi Filiale e i nodi Clienti connessi.

Come relazioni vengono presi i nodi Cliente che hanno la property soddisfazione che sono gli unici utili per valutare il livello di soddisfazione di una filiale.

Estimation creazione grafo:

```
//memory estimation grafo delle filiali
CALL gds.graph.create.cypher.estimate(
'MATCH (n)
WHERE n:Filiale OR n:Cliente  RETURN id(n) AS id,labels(n) AS labels',
```

```
'MATCH (f:Filiale)-[:FILIALE_DI]->(n:Cliente)
WHERE exists(n.soddisfazione) RETURN id(f) AS source, id(n) AS target ')
YIELD requiredMemory,bytesMin, bytesMax,  nodeCount,  relationshipCount
```

| requiredMemory | bytesMin | bytesMax | nodeCount | relationshipCount |
|---|---|---|---|---|
| "[107 MiB ... 120 MiB]" | 112341696 | 126237024 | 3505346 | 1752673 |

```
// creazione grafo delle filiali
 CALL gds.graph.create.cypher( 'graph_filiali',
'MATCH (n)
WHERE n:Filiale OR n:Cliente  RETURN id(n) AS id,labels(n) AS labels', // prendiamo i nodi Filiale e Cliente
'MATCH (f:Filiale)-[:FILIALE_DI]->(n:Cliente)
WHERE exists(n.soddisfazione) RETURN id(f) AS source, id(n) AS target ') // i nodi Cliente devono avere un valore di soddisfazione
YIELD graphName, nodeCount, relationshipCount
```

| graphName | nodeCount | relationshipCount |
|---|---|---|
| "graph_filiali" | 3507006 | 74063 |

Impieghiamo l'algoritmo Degree Centrality per calcolare quanti clienti sono associati a ciascuna filiale. Degree centrality misura il numero di archi entranti o uscenti (o entrambi) da un nodo. Può essere applicato a grafi pesati o non pesati.

Questo algoritmo viene utilizzato per scrivere la property n_clienti nei nodi del grafo appena creato. In questo modo potremo conoscere il numero di clienti per ogni filiale.

Memory estimation:

```
// memory estimation dell'algoritmo degree centrality
CALL gds.degree.write.estimate('graph_filiali', // grafo impiegato
 { writeProperty: 'n_clienti' }) // nome della property che verrà scritta
YIELD requiredMemory,bytesMin, bytesMax,  nodeCount,  relationshipCount
```

| requiredMemory | bytesMin | bytesMax | nodeCount | relationshipCount |
|---|---|---|---|---|
| "64 Bytes" | 64 | 64 | 3507006 | 74063 |

Esecuzione dell'algoritmo:

```
// applicazione algoritmo
CALL gds.degree.write('graph_filiali', // grafo impiegato
  { writeProperty: 'n_clienti' }) // nome della property che verrà scritta
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore, nodePro
pertiesWritten
```

| minimumScore | meanScore | nodePropertiesWritten |
|---|---|---|
| 0.0 | 0.021118642559815527 | 3507006 |

Per ogni nodo filiale abbiamo il numero dei suoi clienti che hanno espresso un grado di soddisfazione. Bisogna calcolare lo score di soddisfazione della filiale basandoci sulla soddisfazione dei clienti della stessa

filiale.

```
//calcolo score totale di ogni filiale
MATCH (f:Filiale)
WITH DISTINCT collect(f.id_filiale) as ids //mettiamo in una lista gli id
UNWIND ids AS sourceId // per ogni id viene eseguito il codice sotto
MATCH (source:Filiale{id_filiale: sourceId})
CALL gds.allShortestPaths.dijkstra.stream('graph_filiali', {sourceNode:source})
YIELD sourceNode, targetNode
WHERE sourceNode<>targetNode and exists (gds.util.asNode(targetNode).soddisfazione) // nodo di partenza e di arrivo devono essere diversi e target deve avere un valore di soddisfazione
WITH
   sourceNode,
   gds.util.asNode(sourceNode).score AS sourceScore,
   gds.util.asNode(targetNode).soddisfazione AS targetSoddisfazione
WITH
   sourceNode,
CASE
  WHEN sourceScore IS NULL //se non è presente la property
  THEN targetSoddisfazione
  ELSE (sourceScore+ targetSoddisfazione) //se la property è già presente, il valore di soddisfazione sommato allo score della filiale
END AS score_filiale

SET gds.util.asNode(sourceNode).score= score_filiale // viene aggiornato lo score della filiale
```

Ora per ogni filiale abbiamo lo score totale che va diviso per il numero di clienti (n_clienti).

```
//divisione score totale per numero di clienti della filiale
match (f:Filiale)
where f.n_clienti>0
set f.score_filiale=f.score/f.n_clienti
```

## Implementation phase

Individuiamo le filiali più virtuose.

```
MATCH (f:Filiale)
WHERE EXISTS(f.score_filiale )
RETURN f.score_filiale, f.id_filiale
ORDER BY  f.score_filiale DESC //ordino le filiali in ordine decrescente di score
LIMIT 3
```

La prima e seconda soluzione restituiscono le stesse filiali e lo stesso score:

| score_filiale | filiale |
|---|---|
| 1.0 | 4154 |
| 1.0 | 18837 |
| 1.0 | 18265 |

Individuiamo le filiali più critiche:

```
MATCH (f:Filiale)
WHERE EXISTS(f.score_filiale )
RETURN f.score_filiale, f.id_filiale
ORDER BY  f.score_filiale  //ordino le filiali in ordine crescente di score
LIMIT 3
```

Risultato della prima soluzione:

| score_filiale | id_filiale |
|---|---|
| -0.88 | 18538 |
| -0.88 | 18789 |
| -0.86 | 18340 |

Risultato della seconda soluzione:

| f.score_filiale | f.id_filiale |
|---|---|
| -0.88 | 18538 |
| -0.88 | 18789 |
| -0.75 | 18529 |

Le due filiali più critiche sono le stesse in entrambe le versioni e hanno lo stesso score. Viene invece individuata una terza filiale più critica diversa.

# 5. Conclusioni

Come risultato della text analytics sono stati individuati i clienti più soddisfatti (20200569, 23007684, 13076375) e meno soddisfatti (7117139, 22996262, 11478292). Il modello di machine learning che classifica con più accuratezza le riposte aperte è basato su Bernoulli Naive Bayes.

Come frutto della graph analytics si è visto che i clienti più soddisfatti (6127958, 6131670, 8586145) influenzano solo 1 nodo. I clienti meno soddisfatti sono 6424329, 10562699, 6437993. Alcuni di questi influenzano due nodi. In generale i sottografi di influenza sono ristretti.

Per quanto riguarda le filiali le due versioni riportano le stesse due filiali più critiche (18538, 18789) con anche lo stesso score di soddisfazione. La terza più critica invece è diversa (18340 e 18529).