UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

CORSO DI LAUREA IN INFORMATICA

Costruzione di una dashboard intelligente per la composizione di Tweet in ambito Cultural Heritage per dispositivi Android

Relatore:
Prof. Riccardo Martoglia

Laureanda:
Chiara Boni
Matr. 97120

ANNO ACCADEMICO 2018/2019

PAROLE CHIAVE:

Dispositivi mobili

Machine Learning

Text Classification

Cultural Heritage

Twitter API

INDICE

Introduzione

Distinguersi sui *social network* è fondamentale.

In un mercato globalizzato e in continua espansione come quello attuale, ogni azienda investe innumerevoli risorse per avere una visibilità duratura ed un forte impatto su quelle che sono le comunità online.

Questo progetto è nato, dunque, dalla necessità di sfruttare al meglio la vetrina globale che l'uso dei media sta concedendo.

Più nel dettaglio, si pone l'obiettivo di intervenire con strategie pubblicitarie più efficienti sulla piattaforma che meglio si addice alla gestione del testo: Twitter.

Si prende come riferimento il settore culturale, di cui fanno parte musei e mostre permanenti, con lo scopo di implementare una tecnologia in grado di dare gli strumenti necessari, ad ogni manager, per scrivere il messaggio che avrà il più alto impatto mediatico.

Tutto questo è reso possibile mediante lo studio approfondito dei messaggi precedenti – e del loro andamento sul pubblico – per creare un insieme di suggerimenti atti a migliorare il messaggio che si ha intenzione di condividere.

Questo progetto è stato costruito per poter essere utilizzato su dispositivi mobili, in quanto vuole essere accessibile per la maggior parte dei suoi utilizzatori, senza rimanere però esterno alle tecnologie utilizzate attualmente.

La scelta del sistema operativo Android, invece, è stata compiuta in base alla disponibilità personale degli strumenti di progettazione.

I compiti svolti dall'applicazione creata sono vari:

- L'utente potrà accedere ed inserire la propria idea di messaggio mediante l'apposita schermata di avvio;
- Avverrà l'effettiva valutazione mediante l'utilizzo di un apposito classificatore;
- Sarà possibile navigare attraverso consigli e suggerimenti resi disponibili per migliorare l'effetto sul pubblico;
- Infine, una volta ritenuti soddisfatti dall'esito della ricerca, è possibile accedere a Twitter, mediante un account esistente, e pubblicare effettivamente il messaggio.

La configurazione dell'elaborato segue e si ramifica in base all'operato cronologico di progettazione.
Si apre con la parte di discussione del caso di studio: la creazione di una struttura basata su una comunicazione tra *client* e *server*; una introduzione sui concetti di base per il *Machine Learning* e, infine, un approfondimento su analisi testuali e classificatori.
Sarà, inoltre, presente l'elenco dettagliato delle tecnologie che sono state impiegate per lo sviluppo dell'applicazione.
La seconda parte prevede un approccio più mirato su quella che è stata la progettazione vera e propria del programma: dal recupero della documentazione di studio, all'implementazione concreta delle tecnologie.
Si conclude con alcune riflessioni per eventuali sviluppi futuri.

Parte I

Il caso di studio

CAPITOLO 1

Concetti teorici

1.1 Il caso di studio

Questo progetto nasce dall'esigenza di realizzare una piattaforma in grado di rispondere, in maniera accurata, a quelli che sono i bisogni di un settore in forte crescita, come quello del *cultural heritage*, composto quindi da musei ed esposizioni, che vogliono mettersi in contatto direttamente con il loro pubblico.
Ogni museo deve, inoltre, costantemente confrontarsi con i possibili *post* che concorrenti affini possono pubblicare.
Occorre, dunque, una strategia per poter affinare la propria metodologia di pubblicità.
Vengono introdotti nuovi usi a tecnologie già conosciute, come algoritmi di *machine learning*, in modo da adattarle ad un'analisi di tipo testuale.

Lo studio inizia dalla scelta dei musei sui quali occorre concentrarsi: sono stati selezionati **25 musei**, tra i più conosciuti a livello globale, e sono stati suddivisi in **6 gruppi** in base alla loro popolarità. Questa divisione viene effettuata in modo da poter garantire confronti più equilibrati tra musei concorrenti, come si può notare dalla Tabella 1.1 sottostante. Si ricorda che il numero di *followers* è aggiornato in base al periodo preso in esame durante gli studi preliminari.
Ogni museo ha concesso di analizzare un numero di messaggi che si aggira a 800, per un totale di circa 19.527 tweet.
Il *successo* di un messaggio è un fattore che viene deciso anche in base alla vastità del pubblico considerato: un esempio concreto viene descritto da un tweet con 10 *like*, che può essere considerato un buon risultato su un pubblico sull'ordine delle decine, ma risulta invece un esito deludente su uno spettro di lettori dell'ordine del milione.

Proprio per questa possibilità di vaghezza, si è deciso di concentrare l'analisi del messaggio per gruppi e, soprattutto, andando ad eliminare il contesto del mittente, facendo emergere solo le caratteristiche proprie del tweet.

Table 1: Museum groups

Group 1 (G1)	# Followers	Group 2 (G2)	# Followers	Group 3 (G3)	# Followers
@MuseumModernArt	5,120,000	@britishmuseum	1,560,000	@CentrePompidou	970,000
@Tate	4,500,000	@vangoghmuseum	1,330,000	@NationalGallery	887,000
@metmuseum	3,680,000	@MuseeLouvre	1,250,000	@museofrodakahlo	847,000
@Guggenheim	3,350,000	@GettyMuseum	1,250,000	@MuseeOrsay	610,000
@saatchi_gallery	2,800,000	@museodelprado	1,180,000		
Group 4 (G4)	**# Followers**	**Group 5 (G5)**	**# Followers**	**Group 6 (G6)**	**# Followers**
@mfaboston	332,000	@visitmuve_it	88,000	@MuseoEgizio	21,600
@museiincomune	264,000	@museupicasso	65,000	@Uffizi	19,900
@philamuseum	247,000	@mart_museum	64,000	@MUSE_Trento	12,600
@maspmuseum	245,000				
@ngadc	216,000				
@Museo_MAXXI	190,000				

Tabella 1.1 - Gruppi di musei

Non vengono analizzate tutte le componenti "esterne" al messaggio, come l'influenza del mittente o il numero di seguaci, ma si devono tenere in considerazione tutte quelle che sono le conseguenze che il messaggio stesso comporta.
Si sottolineano soprattutto le componenti di tipo numerico: come ad esempio il numero di media, di *hashtag* o di menzioni che contiene.

La fase successiva prevede una classificazione binaria dei messaggi, divisi per gruppi. In questo primo studio si definiscono due macro-categorie: **GOOD** o **BAD**.
Esse hanno lo scopo di distinguere qual è stato l'andamento del messaggio in base al numero di like che esso ha ricevuto: intuitivamente, se riceve un numero di like vicino a quello che è il valore massimo, all'interno del gruppo, si può affermare la presenza di un andamento positivo.
Il ragionamento sarà analogo per quanto riguarda la valutazione negativa.
Si tenta di eliminare la massima ambiguità possibile: devono essere eliminati i messaggi per i quali è difficile dare un esito anche per gli esseri umani.

Per fare questo, è stato introdotto un meccanismo di gestione a *code* per i due insiemi: si considera il 20% dei messaggi che hanno ricevuto il massimo numero di like come il gruppo di riferimento per definire un messaggio GOOD.
Analogamente si considera il 20% anche per i messaggi con il minor numero di like, che comporranno la categoria BAD.
Una volta assegnate le varie etichette ai messaggi da analizzare, è necessario automatizzare il processo di classificazione ed estrarne un vero e proprio modello.
È presente inoltre una fase di *scelta* delle caratteristiche del tweet, secondo uno studio delle correlazioni che possono esserci tra di esse, in modo da minimizzare il numero di *features* da osservare e, al tempo stesso, massimizzarne il risultato.
Questo sistema è basato su una classificazione che richiede meccanismi di *machine learning*.

Una volta costruito il modello, esso dovrà essere messo in comunicazione con l'utente finale, in modo da poter dare suggerimenti per costruire un messaggio migliore per il proprio pubblico.
Questo è possibile costruendo un canale di comunicazione tra il *server,* che mantiene i dati e il modello, e il *client* che viene rappresentato dall'applicazione in uso dall'utente.
I suggerimenti si basano su quelle che sono le caratteristiche dei messaggi simili a quello inviato, ma con esito differente.
Ne verranno mostrati più di uno, in modo da dare la scelta all'utente su quale sia la strategia migliore per il proprio museo e la propria idea di pubblicità.
Tramite l'applicazione realizzata, infine, sarà possibile accedere a Twitter mediante credenziali esistenti e avere la possibilità di pubblicare effettivamente il messaggio oggetto di studio.

Ognuna delle fasi sopracitate verranno analizzate e studiate più nel dettaglio nel corso dell'elaborato seguente.

1.2 Architettura *Client-Server* e REST API

Questo tipo di architettura rappresenta il costrutto di base per la maggior parte delle applicazioni oggi disponibili.
Si struttura su un'interazione tra due componenti principali: il *client* e il *server*, che genericamente, operano su due macchine differenti collegate tra loro mediante una connessione di rete.
Presentano caratteristiche contrapposte in quanto svolgono ruoli ben definiti: i compiti del *client* saranno, infatti, quelli dedicati all'invio di richieste al *server*, con annessi servizi di formattazione del messaggio in modo da non generare errori, sia in fase di invio che di ricezione.
Dal lato diametralmente opposto, il *server*, è il soggetto che si deve occupare di ricevere, gestire e rispondere alle molteplici richieste che possono raggiungerlo.
Per tanto presentano ulteriori differenze anche dal punto di vista strutturale: un applicativo *client* può essere implementato da un qualsiasi generico programma che necessita di un servizio di rete, viene eseguito spesso in locale e non richiede specifico hardware aggiuntivo.
Per un applicativo *server* è possibile, invece, dover ricorrere ad apparecchiature più performanti in grado di gestire flussi di richieste a volte anche considerevoli.
Esso deve svolgere un insieme di operazioni che gli permettano di portare a buon fine l'esecuzione del servizio: spesso, dunque, gestisce anche collegamenti esterni a banche dati o si appoggia a sua volta a server più grandi.

Affinché il servizio richiesto dal client possa essere eseguito correttamente è necessario avere una **connessione** di rete ed un linguaggio comune di comunicazione, dunque un protocollo.
Per dare un esempio concreto occorre basarsi su quelli che sono i servizi più diffusi: in primo luogo la Posta Elettronica (SMTP), trasferimento di file da macchine diverse (FTP) e servizio di navigazione *World Wide Web* (HTTP).

L'approccio architetturale vincente, nel caso specifico di questo progetto, è stato quello di avvalersi dell'utilizzo di **Representational State Transfer** (REST) API, ovvero interfacce di programmazione basate sul protocollo HTTP (HyperText Transfer Protocol) che permettono di trasferire risorse con formattazioni specifiche.
È un servizio molto leggero, performante e accessibile mediante *browser*, tramite l'indirizzo URL della risorsa, quindi il programmatore ha la possibilità di verificare la correttezza del codice in modo molto semplice ed immediato.
REST è una piattaforma totalmente indipendente dalle caratteristiche di programmazione: può essere implementato da qualsiasi linguaggio.
Tendenzialmente vengono utilizzati però PHP, Python o Ruby, in quanto più adatti per applicazioni Web.
Questo tipo di comunicazione non si basa sullo scambio, dunque, di pagine ipertestuali come per esempio quelle con contenuto HTML (HyperText Markup Language), ma ha come oggetto dati in formato JSON o XML, come mostra la Figura 1.1 seguente. [1]
JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati, è indipendente dal linguaggio di programmazione con cui viene utilizzato e si basa su una struttura a coppia *chiave-valore*. [2]

Il grande successo di questo tipo di interfacce è facilmente riconducibile alla possibilità di quest'ultime di utilizzare i più comuni metodi HTTP per la comunicazione: GET e POST tra i primi ad essere citati.
Il metodo GET permette di richiedere dati ad una specifica risorsa, mentre il metodo POST permette di inviare dati da processare.

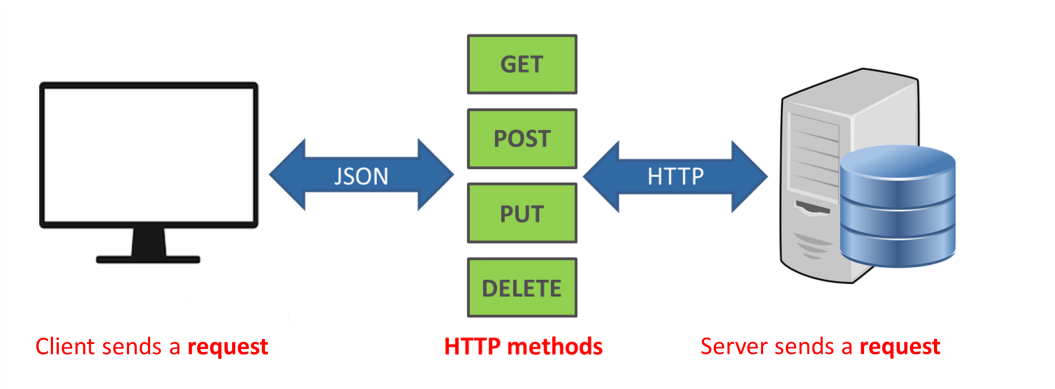

Figura 1.1 - Architettura REST API

Le REST API sono dette *stateless*, quindi prive di stato e di "memoria", in quanto ogni richiesta viene ricevuta e gestita in maniera indipendente e slegata dal contesto di qualsiasi altro servizio che può essere stato richiesto al server. [1]

È fondamentale aderire ad una sintassi precisa, quando si tratta di comunicare con un server, perché ogni messaggio è composto da specifiche sezioni.

Una **richiesta** si struttura da:

- Riga di richiesta, che specifica l'oggetto della domanda stessa (es. GET o POST);
- *Header*, che contiene informazioni riguardanti il messaggio;
- Riga vuota, per indicare la fine la fine della sezione dei meta-dati;
- *Body*, che rappresenta il corpo vero e proprio del messaggio.

Un messaggio di **risposta** si struttura, invece, con:

- Riga di stato, che rappresenta lo stato della richiesta, in caso di successo o di fallimento;
- *Header*, che contiene informazioni aggiuntive;
- Riga vuota, per indicare la fine dei meta-dati;
- *Body*, che è il corpo del messaggio.

1.3 Introduzione al Machine Learning

Il *Machine Learning* (ML) è uno dei settori più all'avanguardia su cui si sta tentando di investire, in quanto base dell'intelligenza artificiale.

Si vogliono creare strumenti in grado di gestire un **apprendimento automatico**, inteso come abilità della macchina stessa di imparare comportamenti ed analisi rispetto a dati che già possiede, senza che essa venga precedentemente programmata.

Fulcro è il concetto di "esperienza": lo strumento diventa conscio di dati pregressi e, da essi, dovrà apprendere informazioni e cambiare il modo di agire per orientarsi ad un miglioramento futuro delle prestazioni.

Lo scopo diventa quello di creare un generico algoritmo che sappia portare a termine esempi o compiti nuovi, mai affrontati prima, dopo che la macchina ha fatto esperienza su dati di apprendimento iniziali.

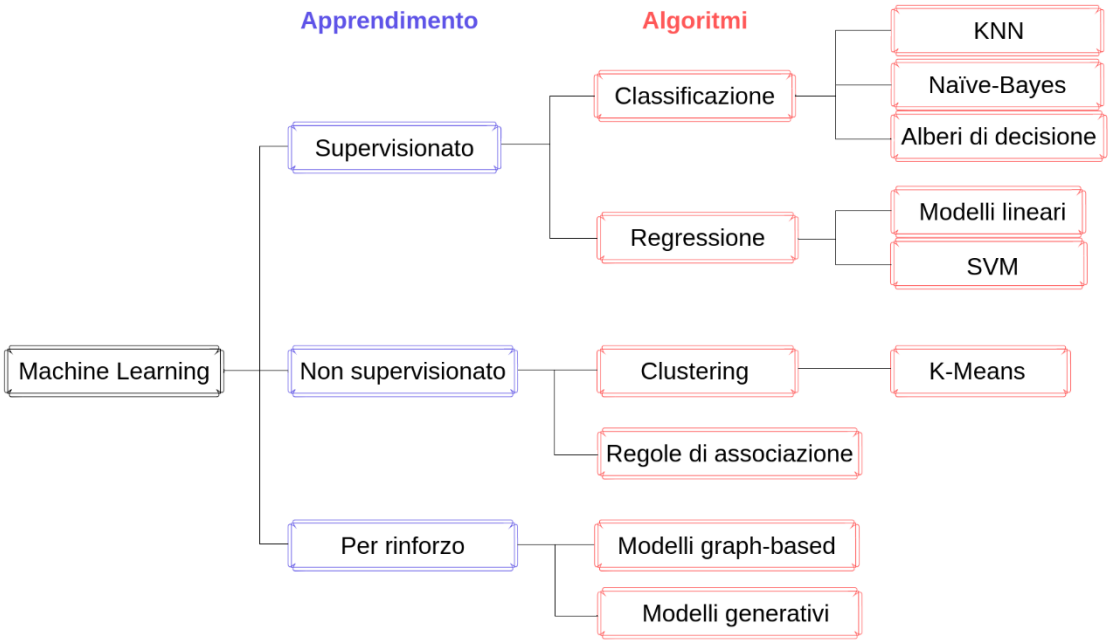

Figura 1.2 - Divisione tipi machine learning

Il Machine Learning è uno strumento estremamente complesso che si articola in molteplici sotto sezioni – mostrate nella Figura 1.2 precedente - che, in base al tipo di apprendimento implementato, trova applicazioni differenti.
Il ML si distingue, dunque, tramite:

1. apprendimento supervisionato
2. apprendimento non supervisionato
3. apprendimento per rinforzo.

L'**apprendimento supervisionato** (*Supervised Learning*) prevede due fasi di sviluppo dell'algoritmo: la prima si occupa di raccogliere dati di esempio ed etichettarli secondo parametri specifici del progetto.
Viene, poi, successivamente creato un modello istruito con il quale analizzare dati nuovi. Questo tipo di approccio deve essere basato su un dataset o campioni di dati, perché sarà da loro che si potranno raccogliere le informazioni per "istruire" la macchina.

Partendo da un insieme di dati iniziali è necessario riuscire ad ottenere due sottoinsiemi, rispettivamente composti da dati usati per allenare l'algoritmo (*training set*) e dati per valutare la precisione del modello creato in precedenza (*test set*).
Esistono due principali strategie per ottenere questa divisione:

- *Split*: tecnica che consiste nel decidere arbitrariamente una percentuale con la quale sezionare l'insieme dei dati (come mostra la Figura 1.3).
  Solitamente il *test set* occupa una porzione marginale rispetto al totale degli esempi, in quanto si aggira intorno al 20% di essi.
  È un metodo molto performante per dividere i dati, in quanto estremamente semplice e veloce, ma occorre un'analisi preliminare dei dati per essere in grado di consegnarne una porzione sufficiente al modello, tale che apprenda nel modo più accurato possibile.

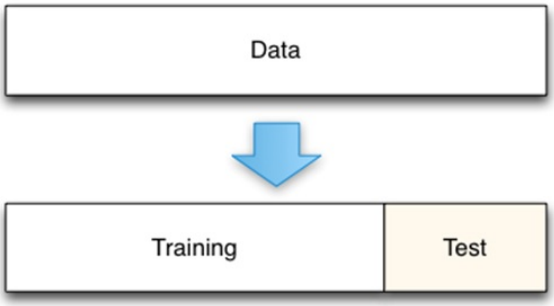

Figura 1.3 - Divisione "split" dei dati

- *K-fold cross-validation*: questo tipo di approccio è più complesso perché richiede la divisione dell'intero *data set* in K sottoinsiemi della stessa dimensione, detti *fold.* Ciclicamente si otterrà che ogni fold verrà utilizzato una volta come test set e le restanti K-1 volte come training, come viene illustrato nella Figura 1.4 sottostante.

  Viene utilizzato poi il valore medio, calcolato al termine del ciclo, per stimare la percentuale di accuratezza del modello.

  È una strategia più meticolosa della precedente, perché permette di utilizzare la totalità dei dati in entrambi i contesti e fornisce una stima più accurata.

  L'aspetto negativo è sicuramente sulla sua efficienza, in quanto viene considerato un metodo lento a causa dei K cicli che deve eseguire.

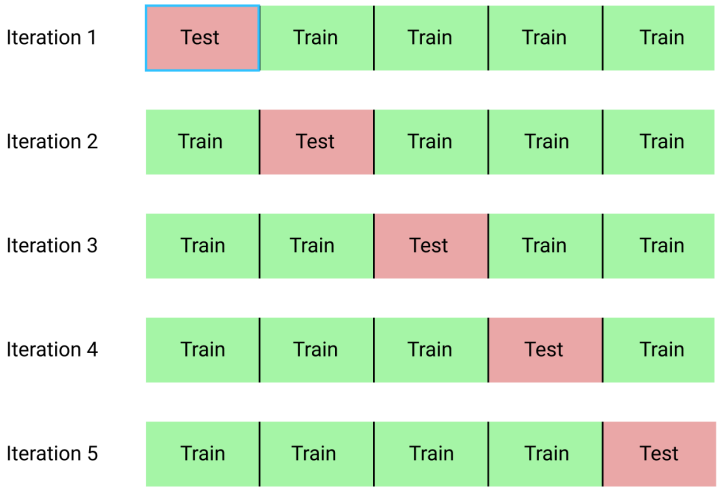

Figura 1.4 – Divisione con K-fold cross-validation

L'apprendimento di tipo supervisionato ritrova particolare uso per problematiche di discriminazione tra dati che possono essere discreti o continui.

Con dati discreti si parlerà di algoritmi di **classificazione**, quindi di tecniche di apprendimento nelle quali lo scopo è quello di riuscire a prevedere l'etichetta di dati nuovi, in base ad un modello istruito.

L'algoritmo ha il compito di fungere da "confine decisionale" e associare i dati alle rispettive classi, come mostrato in Figura 1.5 sottostante.

Gli algoritmi di **regressione**, invece, sono tecniche di apprendimento che meglio interpretano i dati di tipo continuo: si dispone di variabili *predittive* e di variabili continue che rappresentano il *target* dello studio.

Lo scopo principale è quello di trovare un'associazione tra le due e prevedere i target, quindi i risultati.

La tecnica di regressione più conosciuta è quella lineare, che riesce a definire una retta i cui punti hanno la distanza minore rispetto ai dati studiati, come specificato in Figura 1.6 sottostante. [3][4]

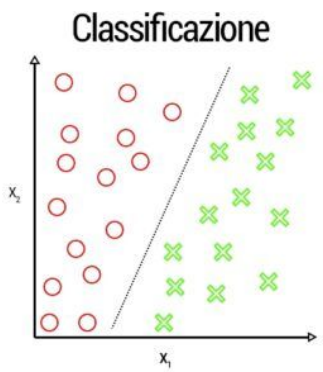

Figura 1.5 – Classificazione binaria

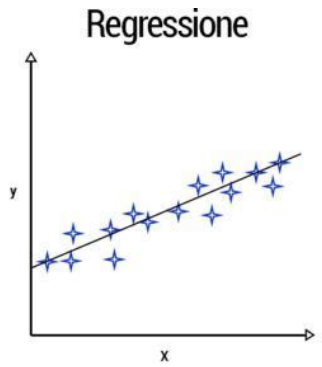

Figura 1.6 – Regressione lineare

L'**apprendimento non supervisionato** (*Unsupervised Learning*), invece, si basa un tipo di studio in cui i dati iniziali sono privi di etichette, quindi non è possibile dare una struttura di partenza al data set stesso.
L'acquisizione di informazioni avviene tramite inserimento di dati da input, che il sistema avrà il compito di analizzare per trovare schemi o relazioni tra essi.
Si parla di un'assenza di "supervisione", in quanto non è presente il corrispettivo valore di uscita del dato – che nel modello supervisionato era rappresentato dall'etichetta corrispondete – quindi non esiste un modello dal quale il sistema possa imparare, ma dovrà apprendere da sé stesso: le classi di associazione, che non sono note a priori, verranno acquisite nel corso dell'esecuzione.
Questo tipo di algoritmi elaborano i dati in base a similarità e differenze; ne sono un chiaro esempio i motori di ricerca, che ordinano i risultati in base alla loro pertinenza rispetto alla domanda posta dall'utente.

È possibile dividere i problemi di apprendimento senza supervisione in due macro aree: riguardanti il raggruppamento e le regole di associazione.
Gli algoritmi di **raggruppamento**, detti anche di *clustering*, vengono utilizzati per creare insieme di dati omogenei, mediante l'individuazione di caratteristiche simili da parte del sistema stesso.
Il concetto di similarità viene interpretato come idea di distanza tra gli elementi: è, quindi, fondamentale calibrare l'algoritmo su metriche apposite in base ai dati presenti.
I problemi riguardanti **regole di associazione**, base fondamentale per il *data mining*, invece hanno lo scopo di trovare relazioni, regole e schemi frequenti tra porzioni di dati.
Si individuano dipendenze tra i dati, per poi estrarne definizioni di classi che rappresentano i comportamenti tipici.
Solitamente sono impiegati nelle attività di marketing, in quanto permettono di identificare comportamenti comuni tra i consumatori, in base ai prodotti che acquistano. [4][5]

L'**apprendimento per rinforzo** (*Reinforcement Learning*), infine, rappresenta una metodologia di studio che si basa su continui *feedback* esterni che vengono inviati al sistema.

È una struttura complessa che ha lo scopo di mutare sé stessa rispetto all'ambiente in cui viene inserita, mediante una funzione di rinforzo, la quale ha il compito di misurare il grado di successo di un'azione rispetto ad un obiettivo predeterminato.

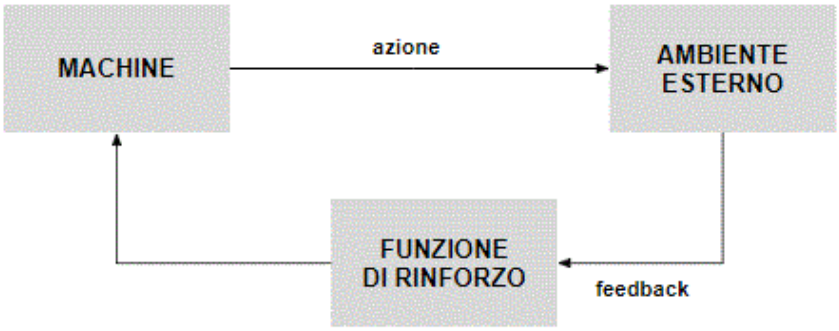

Figura 1.7 – Machine learning con rinforzo

Come mostrato dalla Figura 1.7, questo tipo di apprendimento necessita di una comunicazione continua tra l'ambiente esterno ed il sistema stesso.

Quest'ultimo prende il nome di **agente**, che meglio incorpora il concetto di entità software che interagisce con dati nuovi in input.

È necessario ricordare che la funzione di rinforzo fornisce una risposta circa il successo o meno dell'azione, ma non dà nessuna informazione su come aggiornare il comportamento dell'agente, che deve essere invece implementato durante la fase di apprendimento stessa.

Questo tipo di studio unisce i vantaggi delle due tipologie precedenti: non è legato ad un insieme di dati predefinito, come le tecnologie non supervisionate, che gli concede maggiore flessibilità, ma viene comunque aiutato nella fase di apprendimento dai feedback che riceve, similari alle etichette del surpervised learning, con anche la possibilità di analizzare situazioni non previste in precedenza dal progettista. [6]

1.4 Analisi del testo e classificatori

Come visto in precedenza, le tecniche di studio che si basano su algoritmi di machine learning possono avere numerose applicazioni, direttamente riconducibili a necessità reali.

Più nel dettaglio, si deve analizzare l'uso di queste tecnologie applicate allo studio e all'analisi del linguaggio naturale parlato.

Il **Natural Language Processing** (NLP) fa riferimento al trattamento informatico del linguaggio naturale, indipendentemente dall'approfondimento dell'analisi stessa.

È necessario definire la fondamentale differenza tra comunicazione naturale e formale: la prima fa riferimento a quella, verbale o scritta, tra esseri umani; mentre la seconda si riferisce al tipo di informazione che solo le macchine sono in grado di interpretare.

Il NLP si pone l'obiettivo di creare un collegamento tra le due, allo scopo di costruire sistemi informatici che possano interagire con gli esseri umani ed estrarre in maniera automatica informazioni da testi o altri media.

L'elaborazione del linguaggio naturale può avvenire a diversi livelli:

- *Lessico e morfologia*: avere la conoscenza delle parole di una lingua, della loro struttura ed organizzazione.
- *Sintassi*: composizione delle frasi in maniera corretta.
- *Semantica*: assegnamento di significato a componenti specifici.
- *Pragmatica*: abilità di utilizzare le frasi nei contesti più appropriati, dal punto di vista comunicativo.

Il NLP si suddivide in numerose sezioni che ricoprono, ognuna, un compito specifico.

Uno dei settori in forte crescita è la **classificazione del testo** (o *Text Classification*), ovvero un processo che permette di analizzare un testo ed essere in grado di categorizzarlo, inserendo particolari *tag*, in base al suo contesto.

Si dà particolare importanza a questo insieme di tecnologie, perché hanno applicazioni in ogni genere di settore: come quello del *sentiment analysis*, *spam detection* e *topic labelling*.

La gran parte delle informazioni disponibili sono, infatti, definite mediante dati non strutturati come email, chat, pagine web e social media; è necessario dunque essere in grado di ricevere in ingresso questi dati ed estrapolarne una struttura, dalla quale ricavare qualche tipo di insegnamento.

I classificatori di testo possono essere utilizzati per organizzare, strutturare e categorizzare ogni tipo di dato testuale.
Esistono numerosi approcci che rendono possibile la loro implementazione, anche se i più diffusi restano quelli basati su tecnologie di machine learning con apprendimento supervisionato, in quanto l'algoritmo è in grado di individuare associazioni tra porzioni di testo e una particolare etichetta di output.
La classificazione testuale con machine learning è più accurata e più scalabile di qualsiasi approccio manuale con sistemi di regole e viene, per ciò, prediletta.
Uno dei classificatori più conosciuti è quello che si basa sull'algoritmo **Naïve-Bayes**, che unisce la teoria della probabilità di Bayes con assunzioni semplici e spesso vere (da qui il nome "naive", primitivo, ingenuo).

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Figura 1.8 Formula di Bayes

Nella Figura 1.8 viene mostrata la formula che è alla base del teorema di Bayes: è possibile trovare la probabilità di un evento A, data quella dell'evento B.
L'evento B è definito *prova* ed A è l'ipotesi che bisogna verificare che accada.
Le assunzioni *naive* sono principalmente due:

- *Indipendenza* tra le caratteristiche: la presenza di non andrà ad influire sulla probabilità di un'altra di apparire.
- *Uguaglianza* di peso: ognuna delle proprietà contribuisce con la medesima importanza delle altre per il risultato.

È importante sottolineare che queste assunzioni risultino vere per la grande maggioranza dei casi, ma restano comunque delle restrizioni iniziali sul modello.

Per creare il modello del classificatore bisogna estendere la regola di Bayes: è necessario trovare la probabilità di un insieme di dati di input, per ogni possibile valore della classe, rappresentata dalla variabile Y, e scegliere il valore massimo che si ottiene in output.
Dalla Figura 1.9 sottostante è possibile ricavare P(y), chiamata anche *probabilità di classe* in quanto definisce la probabilità di ogni classe Y, e P (x_i | y), chiamata *probabilità condizionata* perché rappresenta la probabilità di ogni valore, data la classe Y.

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

Figura 1.9 – Massima probabilità

L'algoritmo Naïve-Bayes si struttura poi in numerosi sotto-algoritmi: tra i più conosciuti vengono ricordati il *Multinomial*, in cui la caratteristica utilizzata è la frequenza delle parole nel testo di un documento; la versione di *Bernoulli*, in cui si attua una discrezione di tipo booleano e, infine, la variante di *Gauss* che viene applicata con dati di tipo continuo.
Sono performanti e semplici da implementare, anche se si basano su supposizioni che agevolano la rappresentazione della realtà, come l'indipendenza tra le caratteristiche.
In generale restano approcci molto utilizzati negli ambiti di *sentiment analysis* e *spam filtering.* [7][8][9]

Un secondo algoritmo, ampiamente utilizzato per la classificazione, è il **K-Nearest Neighbors** (KNN) che si basa anch'esso su tecnologie di machine learning con apprendimento supervisionato.

Il fulcro di questo approccio è il concetto di distanza: un oggetto viene categorizzato in base ai suoi K elementi più vicini.

K è un intero positivo, solitamente dispari per evitare situazioni di perfetta parità, ed ha un valore non troppo elevato per evitare rallentamenti sulla performance.

Questo algoritmo viene chiamato *non parametrico*, in quanto non esistono predefinite assunzioni che possono influenzare la distribuzione dei dati: il modello può dipendere solamente dal data set sottostante.

Figura 1.10 - Algoritmo KNN

Come mostra la Figura 1.10, la classificazione si basa in grande parte sulla scelta del parametro K: esso è il responsabile dell'etichetta di output che riceve il dato e, anche, della complessità dell'algoritmo stesso.

Numerose ricerche hanno dimostrato che non esiste un valore predefinito di K tale che possa essere quello ottimale per tutti i data set: tendenzialmente, valori molto bassi creano molto *rumore*, capace di influenzare il risultato, ma aumentare il numero di vicini significa implementare un procedimento meno performante.

È necessario studiare il proprio data set, compiendo numerosi test, per decidere effettivamente il migliore valore di K al quale adattarsi.

L'algoritmo KNN si suddivide in tre macro fasi:

1. Fase di *training*, in cui lo spazio viene partizionato in base alle caratteristiche degli oggetti.
2. Calcolo della *distanza*, che è il passaggio principale.

   Gli elementi vengono rappresentati come vettori nello spazio ed occorre misurare la distanza per esplicitare quali sono i vicini. Esistono numerose metodologie per questo tipo di calcolo che, ovviamente, andranno ad influire sulle prestazioni stesse.

   Le più frequenti sono la distanza euclidea, che è la misura del segmento che unisce i due punti; quella di Manhattan, detta anche di Minkowski, secondo il quale la distanza tra due punti è la somma del valore assoluto delle differenze delle loro coordinate e, infine, la distanza di Hamming che viene applicata nei contesti delle stringhe e misura il numero di sostituzioni necessarie per convertire una stringa in un'altra.
3. *Classificazione*: l'elemento viene assegnato alla specifica classe C, in base al numero di suoi vicini, di cui è già nota la classificazione.

L'algoritmo KNN è più performante nei casi in cui si devono analizzare meno caratteristiche. Se dovesse aumentarne il numero, verosimilmente, aumenterebbe anche l'insieme di dati, andando a generare il problema del sovradimensionamento (*overfitting*), ovvero il fenomeno per cui il modello smette di essere valido per esempi generali e inizia a memorizzare gli aspetti del training set.

Il KNN è un algoritmo molto suscettibile all'overfitting data la *curse of dimensionality*, quindi man mano che lo spazio delle caratteristiche aumenta, il numero di configurazioni cresce esponenzialmente e, dato che il data set non varia, la quantità di configurazioni coperte da un'osservazione diminuisce.

In generale si ottiene una fase di training molto performante, a fronte però di una fase di testing più costosa rispetto ad altri classificatori, richiedente quindi più memoria. [10]

CAPITOLO 2

Tecnologie utilizzate

Questo capitolo ha lo scopo di illustrare le tecnologie che sono state impiegate per la realizzazione del progetto sopra descritto.
Ogni strumento inserito verrà accompagnato dalla spiegazione delle scelte implementative che hanno condotto a prediligerlo, rispetto ad altri.

2.1 Sistema Operativo

Un sistema operativo è un apparato software, composto quindi da molteplici programmi, in grado di riuscire nel compito di intermediario tra quello che è l'utente finale e il resto della complessa struttura dell'insieme hardware.
Esso gestisce l'operatività di base di un calcolatore, coordinando e gestendo le risorse hardware di processamento e memorizzazione. [11]

2.1.1 Android

Il sistema operativo Android è quello più largamente utilizzato basato sul kernel Linux e sviluppato da Google.
Si parla di sistema *embedded*, ovvero di un sistema incorporato a microprocessore progettato per una specifica applicazione e non riprogrammabile dall'utente finale, principalmente per dispositivi smartphone o tablet.
La ragione del suo enorme successo è data anche dalla politica *open source* sulla quale si basa: è facilmente possibile accedere al suo codice sorgente in modo da studiarlo e modificarlo.

Tutte le applicazioni Android sono *Java-based*, tali che, altre scritte in linguaggi differenti, devono richiamare codice Java tramite chiamate di sistema alla macchina virtuale incorporata.
Senza contare che gli utilizzatori di dispositivi Android possono ottenere i permessi di *root*, quindi di utente amministratore, ed accedere a funzioni avanzate, gestire direttamente la CPU e le applicazioni di sistema.
Esistono altri sistemi operativi in grado di competere con Android sul mercato, ma, le caratteristiche di open source e la facilità di implementazione in un qualsiasi altro sistema esterno, hanno fattosi di essere preferibile per progetti come quello proposto. [12]

2.1.2 SDK ed emulatori

Per poter sviluppare applicazioni è necessario introdurre il concetto di **SDK** (*Software Development Kit*), ovvero di pacchetto contente strumenti per la progettazione e documentazione di software.
Continuando la filosofia open source, Android rende disponibile il proprio codice sorgente per dare la possibilità di scaricare, dal sito ufficiale, questi strumenti per la programmazione di un qualsiasi tipo di applicazione.

Ogni kit software ha componenti di base come compilatori per la traduzione del codice sorgente in eseguibile, librerie standard dotate di interfacce pubbliche (API), varia documentazione per il linguaggio di programmazione con il quale l'SDK è stata progettata e, infine, sono presenti informazioni su licenze da utilizzare per distribuire i programmi creati con esse.
La libertà resta comunque massima: è possibile estendere il pacchetto software con ulteriori compilatori, editor di codice sorgente e programmi di vario tipo. [13]

Per installare le SDK di Android esistono varie metodologie: si possono scaricare direttamente dal sito ufficiale e poi gestire il pacchetto compresso, o attraverso ambienti di sviluppo come Android Studio, che verrà approfondito successivamente.

In fase di progettazione può tornare utile, inoltre, avere un **AVD** (*Android Virtual Device*), più facilmente detto *emulatore*, che ha il compito di simulare un particolare dispositivo Android sul quale eseguire test e verifiche dell'applicazione stessa.
È largamente utilizzato in quanto permette di avere un'intera struttura di riferimento, comodamente accessibile dal proprio computer.
Imita la complessità del sistema sia hardware che software con, in aggiunta, la possibilità di modificare alcuni aspetti per meglio adattarli al proprio utilizzo e alle proprie esigenze.
Esso viene in ausilio ai programmatori, fornito all'interno del pacchetto software delle SDK, ma non è obbligatorio usufruirne.
Si dà un esempio di quella che può essere la schermata di un emulatore Android con la Figura 2.1 sottostante. Sulla destra si notano un insieme di pulsanti che danno la possibilità di utilizzare l'emulatore come un vero e proprio dispositivo: tasti come la Home, l'avvio della fotocamera, la gestione del volume e dell'avvio.

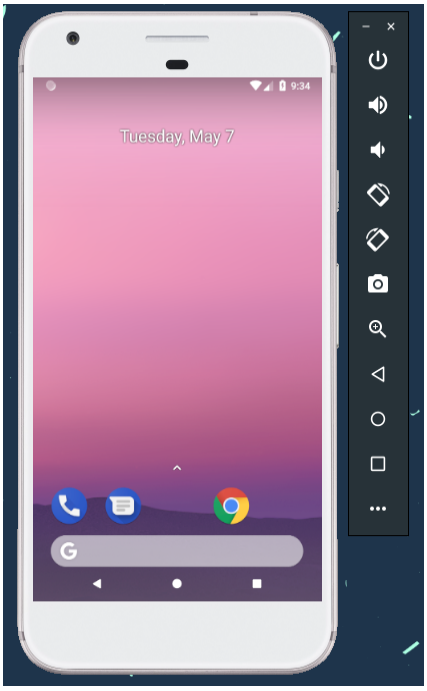

Figura 2.1 – Schermata emulatore Android

2.1.3 APK

Le potenzialità del sistema Android non si esauriscono con le SDK e gli emulatori disponibili, ma è possibile inoltre gestire file APK (*Android Package*), che sono una variante del formato JAR utilizzato per la distribuzione e installazione di componenti su dispositivi mobili. [14]
Utilizzando l'emulatore fornito, poi, è estremamente semplice montare gli archivi APK in quanto, dagli ultimi aggiornamenti, è sufficiente trascinare il file sulla schermata dell'emulatore stesso per installarlo.

Nel progetto descritto da questo elaborato si è deciso di sfruttare le APK disponibili di Twitter, per implementarle nell'emulatore, e dare la possibilità all'applicazione di collegarsi e pubblicare effettivamente i messaggi studiati.

2.2 Ambienti di sviluppo e librerie

Per ambiente di sviluppo o **IDE** (*Integrated Development Environment*) si intende avere un sistema integrato di progettazione in grado di assistere gli sviluppatori in fase di programmazione.
Spesso sono dotati di numerosi strumenti come un editor di testo e un compilatore, o interprete a seconda delle necessità dei linguaggi.
Un IDE è in grado di segnalare, inoltre, gli errori di sintassi nel codice e dà un sostegno in fase di *debugging*. Possono essere sia orientati ad un solo linguaggio sia essere in grado di gestire uno spettro multi-linguaggio. [15]

2.2.1 PyCharm

PyCharm è un ambiente di sviluppo (IDE) che permette di gestire ogni tipo di progetto sviluppato in Python, ovvero un linguaggio dinamico orientato agli oggetti.
Come molti degli ambienti ormai conosciuti, anche PyCharm è multi-piattaforma, quindi è possibile utilizzarlo con qualunque sistema operativo (Linux, Windows o iOS).
Il motivo della sua estrema diffusione sta nella capacità di gestire ed implementare Web Framework, come Django o Flask.
È su questa proprietà che è stato utilizzato per implementare il lato server del progetto presentato, quello responsabile della comunicazione con l'applicazione Android gestita dal client utente.

2.2.2 Flask Web Development

Flask viene considerato un *mini framework* web, in quanto presenta un nucleo semplice ma estendibile, scritto in Python e basato sullo strumento Werkzeug WSGI e con motore template Jinja2.
È, dunque, un applicativo software in grado di progettare siti web dinamici, applicazioni e servizi web. Fornisce numerose librerie che possono essere utilizzare per agevolare il programmatore nelle fasi di sviluppo di un'applicazione web, come accesso a basi di dati o alla creazione di template. [16] [17]
È stato necessario utilizzare questa tecnologia per essere in grado di creare le applicazioni lato server che, con algoritmi di machine learning, analizzassero i messaggi che l'utente voleva inviare.

Un altro web framework molto diffuso è Django, anch'esso basato su linguaggio Python e che presenta una struttura più ampia e completa rispetto a Flask.

2.2.3 Android Studio

Android Studio è l'unica piattaforma IDE ufficiale per la programmazione nell'ambito del sistema Android, basato su IntelliJ IDEA, scritta in Java.

Propone una struttura ampia per l'ausilio alla programmazione, con editor di testo e strumenti per la gestione dell'aspetto grafico dell'applicazione stessa.

È uno dei dispositivi più completi per quanto riguarda lo sviluppo in quanto gode di enorme flessibilità: permette di installare direttamente dall'interfaccia ogni tipo di SDK ed emulatore; ha un supporto interno per la Google Cloud Platform, in modo da rendere accessibile l'integrazione al Google Cloud Messaging direttamente dall'app e, infine, supporta C++ e NDK.

La grande particolarità di Android Studio sta nel suddividere il lavoro in *attività*, ovvero blocchi predefiniti di codice che rappresentano punti d'ingresso per l'applicazione.

Essa, infatti, può avviarsi da qualsiasi attività specificata, anche se solitamente questo avviene in caso di errori o sospensioni di quella principale.

Il loro ciclo di vita viene implementato come raccolta di chiamate di sistema, tale che, durante l'esecuzione, l'attività cambia il suo stato come mostrato nella Figura 2.2 sottostante. [18] [19]

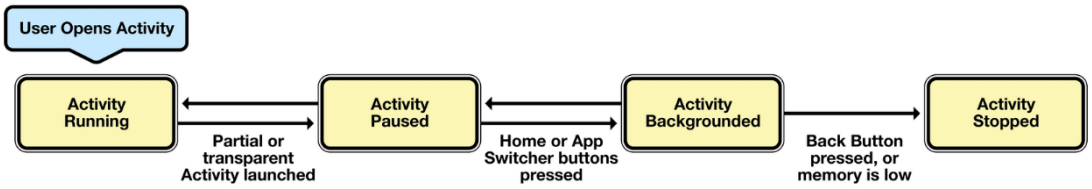

Figura 2.2 – Ciclo di vita di un'attività

2.2.4 Postman

Postman è un API, quindi un'interfaccia di programmazione, in grado di simulare l'utilizzo di applicazioni web tramite richieste HTTP.
È un software estremamente diffuso, perché viene in ausilio dello sviluppatore con metodi di testing e debugging intuitivi e semplici.
Presenta, naturalmente, una struttura multi-piattaforma adatta, quindi, per Linux, Windows o iOS.
In un contesto client-server, con Postman, è stato possibile testare ogni tipo di *request*, con aggiunta anche di oggetti JSON, e personalizzare le *responses*.

2.2.5 Volley

Volley è una libreria basata sul protocollo HTTP, che ha lo scopo di gestire le connessioni di rete per le applicazioni Android.
Offre numerosi benefici quali la pianificazione automatica delle richieste di rete, strumenti per l'ausilio al debugging, connessioni multiple concorrenti e supporta inoltre la necessità di avere richieste prioritarie e gestirle come tali.
È una libreria ad uso gratuito e open source, disponibile facilmente su GitHub.
È largamente utilizzata, data la sua semplicità d'implementazione in un qualsiasi progetto già presente su Android Studio: è necessario solamente inserire la dipendenza all'archivio ufficiale, nel proprio file *build.gradle*.
Inoltre si integra facilmente con ogni tipo di protocollo e permette di amministrare la gran parte di dati come stringhe, immagini e oggetti JSON. [20]
Per il progetto proposto è stata una libreria fondamentale per coordinare la comunicazione tra client, rappresentato dalla applicazione Android, e server.

2.2.6 Pandas

Pandas è una libreria open source scritta in Python che fornisce strutture dati semplici da utilizzare e altamente performanti, che possono essere utilizzate per *data analysis*.
È multi piattaforma e facilmente installabile tramite il comando di sistema *pip*.
Il suo successo è dovuto ai numerosi metodi disponibili per leggere ed ottenere dati, dalla maggioranza dei formati dei file.
Fornisce, inoltre, la possibilità di manovrare e modificare le strutture dati tale da unire, cancellare e selezionare anche solo porzioni di esse.
Il tipo di dato più utilizzato, introdotto proprio da questa libreria, è sicuramente il **DataFrame**, ovvero una struttura bidimensionale, mutabile in grandezza, che è in grado di contenere dati eterogenei. [21]

2.2.7 Scikit-learn

Essa è una delle librerie open source più utilizzate per l'elaborazione dei dati e l'esecuzione di *data analysis*.
Si appoggia ad altre strutture come *matplotlib* e viene sfruttata principalmente per implementare algoritmi di classificazione, regressione, raggruppamento e preprocessing di dati. [22]
Versatile nell'utilizzo: sia per problemi di machine learning supervisionati che non.
Nel progetto proposto è stata, dunque, utilizzata per sviluppare gli studi sui dati preposti, per essere in grado di realizzare un modello per il classificatore di tweet.

2.2.8 Matplotlib e Seaborn

Sono entrambe librerie grafiche per la visualizzazione di dati, scritte in Python.
Seaborn è stata progettata sopra la precedente Matplotlib, per essere più performante ed estendere le sue funzionalità.
Matplotlib gestisce prettamente grafici in due dimensioni, mentre Seaborn dà una completezza maggiore nella gestione dei dati con l'integrazione con Pandas.
È possibile quindi avere un data set orientato alle API per esaminare le relazioni tra variabili multiple; si ha uno speciale supporto alla categorizzazione delle variabili, una stima automatica per i modelli di regressione lineare e, infine, viste convenienti per strutture di dati complessi. [23]

2.3 Tipi di dato

2.3.1 JSON

Nel progetto sopra proposto sono stati utilizzati numerosi tipi di dato, tra i quali occorre specificare meglio il formato JSON (*JavaScript Object Notation*).
Esso è un formato per lo scambio di dati che coniuga la facilità di comprensione sia per gli umani, che per le macchine.
Sono strutture dati universali che risultano indipendenti da ogni tipo di linguaggio utilizzato a basso livello, formate da una collezione di coppie *chiave-valore*. [2]
Questo formato è stato utilizzato per codificare, in maniera generalmente conosciuta, i messaggi tra il client e il server.

Parte II

Progetto dashboard intelligente

CAPITOLO 3

Struttura applicazione

I capitoli seguenti hanno lo scopo di introdurre, più nel dettaglio, quello che è stato il lavoro attivo svolto per il progetto presentato.

Come verrà analizzato successivamente, l'attività è stata divisa in due macro-aree: la prima aveva come obiettivo quello di creare un applicativo server funzionante che potesse essere in grado di gestire un modello a classificatore, studiare il messaggio in ingresso e formulare eventuali suggerimenti per migliorarlo.

Il lato server è il fulcro del progetto, in quanto in esso sono coordinate le risorse e gli algoritmi di machine learning, con apprendimento supervisionato, visti nel capitolo 1.

Una volta implementato, è stato possibile dedicarsi alla realizzazione di un'applicazione Android semplice ed intuitiva, che potesse essere utilizzata dai dirigenti dei musei presentati, per mettersi in comunicazione con il server.

Questa applicazione deve differenziare l'accesso in base al tipo di museo di cui il manager fa parte e mostrare tutti i risultati dello studio con il classificatore, suggerimenti per migliorare il messaggio inclusi.

Per dare un senso di completezza allo studio stesso, si ha scelto di implementare inoltre un collegamento a Twitter, tramite username e password di account reali, che dà la possibilità di pubblicare effettivamente il messaggio studiato, insieme ad immagini, hashtag e menzioni.

3.1 Elaborazione dei dati per il classificatore

In primo luogo è stato necessario raccogliere i diversi tweet, resi disponibili dai musei studiati, tali che formassero dunque sei distinti data set: la Tabella 3.1 sottostante mostra come sono stati formati i gruppi in base alla popolarità di ogni museo.

Ogni parte del progetto in esame è stata implementata perché prendesse in considerazione sempre le differenze tra i vari insiemi di dati.

Sarà l'utente, tramite l'applicazione Android e la scelta del museo da studiare, a decidere a basso livello da quale dei sei data set prendere i dati di riferimento.

Utilizzando la libreria *Pandas*, descritta nel capitolo 2, è stato possibile strutturare velocemente i dati e salvarli in DataFrame per poter essere in grado di manipolarli.

	Gruppo 1	**Gruppo 2**	**Gruppo 3**	**Gruppo 4**	**Gruppo 5**	**Gruppo 6**
	Museum of Modern Art	British Museum	Centre Pompidou	MFA Boston	Visit MUVE	Museo Egizio
	Tate	Van Gogh Museum	National Gallery	Musei in comune	Museu Picasso (Barcelona)	Uffizi
	MET Museum	Musee Louvre	Museo Frida Kahlo	Phila Museum	Mart Museum	MUSE Trento
	Guggenheim	Getty Museum	Musee Orsay	Masp museum		
	Saatchi Gallery	Museo del Prado		National Gallery of Art		
				Museo MAXXI		
Totale Tweet	**3761**	**3838**	**2474**	**4931**	**2164**	**2420**

Tabella 3.1 - Divisione dati

	FORMULA	NRETWEET	NLIKE	NANS	MUSEUM	NURLS	NIMG	NHASH	NMENTION	LENGTH	DENSE	SENT	ENG	LONG	FULL_TEXT
2	0	294	458	5	britishmuseum	1	2	0	0	227	0	1	0	1	June is named after the Roman goddess Juno - the god of marriage
3	0	323	691	6	britishmuseum	0	2	1	0	305	0	1	0	1	The Houses of Parliament clock, often popularly referred to as Bi
4	0	615	1774	26	britishmuseum	0	4	0	0	196	0	1	0	1	15,000 tiny turquoise tiles decorate this Aztec serpent, made
5	0	273	543	13	britishmuseum	1	2	0	0	236	0	1	0	1	Did you know workers who built the pyramids were paid in beer? W
6	0	131	397	10	britishmuseum	1	1	1	0	274	0	0	0	1	'It's the artist who tells the truth and the photographer who lies
7	0	65	215	2	britishmuseum	1	2	1	0	177	0	1	0	1	Get close to the artist's masterpieces in our #RodinExhibition an
8	0	264	700	13	britishmuseum	0	2	1	1	184	0	1	0	1	The Tube hasn't changed much in 84 years! This print of a @distri

Figura 2.1 - Visione tabella di un gruppo

Ogni gruppo di dati viene elaborato come un file tabellare in cui ogni riga rappresenta un thread, ovvero un tweet con tutte le sue componenti divise per colonne, come schematizzato dalla Figura 3.1.

Le caratteristiche evidenziate per ogni messaggio sono:

- NRETWEET: indica il numero di *retweet* che il messaggio stesso ha conseguito.
- NLIKE: indica la quantità di *like* ricevuti.
- NANS: specifica la somma di risposte sotto il tweet.
- MUSEUM: consente di indicare quale è stato il museo mittente del messaggio.
- NURLS: numero di link url contenuti nel messaggio.
- NIMG: quantità di immagini nel tweet.

  Si ricorda che Twitter impone un limite di massimo 4 immagini per messaggio.
- NHASH: somma degli hashtag presenti.
- NMENTION: numero di menzioni contenute.
- LENGTH: indica la lunghezza del messaggio.
- DENSE: valore booleano per indicare un tweet "denso", quindi con molte componenti speciali come menzioni e hashtag.

  Più nello specifico, viene definito denso se contiene più di 4 hashtag, oppure più di un indirizzo url, o più di 3 menzioni.
- SENT: anch'esso è un valore booleano che viene usato per indicare il sentimento generale del messaggio.
- LONG: valore booleano per indicare un tweet lungo, ovvero composto da più di 100 caratteri.
- ENG: indicante la lingua inglese principale.
- FULL_TEXT: ingloba l'intero corpo del messaggio.

È visibile come i valori siano eterogenei tra loro, a livello di intervalli che comprendono, quindi con data set particolarmente estesi si tende a **normalizzare** i dati, a scalare quelle che sono le quantità numeriche, in modo da limitare la differenza dei valori per farle sottostare entro determinati limiti.
Una pratica comune per scalare i dati è quella di rappresentarli in un intervallo di valori [0,1]: la distribuzione e l'importanza di ognuno restano invariati, ma vengono adattati per restare nei valori tra zero e uno.
Un ulteriore tipologia di normalizzazione è quella che racchiude i valori numerici entro il minimo e massimo dell'insieme di cui fanno parte; l'obiettivo resta il medesimo, ovvero quello di avere un intervallo meglio distribuito da cui iniziare le proprie analisi.

Una volta processati i dati in modo da avere formazioni tabellari meglio organizzate, si procede alla costruzione del **classificatore**, nel quale occorre, genericamente, avere un metodo di discrezione in grado di etichettare i dati in base a determinate proprietà *target*, perché si vuole implementare un algoritmo di machine learning supervisionato, ed avere uno studio preliminare per selezionare le caratteristiche (*features*) da monitorare per creare il modello.
Per quanto riguarda la fase di assegnazione di un'**etichetta** ad ogni tweet, bisogna avere una metodologia chiara che sia in grado di distinguere un messaggio con esito positivo rispetto ad uno con esito negativo.
Le caratteristiche principali per questa scelta sono il numero di *like*, di *retweet* o di risposte, in quanto tutte e tre buoni indicatori dell'impatto sul pubblico; è stato deciso il numero di like come variabile target, perché universalmente riconosciuta come quella più espressiva dell'assenso dei lettori.
Concentrandosi su questo valore numerico, è stato implementato un meccanismo di gestione a *code* per decretare la soglia che avrebbe rappresentato il "successo".
Viene considerato il 20% dei messaggi che hanno ricevuto più like come l'insieme di riferimento per l'etichetta 'GOOD' e, analogamente, il 20% con meno like ricevuti compongono l'insieme dei messaggi 'BAD'; questo per diminuire la possibilità di dover analizzare un messaggio ambiguo difficilmente classificabile anche da un essere umano.

Successivamente è stato necessario analizzare ogni tweet in modo più generale: attraverso le **componenti** che lo costituiscono, quindi strutturare un'analisi per colonne dei file come quello mostrato in Figura 3.1.

Mediante la funzione di correlazione della libreria Pandas e con l'ausilio della libreria Seaborn per la creazione del grafico, si costruisce una *matrice di correlazione* in grado di mostrare le eventuali relazioni tra le caratteristiche generali e quella target definita in precedenza (numero di like).

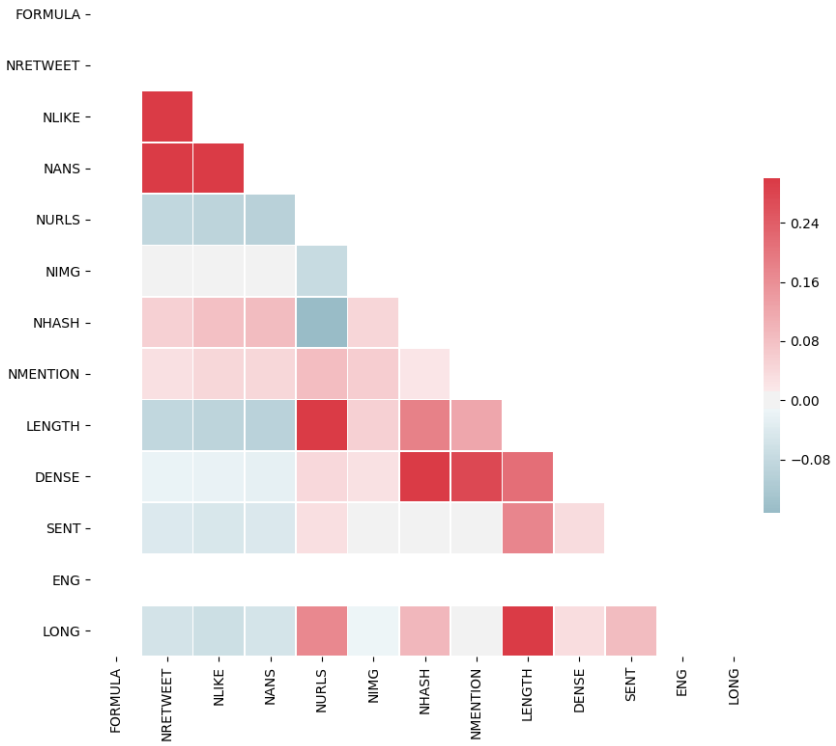

Figura 3.2 - Matrice di Correlazione sul gruppo 1

La Figura 3.2 mostra le relazioni che sono emerse nel primo gruppo di studio di musei, tra le componenti dei tweet.

Lo scopo di questo studio preliminare è quello di eliminare caratteristiche contrastanti (o molto simili) in modo da rimanere con solo quelle strettamente necessarie per il target.

Dalla Figura 3.2 emergono chiaramente forti correlazioni, espresse dal colore rosso più intenso, e autonomie che vengono identificate dal colore blu.

Inizialmente devono essere eliminate le proprietà che un nuovo messaggio non può possedere: come il numero di like, di retweet e di risposte.
Comprensibilmente verranno ignorate quelle che non hanno alcun impatto sulla variabile target NLIKE: la formula e l'indicazione della lingua inglese.

Effettuata questa prima selezione, occorre avere una prima bozza elementare del modello implementato mediante l'algoritmo KNN, con l'ausilio della libreria Scikit-learn, per analizzare quali caratteristiche occorre eliminare e su quali invece occorre concentrarsi per migliorare le prestazioni di accuratezza.
Dalla Figura 3.2 sono facilmente visibili particolari relazioni tra le caratteristiche: sia quelle positive, ovvero quando due proprietà concorrono per esprimere lo stesso sentimento, rappresentate dal colore rosso acceso, sia quelle negative, viste in blu, che sottolineano due attributi antagonisti.
La prima forte correlazione è visibile tra gli attributi LENGTH e LONG, in cui viene eliminato quest'ultimo in quanto non genera alcuna differenza sull'accuratezza del modello.
La lunghezza, però, è in correlazione inoltre con NURLS, ovvero con il numero di collegamenti che il tweet possiede. Tra queste due caratteristiche quella di maggiore impatto sicuramente è quest'ultima, che viene dunque mantenuta.
Viene eliminato anche l'attributo SENT, in quanto non incide sulla variabile target.
Gli ultimi due legami da analizzare sono tra le caratteristiche DENSE/NHASH e NURLS/NHASH: rispettivamente vengono ignorate NURLS, che peggiorava le prestazioni, e DENSE, che non aveva correlazione sul numero di like.
Questo passaggio di selezione delle caratteristiche sarà ripreso in maniera più dettagliata e con supporto di tabelle nel capitolo 5, dove si discuteranno i risultati ottenuti dalle prove sperimentali.
Da questo primo studio si conclude che le proprietà su cui basare il classificatore sono NIMG, NHASH e NMENTION.

In conclusione si afferma che, partendo dai tweet privi di forma, è stato possibile dare loro una struttura più consona per poterli analizzare; ognuno di essi è stato etichettato in base ad una variabile target decisa e sono state selezionate le sole proprietà su cui basare il classificatore, in modo da massimizzare le sue performance.

3.2 Studio delle distanze e dei suggerimenti

Il concetto di **distanza** è fondamentale per l'intero algoritmo da implementare: il KNN è, infatti, interamente basato sulla rappresentazione degli elementi come vettori in uno spazio multi dimensione e sul calcolo delle distanze per definire le classi di appartenenza.

Tramite l'ausilio della libreria Scikit-learn, meglio evidenziata nel capitolo 2, è possibile implementare questo tipo di algoritmo personalizzando anche lo studio delle distanze.

La misura più adottata è quella di Minkowski, secondo la quale la distanza tra due punti è la somma del valore assoluto delle differenze delle loro coordinate, come mostrato nella Figura 3.3 sottostante.

$$D(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{l=1}^{d} |x_{il} - x_{jl}|^{1/p} \right)^p$$

Figura 3.3 – Distanza di Minkowski

Viene definita come formula di base per il calcolo delle distanze in quanto, andando a cambiare il parametro *p,* si definiscono altre funzioni, ad esempio con *p = 1* si intende la distanza di Manhattan, mentre con *p = 2* viene definita la distanza di Euclide.

Una particolarità che è stata introdotta per la realizzazione di questo progetto, è stata quella di sfruttare queste misurazioni non solo per la classificazione iniziale degli elementi, ma anche per lo studio degli oggetti vicini e della formulazione di eventuali **suggerimenti**.

È un approccio diverso rispetto al tradizionale uso di questo algoritmo: estrarre dal messaggio le caratteristiche ricercate dal classificatore e restituirne l'esito positivo o negativo che sia.

In questo caso si studia l'*intorno* del punto in cui si trova il tweet, inteso come spazio composto da messaggi a lui simili.

In questo tipo di studi, dove gli elementi sono rappresentati come vettori in uno spazio, il concetto di *similarità* viene inglobato in quello di distanza: se due oggetti sono strutturati vicini significa che hanno caratteristiche simili.

Applicando gli algoritmi di distanza si è in grado di restituire K vicini dai quali studiare le caratteristiche e, dei positivi entro un limite fissato di lontananza, apprendere il motivo per cui hanno avuto esito diverso da quello del messaggio originale.

Come per la scelta del valore K nel modello, anche per il limite di distanza non esiste un valore predefinito tale che possa essere applicato in ogni contesto: occorre eseguire test e verifiche specifiche per il proprio data set e valutare quale valore ha l'impatto migliore. Una quantità ristretta potrebbe portare a non avere vicini positivi dai quali apprendere, in quanto si rischia di restringere troppo l'intorno, mentre un valore alto porterebbe a prendere in considerazione troppi elementi e, dunque, introdurre rumore nello studio, ponendo l'attenzione su oggetti molto diversi da quello iniziale.

Figura 3.4 – Schema algoritmo suggerimenti

La Figura 3.4 mostra le fasi principali per la realizzazione del procedimento che porta a restituire possibili suggerimenti.

In primo luogo è necessario categorizzare il nuovo messaggio, in modo da ottenere la sua etichetta, positiva o negativa che sia.

Si ricorda che questo procedimento si deve basare sulle tre caratteristiche che in precedenza hanno avuto un impatto maggiore: numero di immagini, di hashtag e di menzioni.

Del nuovo messaggio, dunque, verranno isolate solo queste proprietà, perché sono quelle ricercate dal modello.
Una volta categorizzato, sarà impiegata nuovamente la libreria Scikit-learn, che permette di restituire i K vicini con le relative distanze dal messaggio iniziale.
Sono necessarie operazioni di **filtraggio** dell'intorno: si ignorano tutti i risultati con un esito negativo, in quanto non ha rilevanza apprendere la loro struttura, e si calibra una soglia massima di lontananza entro la quale è conveniente studiare gli elementi simili.
Questo perché, è vero che vengono analizzati i più vicini, ma questa distanza può essere comunque elevata, rappresentando dunque messaggi con caratteristiche diverse da quello iniziale, che porterebbero al suo completo cambiamento.
In una situazione come quella appena descritta, fornire suggerimenti basati su messaggi che non hanno caratteristiche simili produrrebbe un cambiamento trascurabile, se non anche peggiorativo, quindi vengono ignorati.

Al termine delle operazioni di filtraggio si studiano i tweet rimasti, consci del fatto che potrebbero essere stati eliminati tutti i vicini e, in questo caso, si ammette di non essere stati in grado di elaborare dei suggerimenti.
Se vi sono, però, messaggi positivi entro la soglia di distanza, è necessario elaborare una metodologia per studiare le loro caratteristiche.
È stata scelta la **media** matematica intera dei valori numerici che rappresentano le proprietà, come metodologia di studio, perché ha la capacità di descrivere un andamento generale e di smorzare eventuali differenze pronunciate nelle quantità.
Confrontando il numero di ogni caratteristica del messaggio originale, con la media calcolata, è stato possibile intuire l'andamento dei suggerimenti.
Dando un esempio concreto: nel caso in cui la media, di una caratteristica, fosse più alta della quantità della stessa nel messaggio da analizzare, allora potrebbe essere consigliato aumentare quella caratteristica, perché la porterebbe più vicino alla media positiva.
Vale anche il ragionamento contrario, ovvero quando la media risulta minore: in questo caso è consigliato diminuire la quantità di quel attributo.

Segue un caso pratico riportato dalla Figura 3.5 sottostante, in cui si vuole spiegare ancora più nel dettaglio il ragionamento messo in atto per i suggerimenti.

Caratteristiche nuovo messaggio			Media delle caratteristiche dei suoi vicini		
NIMG	**NHASH**	**NMENTION**	**NIMG**	**NHASH**	**NMENTION**
3	1	0	**2**	3	0

Figura 3.5 – Esempio calcolo suggerimenti

Nell'esempio soprariportato viene mostrata una situazione in cui è possibile apprendere dai vicini, in quanto non vengono eliminati dai metodi di filtraggio.
Il numero medio di immagini è inferiore rispetto alla quantità che possiede il nuovo messaggio, quindi si potrebbe consigliare all'utente di eliminare un'immagine, per assomigliare ad un suo vicino con esito positivo.
Viene eseguita la stessa riflessione su tutte le altre caratteristiche prese in considerazione: si nota come il numero di hashtag medio sia maggiore rispetto a quello del messaggio, quindi potrebbe essere aumentato.
Sul numero di menzioni non vengono espressi miglioramenti: è esattamente pari alla media dei suoi vicini.

Un'ultima scelta implementativa, per quanto riguarda le proposte sui tweet, è stata quella di fornire all'utente un consiglio per ogni caratteristica, in modo da dare la libertà ad esso di decidere su quale è meglio concertarsi, rispetto anche alle sue scelte di pubblicità.
Nell'applicazione dunque sarà possibile navigare tra diversi consigli, così che l'utente possa partecipare in maniera diretta alla composizione del suo messaggio.

3.3 Generatore di tweet

Una volta elaborato l'algoritmo di creazione dei suggerimenti, sul progetto proposto, è stato necessario poter esaminare quanto fosse efficiente su esempi concreti.

È stato ideato un **generatore** randomico di tweet, in modo da verificare se effettivamente queste proposte avrebbero portato ad un miglioramento del messaggio.

Il procedimento si basa sulla creazione di un numero prefissato di tweet, con valori delle caratteristiche però randomici, e viene eseguito su di loro l'algoritmo decisionale per la creazione dei suggerimenti.

Viene simulato il comportamento di un utente che interagisce con il sistema: una volta composto il messaggio, verrà scelto in modo randomico quale suggerimento seguire e verrà rianalizzato il tweet.

Questa procedura viene eseguita per un massimo di 5 iterazioni, al termine delle quali viene studiato l'esito del messaggio e la rispettiva etichetta.

Lo scopo principale per cui è stato ideato è quello di verificare se un tweet, che ha avuto un esito negativo, seguendo vari suggerimenti proposti, può arrivare ad ottenere un risultato positivo e, dunque, aumentare il suo impatto sul pubblico.

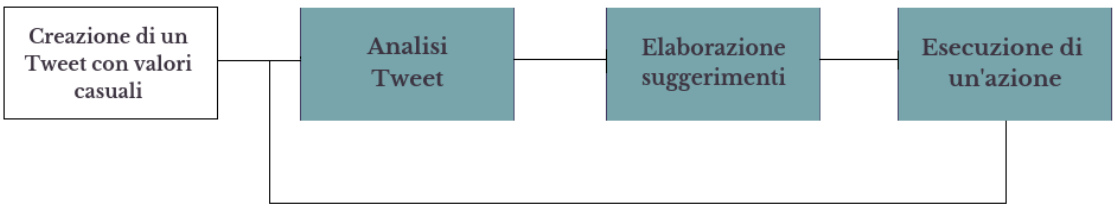

Figura 3.6 – Algoritmo per il generatore di Tweet

Dalla Figura 3.6 è possibile intuire il funzionamento generale che sta alla base del generatore di tweet.

In primo luogo si individua la fase iniziale, che riguarda la creazione vera e propria di un singolo messaggio con caratteristiche rappresentate da valori randomici, tali da riprodurre comunque uno scenario realistico.

Successivamente si instaura un ciclo di analisi del messaggio appena creato, composto anche dalla elaborazione dei suggerimenti stessi.
Il tweet verrà studiato dal classificatore, il quale gli assegnerà una etichetta rappresentante il suo esito e, in caso di risultato negativo, provvederà a inizializzare il procedimento di elaborazione dei suggerimenti.
Di queste proposte ricevute è necessario sceglierne una in maniera casuale, come accadrebbe in uno scenario con un utente vero e proprio, ed eseguirla andando a modificare quindi il messaggio di partenza.
A questo punto è necessaria una seconda analisi del tweet, per verificare un eventuale miglioramento dovuto all'esecuzione del suggerimento scelto.

Il punto di forza del meccanismo ideato sta nella sua componente di casualità: non solo in fase di creazione del messaggio, ma anche nella scelta successiva del suggerimento da applicare.
Con questo tipo di applicazione è stato possibile simulare il comportamento di un utente e, quindi, rendere realistico l'intero procedimento.

3.4 Funzionamento generale

Il progetto proposto ha uno sviluppo bilaterale su due macro-aree: la parte server che elabora i dati e restituisce i risultati, vista nei capitoli precedenti, e la parte client che viene manovrata dall'utente finale.

Hanno due implementazioni distinte, in quanto l'utente è all'oscuro della complessità dietro quella è che l'applicazione che deve utilizzare.

Qui di seguito viene riportato uno schema (Figura 3.7) che rappresenta, macroscopicamente, quello che è il funzionamento di base dell'applicazione.

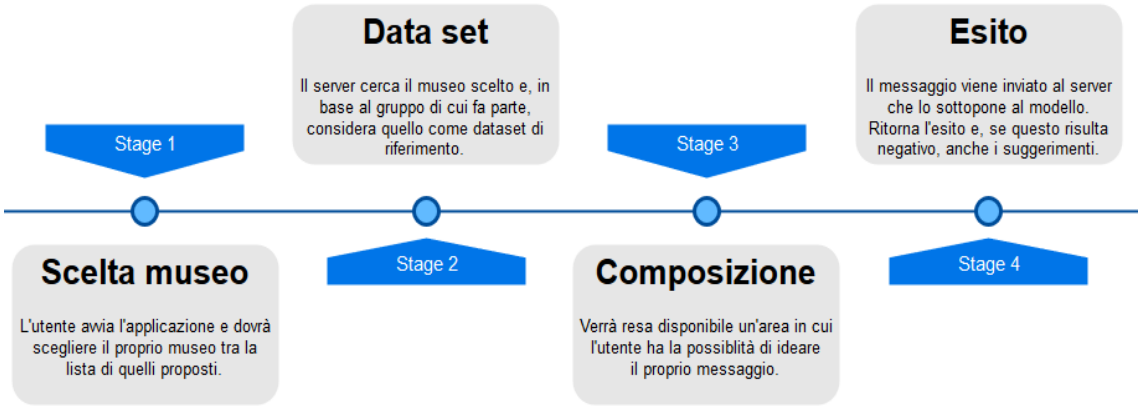

Figura 3.7 – Funzionamento applicazione

Come mostrato nella Figura 3.7 è chiaramente distinguibile un procedimento a più fasi, che hanno portano all'utilizzo dell'applicazione client Android.

In primo luogo all'utente viene chiesto di scegliere il proprio museo da una lista.

A basso livello i **nomi** dei musei sono contenuti all'interno di un file di testo, appositamente scelto perché possa essere modificato in ogni momento, in caso di inserimento di un nuovo museo, o di eliminazione di uno già esistente.

Questo file viene poi mandato dal server all'applicazione, mediante la connessione HTTP che li mette in comunicazione, durante la fase di avvio della stessa.

La scelta del museo ha conseguenze importanti sul progetto: il server, infatti, in base al gruppo di cui fa parte il museo scelto, prenderà quello come **data set** di riferimento per creare il proprio modello a classificatore.

È bene ricordare che questa distinzione in gruppi è stata necessaria al fine di riuscire a confrontare risultati di musei più o meno simili, dal punto di vista del grado di popolarità.
L'utente, a questo punto, si troverà davanti una pagina dedicata alla **composizione** del proprio messaggio: è disponibile un'ampia area testo in cui inserire la propria idea, con tanto di hashtag e menzioni.
Si concede la massima libertà per la scrittura del tweet stesso, sempre ricordando il massimo numero di caratteri che Twitter impone (nel caso del progetto, 140).
Una volta inserito il proprio messaggio, questo verrà inviato al server che procederà al confronto con il modello creato in precedenza.
Vengono estratte le caratteristiche sulle quali è stato deciso di basare il classificatore (numero immagini, hashtag e menzioni) e, dal confronto, è possibile inviare al client l'**esito** dell'analisi, ovvero l'etichetta assegnata al nuovo messaggio.
Se il risultato evidenzia un comportamento negativo, allora il server avvierà il processo di studio e creazione dei suggerimenti, tali da poter essere visibili dall'applicazione e, quindi, dall'utente.
A quest'ultimo verranno date più possibilità per modificare il proprio messaggio e rinviarlo dunque per una successiva analisi, o fino a che non si ritiene soddisfatto del risultato ottenuto.
Non sono presenti limitazioni circa il numero di volte con cui contattare il server per ottenere il controllo sul messaggio.

Una volta che l'utente decide di aver completato il processo di analisi è possibile accedere a Twitter, mediante l'apposito pulsante di login posto nella schermata principale, e pubblicare sul proprio profilo il messaggio da lui ideato.

CAPITOLO 4

Implementazione

Questo capitolo si pone l'obiettivo di illustrare ogni passaggio che ha portato alla realizzazione del progetto proposto.
Dopo aver spiegato meglio le tecnologie adottate, nel corso del secondo capitolo, verrà chiarito il loro ruolo andando a riportare esempi concreti di codice utilizzato.
Il capitolo si apre spiegando la struttura dell'applicativo server e delle sue funzionalità, della creazione dell'applicazione Android e dei collegamenti necessari che questi due componenti hanno dovuto coordinare per sfruttare il classificatore e, inoltre, esamina la costruzione di suggerimenti basati sulle caratteristiche dei messaggi vicini a quello dell'utente.

4.1 Costruzione Server

Esso è la componente con il quale il client, inteso come applicazione Android, deve mettersi in comunicazione in modo da rispondere alle sue interrogazioni sul messaggio proposto.
È stato necessario costruire un servizio web, quindi un sistema software in grado di gestire un'interazione macchina-a-macchina tramite una connessione in rete.
Come già mostrato nel secondo capitolo, viene scelto Flask come framework per questo tipo di compito.

Essendo scritto in Python è facilmente installabile con il comando di sistema 'pip', direttamente dal terminale:

```
pip install Flask
```

Come ogni framework web, anche Flask possiede una logica di indirizzamento delle views, ovvero funzioni Python, in base agli URL specificati: questo implica che ad ogni indirizzo che si sceglie di implementare, deve corrispondere una funzione che esegue compiti sui dati.

Essendo Flask un REST API, si appoggia su quella che è la comunicazione mediante il protocollo HTTP, di cui è possibile utilizzarne i metodi per manipolare i dati all'occorrenza.

I più utilizzati sono i metodi GET e POST: il primo permette di accedere ad una risorsa del server, mentre il secondo dà la possibilità di inviare dati a quest'ultimo, in vari formati, come ad esempio il JSON.

Viene riportato di seguito un esempio pratico che illustra come attivare la prima applicazione Flask.

Facendo eseguire questo breve script verrà restituita all'indirizzo di default 127.0.0.1:5000/, che rappresenta l'indirizzo locale e il numero della porta alla quale il servizio è attivo, la scritta 'Hello World", come illustrato dalla Figura 4.1 sottostante.

La funzione definita come index accetta solo metodi GET, quindi è possibile richiedere la risorsa (la pagina), ma non è permesso mandarle dati tramite URL in quanto non implementa il metodo POST.

```
from flask import Flask
app = Flask(__name__)

@app.route ('/',
        methods=['GET'])
def index ():
    return "Hello, World"

if __name__ == '__main__':
    app.run(debug=True)
```

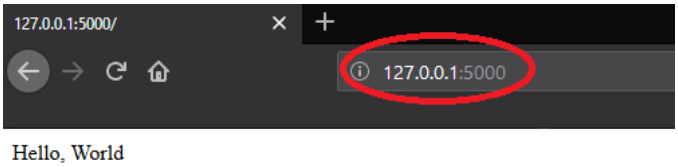

Figura 4.1 - Esempio pagina web

```
@app.route('/lista', methods=['POST'])
def lista():
    raw_data = request.get_json()
    return "Dati ricevuti"
```

Il codice sopra riportato mostra, invece, una view Python definita per un metodo POST. Nel caso semplice dell'esempio, il client ha inviato alcuni dati in formato JSON, che il server riceve e memorizza in una variabile `raw_data`, per utilizzarli in un secondo momento, e notifica il client con un messaggio di avvenuta ricezione.
L'elemento `request`, dunque, rappresenta la richiesta del client.

Dato che il server Python deve essere visibile esternamente, dall'applicazione Android o comunque da altri soggetti della propria rete, è necessario dare come indirizzo principale 0.0.0.0 in quanto, come forma di sicurezza, non viene espresso di default.
Come molti altri framework web, anche Flask implementa la metodologia di esecuzione con il degubber attivo, in modo da aiutare gli sviluppatori nella fase di creazione.
Esso infatti deve essere disattivato in fase di *release*, in produzione.
Il *routing*, ovvero l'indirizzamento delle pagine web e delle funzioni Python, è completamente arbitrario e deciso dal programmatore: l'unico vincolo resta quello di chiarezza tra il nome della funzione e il suo scopo, tale che sia poi la medesima chiamata dall'applicazione per ottenere il servizio richiesto.

4.2 Costruzione classificatore

Una volta eseguiti gli studi preliminari sui dati, tali da ottenere sei data set distinti in base ai gruppi di musei come spiegato nel capitolo precedente, è necessario implementare il vero e proprio modello a classificatore, con algoritmo K Nearest Nearbors, mediante l'ausilio della libreria Scikit-learn.

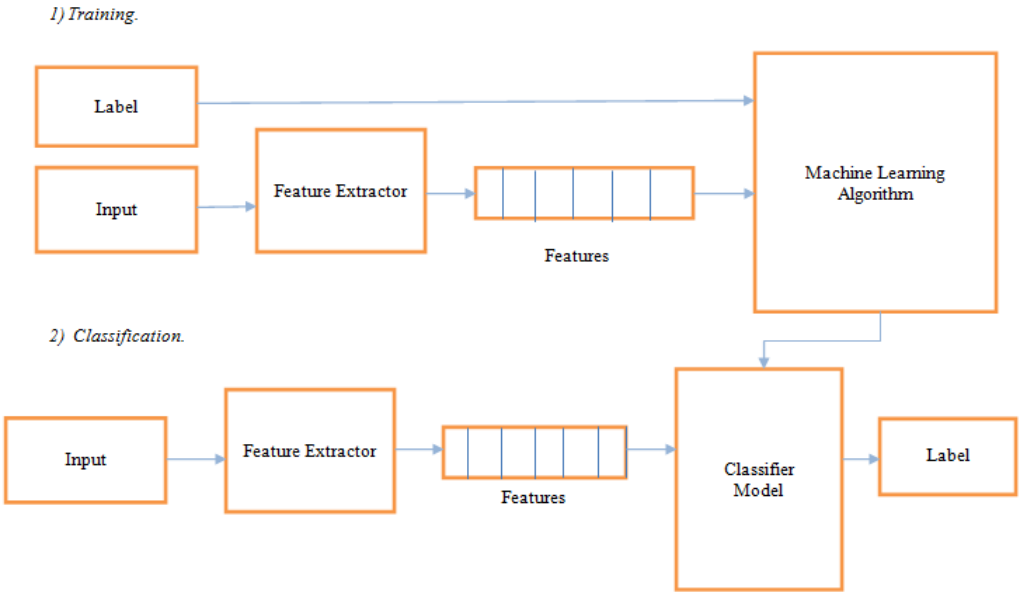

Figura 4.2 – Struttura modello a classificatore

L'intero procedimento di implementazione del classificatore è riassunto dalla Figura 4.2, che chiaramente lo divide in due macro aree: la fase di ***training*** del modello e quella di classificazione, meglio detta anche di ***testing***.

Nel primo metodo si assegnano le etichette ad ogni dato presente e si estraggono le caratteristiche che meglio lo definiscono, mediante analisi o test come quelli effettuati con la matrice di correlazione nel capitolo precedente.

Al termine della fase di training si ottiene un modello, che verrà successivamente utilizzato per assegnare ai nuovi dati la loro classe di appartenenza.

Prima di arrivare alla creazione del modello vero e proprio, è necessario imporre una metodologia chiara per la **divisione dei dati** nelle due collezioni di riferimento: training e testing set.

Si opta per il metodo *split*, che richiede la percentuale di grandezza di entrambi gli insiemi. Tendenzialmente si utilizza la maggior parte dei dati come training, in modo da creare un modello il più accurato possibile, avendogli fornito una grande quantità di esempi dal quale imparare, e la restante porzione viene intesa come campioni nuovi da categorizzare (testing).

```
for indx in range(0, NUM_RUN):

        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                            test_size=0.20,
                                                            random_state=indx,
                                                            shuffle=True)
        # Scala i dati
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
```

L'intera costruzione del classificatore è inserita in un ciclo da `0` a `NUM_RUN`, in modo da poter eseguire il codice più volte e ottenere dunque un valore medio, circa l'accuratezza dello stesso.

Il codice qui riportato illustra la divisione sopra specificata: mediate la funzione `train_test_split` è possibile creare i due insiemi di partenza di cui il modello necessita. Devono essere passati come argomenti: `X`, che rappresenta il DataFrame contente i dati, e `y`, che è il vettore di etichette, precedentemente calcolate e rispettive per i dati, in base alla variabile target NLIKE e il sistema di gestione a code, definito in precedenza.

Come altri parametri, per questo metodo, si sottolinea `test_size` che specifica la grandezza percentuale assegnata all'insieme testing, lasciando chiaramente la restante parte per la porzione di training.

Seppur non esiste un valore per la percentuale, tale che sia ottimale per ogni insieme, approssimativamente esso si aggira intorno al 20% per il testing (`X_test`) e il restante 80% per il training (`X_train`); dunque vengono inserite queste quantità anche per il progetto proposto.

`shuffle` è un valore booleano che indica la possibilità o meno di mescolare i dati prima di essere divisi e, infine, `random_side` rappresenta il "seme", l'inizio, numerico su cui il generatore randomico si basa.

I dati, una volta divisi nei due rispettivi macro insiemi, hanno la necessità di essere **scalati** per essere adattati nel modello a classificatore: è un'operazione molto comune per i gli algoritmi di machine learning, perché permette di distribuire i dati in maniera più omogenea.
Viene scelto il `RobustScaler`, reso disponibile dalla libreria Scikit-learn, che permette di assestare i valori numerici entro gli intervalli dei quartili (tipicamente entro il primo e il terzo).

Successivamente vi è la fase di costruzione del **modello** vero e proprio: viene implementato il `KNeighborsClassifier` della libreria Scikit-learn, definito come segue:

```
KNeighborsClassifier (n_neighbors, weights, algorithm, leaf_size,
p, metric, metric_params, n_jobs)
```

In cui:

- *`n_neighbors:`* è il valore di K, che deve essere un intero positivo e, tendenzialmente, dispari. È consigliato eseguire test preliminari per evidenziare il numero migliore, ma la libreria fornisce comunque un valore di default pari a 5.
- *`weights:`* attributo che definisce il peso di ogni punto all'interno del modello.
  Può assumere il valore '`uniform`', dunque dare un peso uniforme ad ogni elemento, o il valore '`distance`' che assume l'importanza del punto in base alla distanza: elementi più vicini saranno più rilevanti.
  Il valore di default è '`uniform`'.

- *algorithm*: definisce quale è stato l'**algoritmo** usato per individuare gli elementi vicini. Il più semplice e intuitivo è il brute-force (valore '`brute`'), in quanto prevede il calcolo delle distanze tra tutte le coppie di punti del data set.
  È l'approccio meno performante che può essere utilizzato solo con porzioni di dati modeste.
  Il K-D tree ('`kd_tree`'), invece, è una metodologia che si basa sull'utilizzo di strutture dati per rendere più performante l'algoritmo stesso. Facendo considerazioni aggregate, mira a diminuire il numero dei calcoli necessari: un esempio potrebbe essere quello di avere un punto generico A lontano da un punto B, il quale però risulta vicino ad un terzo punto C, allora si conclude che A è lontano anche da C.
  È molto impiegato per data set medio grandi.
  Il Ball tree ('`ball_tree`'), infine, è stato introdotto per migliorare il precedente algoritmo per quanto riguarda data set di grandi dimensioni; si differenzia dal KD nelle metodologie di aggregazione dei dati.
  Scikit-learn dà la possibilità di inserire il valore '`auto`', in modo da adeguare l'algoritmo in base al data set che si deve gestire.
- *leaf_size:* valore passato per gli algoritmi basati sugli alberi binari, che va ad influire sulla velocità di esecuzione e sulla costruzione dell'interrogazione.
  Il valore ottimale deve essere studiato in base al tipo di data set con cui si lavora.
- *p:* è un valore intero che identifica il tipo di formula per il calcolo delle distanze. Con p = 1 è equivalente ad utilizzare la distanza di Manhattan, mentre con p = 2 si impiega quella di Euclide.
- *metric:* la distanza utilizzata per le strutture dati ad albero.
  Il valore di default è Minkowski, ma dipende dal valore che viene dato a p.
- *metric_params:* sono parametri addizionali che possono essere inseriti per specificare la funzione di distanza.
- *n_jobs:* attributo che specifica il numero di lavori paralleli che devono essere eseguiti durante la ricerca dei vicini. [22]

```
accuracy = []
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    accuracy.append(metrics.accuracy_score(y_test, pred_i))

max_acc = max(accuracy)
for i in range(0, len(accuracy)):
    if (accuracy[i] == max_acc):
        optimal_k = i
        break
```

Si vuole mettere in luce, con il codice sopra riportato, la breve applicazione dietro quella che è stata la scelta per il valore ottimale di K, nella fase di apprendimento da parte del modello.

Esso viene definito all'interno di un ciclo, con valori di K varianti in base all'indice assunto, e, per ogni iterazione, si memorizza l'accuratezza del modello stesso.

Il K definitivo sarà quello che ha portato al grado più alto di quest'ultima.

```
knnModel = KNeighborsClassifier(n_neighbors=optimal_k,
                                weights='uniform')
knnModel.fit(X_train, y_train)
y_pred = knnModel.predict(X_test)
```

Dopo aver eseguito tutti gli studi sui dati e la loro distribuzione, aver costruito il modello e analizzato la quantità ottimale per K, è necessario andare ad inserire i dati elaborati nel modello stesso, mediante il metodo `fit`, come mostrato dal codice presente.

Questa funzione permette di addestrare il classificatore con la porzione di dati conosciuta come training set, tale che successivamente possa essere utilizzato sopra l'insieme testing, assegnando etichette su campioni che il modello non ha ancora incontrato, mediante il metodo `predict`.

Si trova, poi, l'**accuratezza** del modello mettendo a confronto l'etichetta precedentemente calcolata sui nuovi campioni (nel caso d'esempio `y_test`), con quella che il classificatore calcola (`y_pred`).
Con questo tipo di algoritmi di machine learning non si pretende una soglia del 100% di precisione, perché sono processi basati su stime statistiche semplificate, che tentano di rimuovere molto del rumore che un caso reale avrebbe.
Un classificatore viene considerato performante già sulla soglia di accuratezza intorno al 70-75%, proprio per il motivo dovuto al calcolo statistico.
Migliorare questo valore resta comunque possibile, concentrando l'attenzione principalmente sull'organizzazione del data set: avere, infatti, un insieme di dati meglio rappresentativo potrebbe aiutare il modello ad apprendere il comportamento di molti più casi, riuscendo meglio a categorizzare i nuovi campioni.
L'obiettivo è quello di costruire il minor numero di casi ambigui possibile, che sarebbero complessi da definire perfino per un essere umano.

Esistono comunque varie alternative all'utilizzo del `KNeighborsClassifier`, molto conosciute e largamente impiegate, come ad esempio il `RadiusNeighborsClassifier` ovvero un modello caratterizzato dall'implementare sempre algoritmi di machine learning con supervisione per cercare i vicini, ma in questo caso non si concentra sul *numero* di elementi simili ma sul **raggio di vicinanza**: partendo dalla classificazione di un elemento, si ha la necessità di specificare un raggio entro il quale cercare altri elementi.
È un tipo di applicazione più delicata della precedente, perché ha bisogno di numerosi studi sull'intorno di ogni elemento per definire l'area di interesse, tale che restituisca almeno un vicino, senza aumentare esponenzialmente il costo computazionale.
Ha la necessità, dunque, di inserire un ulteriore parametro `outlier_label`, che specifica l'etichetta che viene assegnata se l'elemento non trova nessun vicino che rispetti il raggio di distanza stabilito.

4.3 Studio elementi vicini e suggerimenti

Una volta che il modello a classificatore è stato creato e valutato dal punto di vista dell'accuratezza, viene impiegato per mettersi a confronto con elementi esterni, come il messaggio che l'utente vuole analizzare.

Per eseguire questa operazione è necessario categorizzare il nuovo messaggio, dopo aver estratto le caratteristiche rilevanti, e studiare il suo intorno per individuare elementi a lui simili, concentrandosi su quelli che hanno avuto esito positivo.

Utilizzando nuovamente la libreria Scikit-learn vengono sfruttate le funzioni disponibili, sul modello `KNeighborsClassifier`, tali che riescano a restituire un numero K di **vicini** prestabilito.

La funzione principale è `kneighbors`, definita come segue:

```
dist, ind = knnModel.kneighbors(X, n_neighbors, return_distance)
```

- *`X`*: rappresenta il data set, o meglio il messaggio da analizzare, tendenzialmente memorizzato in un DataFrame Pandas.
- *`n_neighbors:`* valore intero che rappresenta il numero di vicini da restituire. La quantità di default è quella stabilita durante la definizione del classificatore.
- *`return_distance:`* valore booleano che indica la possibilità o meno di restituire il vettore delle distanze. [22]

Questa funzione prevede due valori di ritorno: `ind` e `dist`, vettori rispettivamente contenenti gli indici e le distanze degli elementi dell'intorno.

Studiare gli elementi simili al messaggio inviato dall'utente è un processo delicato, che permette di analizzare le loro caratteristiche e implementare una metodologia tale che possa apprendere da essi, per fornire poi **suggerimenti** atti a migliorare, eventualmente, l'esito del messaggio iniziale.

Questo procedimento è stato implementato in più fasi: in primo luogo vengono utilizzati proprio i valori di ritorno della funzione, per accedere ai dati all'interno del data set di riferimento, tali che vengano salvati in un DataFrame a sé per facilitarne lo studio. Successivamente sono stati considerati dei metodi per filtrare i vicini, in modo da selezionare i campioni migliori da cui apprendere.

```
good_df = pd.DataFrame(columns=COLUMNS)

for i in range(0, len(ind[0])):
    if (labels[ind[0][i]] == 'good' and dist[0][i] <= 1.0):
        good_df.loc[len(good_df)] = DATA.iloc[ind[0][i]]
```

Nel frammento di codice sopra riportato viene mostrato come è stato possibile creare un nuovo DataFrame dei vicini (`good_df`) e quali sono stati i metodi di filtraggio applicati. `DATA` è il DataFrame di partenza con tutti i tweet del data set, in modo da essere usato per restituire il valore delle proprietà dei vicini, attraverso i loro indici, mentre la variabile `COLUMNS` specifica quali sono state le colonne, o meglio le caratteristiche, di riferimento per il modello (NIMG, NHASH, NMENTION).

Scorrendo l'insieme dei vicini, mediante la lunghezza del vettore degli indici, si selezionano solo i messaggi con una etichetta (`labels`) positiva, in quanto non ha rilevanza apprendere da un tweet con esito negativo, che rispettino però un limite di distanza dal messaggio originale.

Quest'ultimo vincolo è stato inserito per avere la certezza di studiare le caratteristiche di elementi effettivamente simili (quindi vicini) a quello che si sta studiando, per non incorrere nella possibilità di confrontare due messaggi con nette differenze.

La misura di questo raggio di distanza limite è chiaramente da calibrare in base al proprio insieme di dati; dopo alcuni test è stato preferito il valore `1.0`, in quanto valore di default anche per il `RadiusNeighborsClassifier`, ovvero il modello che basa la sua intera esecuzione esattamente su un raggio di distanza, in questo modo è possibile affermare con una certa sicurezza che gli elementi rimasti sono effettivamente i più vicini a quello di partenza.

Una volta eseguiti questi metodi di filtraggio si procede alla composizione degli eventuali **suggerimenti**.

Come visto nel capitolo precedente, viene considerata la media delle caratteristiche dei vicini positivi e, su questa, si valutano le proposte per cambiare il messaggio di partenza.

Si ricorda che viene dato un suggerimento per caratteristica, piuttosto che uno unico generale, in modo da fornire la massima libertà di scelta al manager che sta scrivendo il tweet.

Viene riportato di seguito un esempio eseguito sul numero delle immagini contenute.

```
if good_df.empty:
    sugg.append("No suggestions available")

else:
    med_img = int(sum(pos_df['NIMG']) / len(pos_df))

    if (med_img > nimg):
        sugg.append("Add an image")
    elif (med_img < nimg):
        sugg.append("Remove an image")
```

`good_df` è il DataFrame Pandas che contiene gli elementi positivi che hanno superato il filtro di distanza, che se dovesse risultare vuoto significherebbe che non si è in grado di formulare suggerimenti, mentre `nimg` rappresenta la quantità di immagini contenute nel messaggio da analizzare.

Infine `sugg` è un vettore che conterrà l'insieme dei suggerimenti, che dovranno essere mandati al client tramite il canale di comunicazione che li unisce.

Come è intuibile dal codice riportato, viene eseguito un confronto tra il valore medio del numero di caratteristiche dei vicini e quello che invece riporta il messaggio di partenza.

Se il valore medio risulta superiore rispetto a quello del messaggio, allora potrebbe essere favorevole aumentare la quantità di quella caratteristica specifica, per farla avvicinare alla media dei suoi vicini positivi.

È analogo il caso contrario: se la media dovesse risultare inferiore, potrebbe essere il caso di diminuire la quantità.

Questo procedimento verrà, poi, eseguito sui valori di tutte e tre le caratteristiche su cui il modello a classificatore è stato basato: NIMG, NHASH e NMENTION.

4.4 Costruzione generatore

Come analizzato nel capitolo precedente, una volta costruito il meccanismo per restituire i suggerimenti, è stato necessario implementare una prova sperimentale allo scopo di verificare l'effettiva efficacia delle proposte suggerite.
Viene così ideato e progettato un generatore randomico di tweet che, ripetutamente, analizza i messaggi, elabora i suggerimenti e ne esegue uno scelto in maniera casuale.

```
tweet = pd.DataFrame(columns=COLUMNS, dtype='int32')
for i in range (0, n_tweet):
      NIMG = random.randint(0, 4)
      NHASH = random.randint(0, max_hash)
      NMENTION = random.randint(0, max_mention)

      tweet.loc[i, 'NIMG'] = NIMG
      tweet.loc[i, 'NHASH'] = NHASH
      tweet.loc[i, 'NMENTION'] = NMENTION
```

Con il codice sopra riportato inizia la procedura di creazione del generatore di tweet. Nello specifico viene dichiarata la grandezza del ciclo di iterazioni, da `0` a `n_tweet`, ovvero il numero di tweet che si ha intenzione di creare e poi analizzare.
Successivamente si inizializzano le tre variabili che rappresentano le tre caratteristiche su cui viene basato il modello: numero immagini, hashtag e menzioni.
La scelta di questi valori è puramente randomica, da cui l'utilizzo della libreria Python `random`, ma è stato predefinito comunque un intervallo di valori entro cui stare, in modo da simulare un messaggio iniziale realistico.
Per quanto riguarda le immagini è necessario un valore massimo di 4, perché è il limite numerico fissato da Twitter stesso, mentre per gli hashtag e le menzioni è stato deciso di inserirli all'interno dei valori massimi del loro data set (`max_hash` e `max_mention`), precedentemente calcolati.

Una volta definiti questi valori numerici, è stata impiegata nuovamente la libreria Pandas per memorizzarli all'interno di un DataFrame (`tweet`), in modo da manipolarli nel modo più semplice possibile.

```
for j in range (0, 6):
      tweetArray = scaler.transform
                        (tweet.loc[i].values.reshape(1, -1))
      prob = knnModel.predict_proba(tweetArray)
      good.append(prob[0][1])

      sugg.append(getSuggestion(good_df, tweet.loc[i], True))
      ind_sugg_scelto = random.randint(0, 2)
      doAction (sugg[0], tweet.loc[i], ind_sugg_scelto,
                        max_hash, max_mention)

prob_good.append(good)
```

Con il codice riportato si definisce un secondo ciclo, interno a quello visto in precedenza, specifico per lo studio dell'accuratezza dei suggerimenti sul tweet appena generato.
Esso viene delimitato in un intervallo da `0` a `6`, in quanto si simula il riesame del messaggio 5 volte e, quindi, 5 suggerimenti.
In primo luogo viene riutilizzato lo stesso `scaler` usato in precedenza sul modello, in modo da normalizzare i dati del nuovo messaggio nella medesima modalità.
Viene poi utilizzata nuovamente la libreria Scikit-learn per sfruttare la funzione `predict_proba`, la quale restituisce un vettore composto dagli elementi che, rispettivamente, rappresentano la probabilità statistica del messaggio di rientrare in una o nell'altra categoria che, nel caso del progetto in questione, sono 'GOOD' oppure 'BAD'.
Essa è risultata fondamentale per questa prova sperimentale, perché mediante la memorizzazione del suo andamento è stato possibile definire o meno un miglioramento nel modello, in base al fatto che la probabilità di avere un esito positivo vada via via aumentando o diminuendo.
Essa viene dunque memorizzata nel vettore `good`, che registra il variare delle probabilità a seconda dei suggerimenti seguiti, per uno stesso messaggio.

Si attiva l'algoritmo di creazione delle proposte, mediante la chiamata alla funzione `getSuggestion`, la quale riceve come parametri il DataFrame degli elementi simili al messaggio di partenza (`good_df`), il tweet generato (`tweet`) e un indicatore booleano rappresentante il fatto che la chiamata venga fatta o meno dal generatore.
I suggerimenti restituiti dalla funzione vengono salvati nel vettore `sugg`, in modo da definire poi un indice randomico che permetta di sceglierne uno tra quelli memorizzati.
A questo punto viene invocata la funzione `doAction`, che permette di svolgere effettivamente il suggerimento scelto.
La preferenza tra le proposte suggerite è stata implementata in modo randomico, al fine di simulare al meglio l'esperienza di un utente, di cui non possiamo conoscere le preferenze.
La funzione `doAction` divide il procedimento di scelta, tra le azioni, a seconda che ci siano o meno suggerimenti disponibili: si ricorda, infatti, che il modello a classificatore può tornare anche il messaggio "`no suggestions available`", se non è stato in grado di trovare elementi simili positivi, tali che superino il vincolo di distanza.
In questo caso si sceglie comunque di eseguire un'azione, scelta anch'essa in maniera completamente randomica, proprio come farebbe un utente nel medesimo caso.
A questo metodo vengono anche passati i parametri, imposti come limiti, `max_hash` e `max_mention`, calcolati in precedenza, in modo da non superarli nel caso venga scelta casualmente l'opzione di aumentare una caratteristica.
Al termine dell'azione, viene riesaminato il messaggio, memorizzato nuovamente il valore della probabilità e scelto un ulteriore suggerimento, fino ad arrivare a 5 iterazioni totali.
Una volta che queste sono state eseguite, l'andamento globale del singolo tweet viene memorizzato in `prob_good`, ovvero un vettore che tiene traccia di tutti i messaggi e di tutti i valori di prestazione.
Esso mette in luce l'evoluzione della probabilità di ricevere un esito positivo, man mano che vengono eseguiti i suggerimenti scelti.

4.5 Applicazione Android

Questa sezione ha lo scopo di concentrare l'attenzione su quella che è stata l'applicazione Android, nelle fasi della sua realizzazione.
Si sviluppa tramite l'approfondimento della costruzione del canale di comunicazione con il server Python, tramite l'utilizzo della libreria Volley, fino ad arrivare a discutere l'intero suo processo di uso, pensata per essere semplice e intuitiva, adatta ad ogni tipo di utente che si troverà a confrontarsi con essa.

4.5.1 Comunicazione con server Python tramite Volley

Come visto nel capitolo 2, dove vengono discusse le tecnologie impiegate per il progetto proposto, Volley è una libreria basata sul protocollo HTTP, che ha lo scopo di gestire le connessioni di rete per le applicazioni Android.
Utilizzare una libreria per questo tipo di collegamenti ha numerosi vantaggi: dalla gestione delle richieste, alla assegnazione di priorità diverse, fino alla coordinazione di connessioni multiple.

```
1) <uses-permission android:name="android.permission.INTERNET" />
2) implementation 'com.android.volley:volley:1.1.0'
```

Per poter utilizzare questa libreria è necessario inserire l'autorizzazione all'utilizzo di internet, all'interno del file *Manifest.xml*, come mostrato dal punto uno sopra indicato, e aggiungere la dipendenza all'archivio stesso di Volley, all'interno del file *build.gradle*, come nel punto due.
Volley basa la logica di comunicazione mediante l'uso di classi, tra cui le più usate sono **Request** e **RequestQueue**.
La prima rappresenta l'oggetto "richiesta" all'interno del protocollo di comunicazione HTTP, quindi deve specificare la risorsa che l'utente domanda al server.

Ogni richiesta, poi, non verrà eseguita immediatamente, ma verrà inserita all'interno di una coda (RequestQueue), in modo da lasciare la libertà di schedulazione alla libreria Volley stessa.

All'interno della richiesta sono presenti due *listeners*, che rappresentano il successo o il fallimento di svolgimento della stessa.

Response.Listener rappresenta il codice eseguito in base alla ricezione di una risposta positiva dal server, mentre *Response.ErrorListener* contiene le istruzioni da eseguire nel caso in cui, all'interno della comunicazione, si siano verificati errori.

Volley implementa numerosi tipi di richieste, in base alla tipologia di dato da trasmettere o ricevere; nel caso specifico del progetto proposto sono state predilette le JSONObjectRequest, in quanto danno la possibilità di inviare oggetti in formato JSON.

```
RequestQueue requestQueue = Volley.newRequestQueue(this);
JsonObjectRequest obj = new JsonObjectRequest(
        Request.Method,
        URL,
        new JSONObject(),

        new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
                Log.d("VOLLEY", response.toString());
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Log.e("VOLLEY", error.toString());
            }
        })

 requestQueue.add(obj);
```

Il codice sopra riportato mostra un esempio per la costruzione di una generica richiesta JSON, tramite l'ausilio della libreria Volley.

In primo luogo viene creata la coda di richieste che verranno inviate al server (`RequestQueue`), seguita dalla realizzazione della richiesta specifica che, nel caso dell'esempio, è di tipo JSONObject.

Ogni richiesta ha determinati parametri che la contraddistingue:

- `Request.Method`: specifica il tipo di funzione HTTP da soddisfare.

  Le più utilizzate sono GET, per richiedere una risorsa al server, oppure POST, per inviare dati ad esso.
- `URL`: è l'indirizzo al quale contattare il server.

  Nel caso del progetto specifico è stato utilizzato l'indirizzo *localhost* dell'emulatore Android, ovvero 10.0.2.2.

  È necessario poi precisare l'URL specifico, che corrisponderà ad una view Python, per accedere effettivamente al servizio messo a disposizione del server, nel momento in cui esso avrà l'indirizzo visibile dall'esterno (si ricorda essere 0.0.0.0).
- `Request`: definisce i dati, le risorse, da inviare al server.

  In caso di metodo di tipo GET, questo valore è `null`.
- `Listeners`: hanno lo scopo di gestire il valore di ritorno della richiesta stessa, sia per i casi di successo che di fallimento.

  Gli oggetti che il client riceve dal server vengono gestiti all'interno di queste funzioni. [20]

Al termine della sua definizione, ogni richiesta verrà aggiunta alla coda mediante la funzione `add` e sarà poi la libreria a pensare alla schedulazione in maniera automatica, tale che possa essere il più efficiente possibile.

4.5.2 Utilizzo applicazione

In questa sezione vengono mostrati i passaggi di un caso d'uso tipico dell'applicazione proposta, in modo da evidenziare quelle che sono state le scelte implementative dietro la sua realizzazione.

Essa è stata ideata per essere semplice ed intuitiva, al fine di essere accessibile ad una molteplicità di utenti con capacità tecniche differenti.

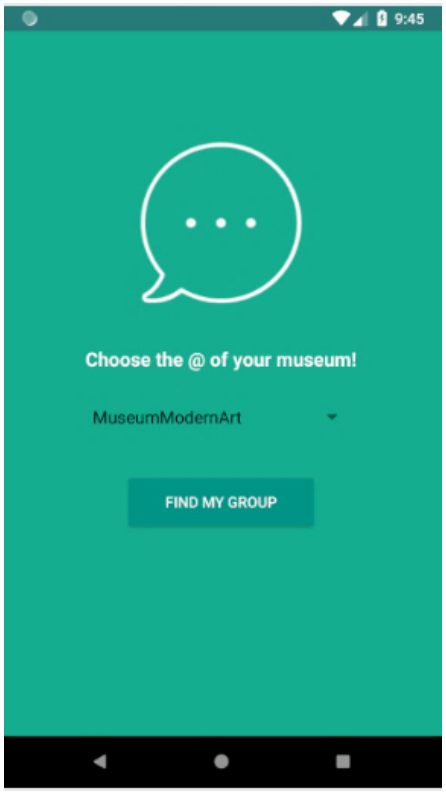

Figura 4.3 – Pagina iniziale

La Figura 4.3, che viene riportata di lato, mostra quella che è la pagina iniziale della applicazione creata con Android Studio.

Questa schermata riporta il logo appositamente creato ed un menù dal quale poter scegliere il proprio museo.

Questa lista viene riempita nel momento stesso in cui si avvia il programma, tramite la connessione con il server, e ha lo scopo di individuare il gruppo di cui fa parte il manager che sta accedendo e, di conseguenza, scegliere il data set di riferimento per il confronto dei messaggi.

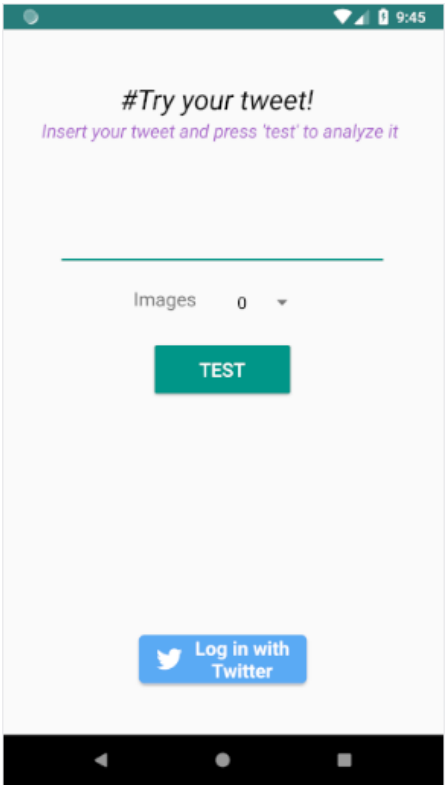

Figura 4.4 – Schermata per la scrittura del messaggio

Nella Figura 4.4 viene mostrata la schermata principale sviluppata per la creazione ed analisi del tweet che si ha intenzione di pubblicare.

È presente un'ampia area testuale, con contatore di caratteri incluso, un menù dal quale scegliere il numero di immagini da inserire ed un pulsante per attivare la comunicazione con il server per mandargli, dunque, il messaggio.

Viene scelto di concentrare la propria attenzione sul singolo numero di immagini, piuttosto che sul loro contenuto, in quanto per il classificatore non ha rilevanza.

In questa pagina è presente inoltre un pulsante di login a Twitter, che verrà utilizzato in un secondo momento, che dà la possibilità di accedere e pubblicare il proprio messaggio.

Una volta che l'utente ha inserito la propria idea di tweet, esso verrà inviato al server che avvierà il processo di **analisi**.

Quest'ultimo prevede l'individuazione del numero di immagini, hashtag e menzioni (caratteristiche su cui il classificatore è stato basato), e procede all'assegnazione di una etichetta indicante l'andamento, positivo o negativo che sia, del messaggio.

Figura 4.5 – Esempio suggerimenti

In caso di esito negativo il server provvederà ad inviare inoltre alcuni possibili suggerimenti, come mostrato dalla Figura 4.5.

Si ricorda che è stato scelto di restituire un suggerimento per ogni caratteristica presa in considerazione, in modo da dare la totale libertà all'utente di agire secondo quello che ritiene più corretto, anche in base al messaggio stesso.

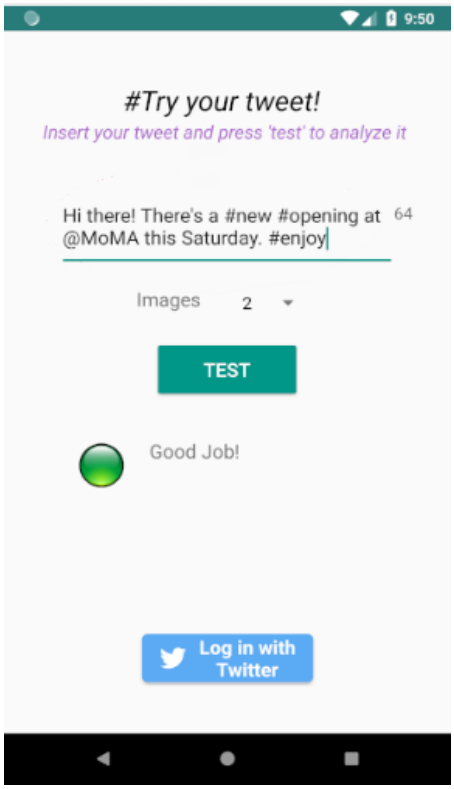

Figura 4.6 – Messaggio con esito positivo

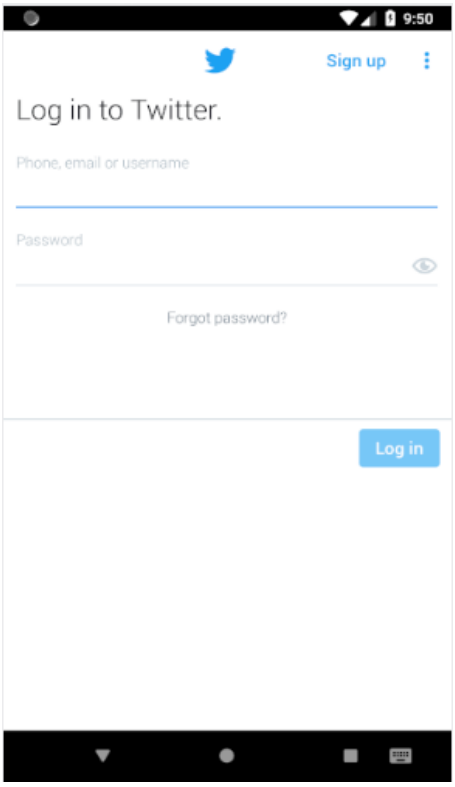

Figura 4.7 – Login a Twitter

Non essendo presenti limitazioni circa il numero di volte in cui è possibile riesaminare il messaggio, eventualmente si raggiungerà una combinazione con esito positivo, come mostrato nella Figura 4.6 a fianco.

In questa circostanza viene sottolineato il risultato mediante un indicatore verde e l'assenza di suggerimenti: il classificatore ha sentenziato il messaggio come un possibile candidato per ottenere successo online.

Una volta che il tweet è stato etichettato come positivo, o nel momento in cui l'utente si ritiene comunque soddisfatto del proprio messaggio, è possibile accedere a Twitter con le proprie credenziali, mediante l'apposito pulsate di login, e **pubblicare** il messaggio stesso.

La Figura 4.7 mette in luce esattamente la procedura di login appena descritta.

Questo collegamento è stato possibile mediante l'utilizzo del *Development Kit* che Twitter stesso rilascia, in formato open source e completamente gratuito, per gli sviluppatori Android.

Twitter, infatti, permette di inserire la potenzialità dell'intero social network all'interno del proprio progetto, mediante la semplice registrazione di quest'ultimo all'interno del loro sito per programmatori.

La procedura di login a **Twitter** fa parte delle numerose attività che sono possibili da implementare all'interno di un progetto in Android Studio.

```
1) implementation 'com.twitter.sdk.android:twitter:3.2.0'
2) Twitter.initialize(this);
```

In primo luogo è necessario andare ad inserire le dipendenze all'archivio nel file *build.gradle*, come mostrato dal codice sopra riportato, nel punto uno.
Successivamente deve essere inizializzato il *Development Kit* mediante la funzione `initialize`, come nel punto due, all'interno della funzione `OnCreate`.

Una volta eseguite queste due operazioni preliminari, è possibile usufruire a pieno del kit rilasciato da Twitter che, per il progetto proposto, andrà ad implementare la procedura di login e quella di pubblicazione del messaggio.

```
loginButton.setCallback(new Callback<TwitterSession>() {
    @Override
    public void success(Result<TwitterSession> result) {

        TwitterSession session = TwitterCore.getInstance()
                        .getSessionManager().getActiveSession();
        TwitterAuthToken authToken = session.getAuthToken();
    }

    @Override
    public void failure(TwitterException exception) {
        Toast.makeText(HomeActivity.this, "Authentication failed",
                Toast.LENGTH_LONG).show();
    }
});
```

Nel codice sopra riportato è presente l'implementazione del pulsante di **login**, rilasciato dalla libreria di sviluppo, che necessita la definizione di due funzioni principali: una eseguita in caso di successo e l'altra in caso di autenticazione fallita.
In quest'ultimo caso verrà mostrato un messaggio all'utente che lo notifica della mancata autenticazione a Twitter.
Se la procedura ha esito positivo, invece, viene creata una `TwitterSession` che contiene le informazioni dell'utente attivo in quel momento.

Una volta eseguito il login, tale che l'utente possa avere la propria sessione, è possibile recuperare il messaggio precedentemente analizzato dal classificatore e pubblicarlo.

Il `TweetComposer` rilascia due metodologie per inviare i propri messaggi, a seconda che sull'emulatore Android sia o meno installata l'applicazione Twitter stessa, facilmente recuperabile comunque attraverso i file APK disponibili.

È possibile iniziare la procedura di compilazione del tweet mediante un `Intent`, ovvero tramite la richiesta di esecuzione di una attività, definita in un componente e richiesta da un altro.

```
TweetComposer.Builder builder = new TweetComposer.Builder(this)
        .text(message.getFull_text());
builder.show();
```

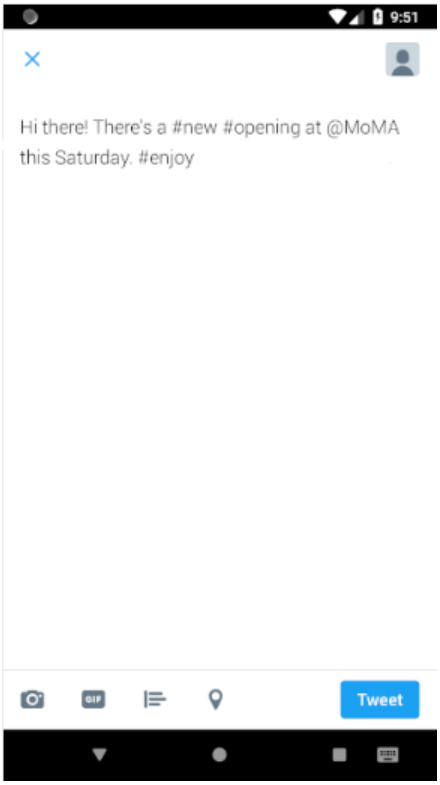

Figura 4.8 – Pubblicazione tweet

Con il codice sopra riportato è possibile creare l'`Intent` necessario per la **pubblicazione** del tweet, il cui risultato viene mostrato nella Figura 4.8.

`message` è una classe Java creata appositamente per contenere il messaggio, con hashtag, menzioni, immagini e l'intero testo inserito dall'utente, che verrà poi passato al `TweetComposer` per poterlo visualizzare nella pagina di pubblicazione. [24]

In questa fase vengono anche specificate le immagini, che precedentemente erano state ignorate, in quanto non era rilevante il loro contenuto.

Mediante l'utilizzo degli strumenti messi a disposizione da Twitter, è stato, quindi, possibile implementare un ambiente intuitivo per l'utente comunque abituato ad utilizzare il social network.

Tramite la finestra di pubblicazione il manager può ulteriormente modificare il messaggio, o aggiungere componenti prima non considerati, come la geolocalizzazione. Si trova davanti l'insieme delle funzionalità disponibili, in modo da poterle sfruttare a pieno per la propria campagna pubblicitaria.

Attraverso la realizzazione di questa applicazione, quindi, l'utente è stato accompagnato nelle fasi principali del progetto: dalla classificazione del messaggio, alla modifica di quest'ultimo per poterlo ridefinire ed avere un esito migliore sul pubblico, fino alla sua pubblicazione.

CAPITOLO 5

Prove sperimentali e risultati ottenuti

In questo capitolo verranno riprese tutte le operazioni svolte finora, dalla composizione del modello, alla classificazione dei nuovi campioni, fino alla restituzione di eventuali suggerimenti in grado di modificare l'esito del tweet inviato dall'utente.
Verranno riesaminate con l'ausilio di test e studi svolti, allo scopo di illustrare a pieno le scelte implementative eseguite.
Le prove sperimentali hanno avuto un impatto importante sullo sviluppo del progetto, in quanto hanno dato un indirizzamento delle preferenze di implementazione ed hanno accreditato ipotesi avanzate solo in fase di ideazione del progetto stesso.
Sono state eseguite verifiche in ogni punto di creazione del piano di studio, in modo da poter testimoniare il motivo delle scelte eseguite.

5.1 Test per la costruzione del modello

5.1.1 Test per la scelta delle caratteristiche

La prima fase di studi verte l'attenzione sulla costruzione del classificatore, essendo l'elemento principale del progetto proposto, in termini di scelta delle componenti da prendere in considerazione e miglioramenti delle prestazioni di accuratezza.
Come studio iniziale è stato necessario definire una strategia tale da rendere chiaro il procedimento di scelta delle caratteristiche su cui basare il modello, dunque su quegli attributi che risultano avere maggiore impatto sulla variabile target, ovvero il numero di like ricevuti.

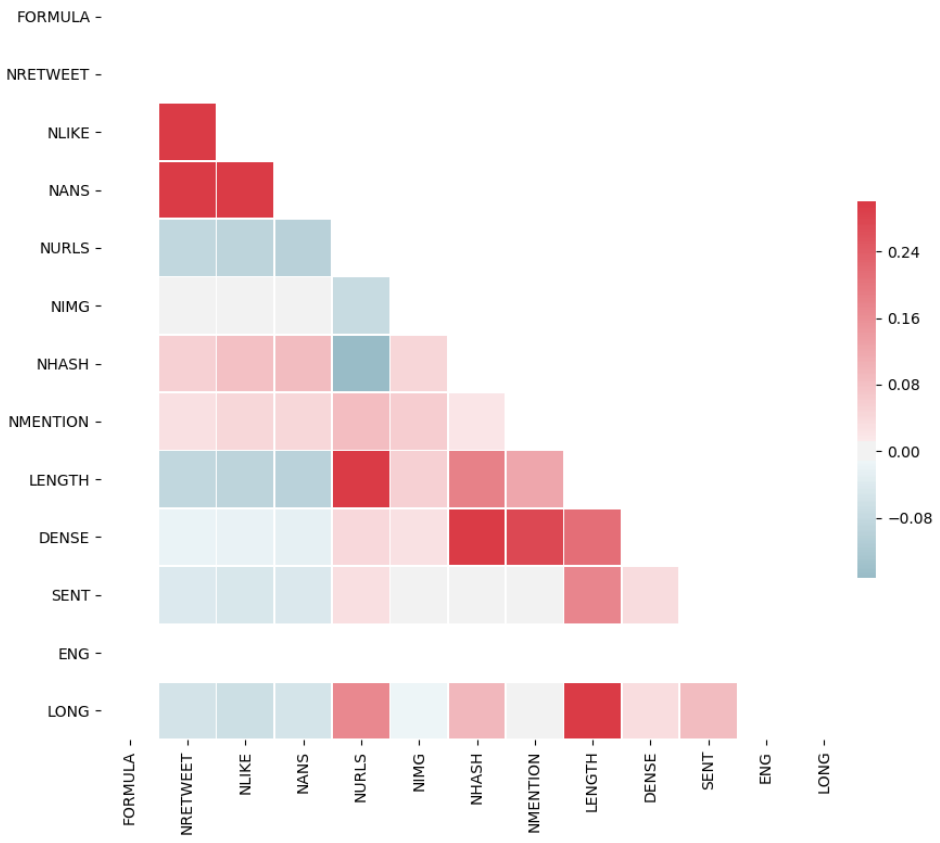

Figura 4.1 - Matrice di correlazione gruppo 1

È stata ripresa la matrice di correlazione dalla Figura 5.1, descritta nel terzo capitolo, in modo da sottolineare le relazioni tra le caratteristiche al fine di comprendere a pieno i motivi dietro la costruzione della Tabella 5.1, riportata successivamente.

Si pone l'attenzione tra le associazioni di quelle caratteristiche che rappresentano concetti molto simili (colore rosso accesso della Figura 5.1), dove potrebbe essere favorevole mantenere una sola delle due, per non appesantire il modello, e tra quelle che, invece, mostrano concetti antagonisti (colore blu della matrice) e occorrerà stabilire quale provvede ad avere un grado di accuratezza maggiore.

Proprio per questi motivi la matrice di correlazione è stata la base per la costruzione della Tabella 5.1 sottostante, che evidenzia i risultati ottenuti riguardo l'accuratezza del modello in base al gruppo di riferimento, e quindi al suo data set, e riguardo anche alle caratteristiche presentate.

Descrizione	*Caratteristiche*	*Accuratezza % (dopo 10 run)*					
		Gruppo					
		1	2	3	4	5	6
Tutte le caratteristiche iniziali	NURLS - NIMG - NHASH - NMENTION - LENGTH - DENSE - SENT - LONG	0,79	0,85	0,79	0,81	0,82	0,82
Correlazione LENGTH - LONG	NURLS - NIMG - NHASH - NMENTION - LENGTH - DENSE - SENT	0,80	0,84	0,78	0,81	0,82	0,82
	NURLS - NIMG - NHASH - NMENTION - DENSE - SENT - LONG	0,79	0,83	0,79	0,79	0,80	0,79
Correlazione LENGTH - NURLS	NURLS - NIMG - NHASH - NMENTION - DENSE - SENT	0,79	0,83	0,79	0,80	0,80	0,79
	NIMG - NHASH - NMENTION - LENGTH - DENSE - SENT	0,79	0,80	0,79	0,80	0,80	0,81
Correlazione DENSE - NHASH	NURLS - NIMG - NHASH - NMENTION - SENT	0,79	0,83	0,78	0,80	0,80	0,79
	NURLS - NIMG - NMENTION DENSE - SENT	0,74	0,81	0,76	0,79	0,79	0,79
Importanza di SENT o meno	NURLS - NIMG - NHASH - NMENTION - SENT	0,79	0,83	0,78	0,80	0,80	0,79
	NURLS - NIMG - NHASH - NMENTION	0,78	0,82	0,80	0,78	0,79	0,80
Relazione NHASH - NURLS	**NIMG - NHASH - NMENTION**	**0,77**	**0,78**	**0,77**	**0,79**	**0,78**	**0,80**
	NURLS - NIMG - NMENTION	0,74	0,79	0,75	0,78	0,78	0,76

Tabella 5.1 – Prova di accuratezza per caratteristiche e gruppi

I valori riportati dalla Tabella 5.1 sono stati calcolati come quantità medie dopo 10 esecuzioni del programma stesso, poi riportati in percentuale.
Si sceglie di calcolare la media delle prestazioni, in modo da ottenere valori pesati e più o meno costanti nel tempo.
È immediato notare che il modello ideato presenta un andamento generale molto positivo, con un intervallo di valori che oscillano dal 78-83% di precisione, indicante comunque che il classificatore è stato addestrato su porzioni di dati significativi, che lo hanno portato a categorizzare in modo corretto i nuovi campioni.
La prima analisi è stata effettuata sull'insieme delle caratteristiche di ogni messaggio, in modo da avere un punto di riferimento su cui basare le scelte implementative successive (prima riga della tabella).
La prima relazione che viene esaminata è quella tra gli attributi LENGTH e LONG, chiaramente rappresentativi di concetti simili, quindi si preferisce eliminarne uno per non appesantire il modello: si mantiene LENGTH che, tutto sommato, fornisce un grado di accuratezza maggiore.
Quest'ultima proprietà risultava essere in relazione anche con l'attributo NURLS, che viene preferito in quanto mantiene le prestazioni senza aggravarle.
Scorrendo la tabella si presenta anche l'associazione tra DENSE e NHASH, rappresentata da un colore rosso accesso nella Figura 5.1. Attraverso i sei data set, si ha notato che l'attributo NHASH portava ad avere un grado di accuratezza maggiore e, dunque, è stato preferito rispetto a DENSE.
L'attributo SENT è stato eliminato perché, come mostrato dalla Tabella 5.1, esso non risulta essere particolarmente incisivo sulle prestazioni, quindi si sceglie di non prenderlo in considerazione.
Nella parte finale della tabella viene riportato lo studio sull'ultima relazione individuata, ovvero tra le caratteristiche NHASH e NURLS.
Viene prediletto NHASH in quanto permette di restituire valori leggermente maggiori di accuratezza.

Una volta esaminate tutte le relazioni, vengono evidenziate le caratteristiche finali su cui il classificatore deve basarsi: NIMG, NHASH e NMENTION.
Esse rappresentano la combinazione migliore di proprietà tale da non averne in eccesso, che appesantirebbero il modello, ma in numero sufficiente per ottenere comunque buone prestazioni di accuratezza.

5.1.2 Test per la grandezza degli insiemi

Una volta decise le caratteristiche su cui basare il modello a classificatore, è stato necessario ponderare le scelte implementative circa la grandezza degli insiemi di dati rispettivi per training e testing del modello stesso.

Si ricorda che, con algoritmi di machine learning come quello impiegato, quindi il K Nearest Neighbors, è richiesta la divisione dei dati tale che compongano due insiemi distinti: il training set ha lo scopo di addestrare il modello al fine di renderlo il più preciso possibile, quindi tendenzialmente viene definito per avere la maggior parte dei dati.

Il testing set, invece, deve contenere i dati non ancora analizzati dal modello, che rappresentano quindi campioni nuovi, utilizzati per verificare il grado di precisione tra l'etichetta assegnata dal classificatore e quella precedentemente definita.

GRUPPO	GRANDEZZA TEST SET (%)	PRECISIONE (%)
1	0,15	0,76
	0,20	0,77
	0,25	0,77
2	0,15	0,77
	0,20	0,77
	0,25	0,78
3	0,15	0,78
	0,20	0,76
	0,25	0,76

GRUPPO	GRANDEZZA TEST SET (%)	PRECISIONE (%)
4	0,15	0,80
	0,20	0,79
	0,25	0,79
5	0,15	0,77
	0,20	0,78
	0,25	0,78
6	0,15	0,80
	0,20	0,80
	0,25	0,80

Tabella 5.2 – Confronto grandezza training e testing set

La Tabella 5.2 ha lo scopo di mettere in evidenza alcune prove eseguite, per meglio comprendere la divisione che avrebbe portato al risultato migliore.
Anche in questo caso i valori delle prestazioni sono stati calcolati come valore medio dopo 10 esecuzioni del programma, per poi riportarli in percentuale.
Lo scopo di questo test è stato quello di mostrare la variazione di accuratezza in base alle dimensioni rispettivamente concesse ai due insiemi.
Si confronta nell'intervallo di valori tra il 15-25% di grandezza per il testing set, assegnando la dimensione di conseguenza per il training set, per ognuno dei sei insiemi di dati precedentemente definiti.
Queste quantità sono le più usate in quanto, tendenzialmente, si vuole destinare la maggior parte dei dati all'insieme di training, al fine di ottenere un modello più preciso possibile.
Dato che non si sono verificate sostanziali variazioni di precisione del modello, durante l'esecuzione di questa prova sperimentale, viene deciso di partizionare i dati mediante una divisione del 20% per il testing set e il restante 80% per il training.

5.1.3 Test per la grandezza delle code

È stato eseguito uno studio anche sulla dimensione delle code, che definiscono rispettivamente l'insieme di riferimento per l'etichetta 'GOOD' e 'BAD', come mostrato dalla Tabella 5.3 sotto riportata.

Si ricorda che questo tipo di gestione a code è stato introdotto per evitare di analizzare messaggi ambigui, complessi anche per esseri umani, dunque si considera solo una percentuale degli elementi che hanno ricevuto più like (variabile target) come insieme di riferimento per i messaggi positivi.

Un procedimento analogo verrà, quindi, eseguito anche per quei tweet di riferimento per l'insieme 'BAD'.

GRUPPO	GRANDEZZA CODA (%)	PRECISIONE (%)
1	0,20	0,77
	0,30	0,69
	0,40	0,64
	0,50	0,62
2	0,20	0,77
	0,30	0,69
	0,40	0,65
	0,50	0,65
3	0,20	0,76
	0,30	0,69
	0,40	0,60
	0,50	0,56

GRUPPO	GRANDEZZA CODA (%)	PRECISIONE (%)
4	0,20	0,79
	0,30	0,68
	0,40	0,61
	0,50	0,60
5	0,20	0,78
	0,30	0,71
	0,40	0,64
	0,50	0,60
6	0,20	0,80
	0,30	0,76
	0,40	0,76
	0,50	0,71

Tabella 5.3 – Confronto grandezze code NLIKE

La Tabella 5.3 è stata ideata al fine di aiutare nella scelta implementativa circa la grandezza di questa soglia, in percentuale.
Mediante la divisione dei tweet in gruppi viene mostrata la variazione del valore di accuratezza, a seconda delle scelte diverse per la grandezza delle code.
Comprensibilmente si prevede un notevole calo delle prestazioni, corrispondente all'aumento della grandezza dell'intervallo per le code, in quanto vengono analizzati sempre più casi ambigui e, di conseguenza, il classificatore apprende un numero minore di comportamenti, risultando meno accurato nella previsione delle etichette.
Questo andamento previsto si è infatti verificato, come mette in luce la Tabella 5.3.
Si ha una accuratezza iniziale intorno al 76-80%, fino poi a diminuire di quasi 20 punti percentuali ed ottenere una precisione che si aggira sul 56-60%.
Per avere, dunque, un equilibrio tra una grandezza tale che possa portare una precisione accettabile, ma restare comunque sufficiente per addestrare il modello, viene scelta la soglia del 20%.

5.1.4 Test per la scelta di K

Dopo questi primi tre studi è stato possibile, quindi, identificare le caratteristiche su cui basare il modello, la grandezza degli insiemi testing e training per i dati e la soglia per distinguere i messaggi con etichetta 'GOOD', rispetto a quelli 'BAD'.
Con queste scelte implementative iniziali si ottiene già un classificatore performante, in grado di gestire comunque sei data set distinti, con esiti soddisfacenti.

Come visto nel capitolo precedente, è stata ideata una prova sperimentale per identificare il valore ottimale di K, inteso come numero di vicini, in fase di apprendimento del modello tale che, attraverso diverse iterazioni, sia possibile stabilire la quantità corrispettiva all'accuratezza maggiore.
Per ogni data set, dunque per ogni gruppo di musei, si esegue un ciclo che, ripetutamente, assegna valori di K crescenti ad un modello e ne memorizza i valori di accuratezza. Al termine di queste iterazioni viene effettuato il controllo sull'indice che ha ottenuto il valore massimo di precisione e verrà definito quello come K ottimale, per la specifica esecuzione.
Si ricorda che ogni modello viene creato in un ciclo a sua volta, definito da `0` a `N_RUN`, tale che si possa memorizzare poi un valore di accuratezza medio delle varie esecuzioni.

```
accuracy = []
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    accuracy.append(metrics.accuracy_score(y_test, pred_i))
```

Si riprende il codice già mostrato nel capitolo precedente, in modo da ribadire meglio il processo di iterazioni, con indice crescente come sopra riportato, al fine di definire il valore ottimale di K per il modello in fase di apprendimento.

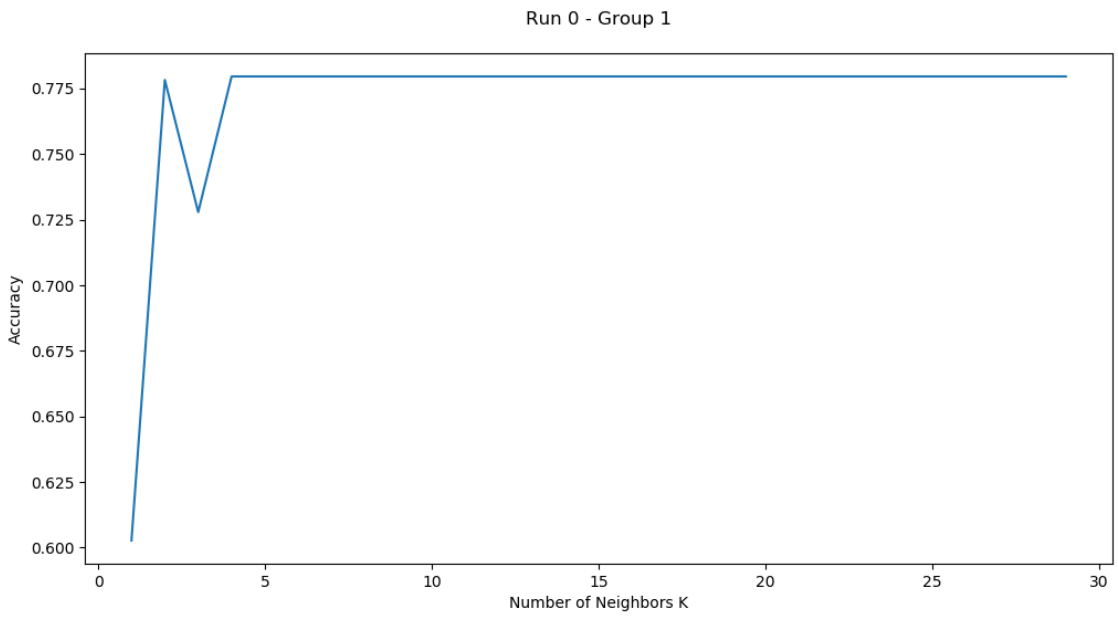

Figura 5.2 – Prova sperimentale per il K ottimale sul gruppo 1

Il grafico mostrato dalla Figura 5.2 specifica come l'andamento del vettore `accuracy`, come definito dal codice precedente, vari nel tempo a seconda del valore che si sceglie di assegnare a K.

Questo, nel dettaglio, è il solo grafico riguardante il primo ciclo di *run*, eseguito sul gruppo uno, riportato come esempio del ragionamento applicato.

Tutto sommato presenta una tendenza prettamente regolare, quindi al variare di K non vengono percepite sostanziali differenze di accuratezza, soprattutto con valori da 5 a 30. Per avere meno peso computazionale possibile, si orienta la scelta sul primo valore di K con la massima precisione che, nel caso riportato dalla Figura 5.2, è pari a 3.

Avendo elaborato l'algoritmo di divisione dei dati tale che abbia un seme numerico diverso su cui basare la randomizzazione, ad ogni iterazione, è possibile avere K ottimali diversi, all'interno dello stesso data set, per cicli di *run* diversi, come viene mostrato dalla Figura 5.3 sottostante riguardante la quinta esecuzione, sempre riferito al primo gruppo.

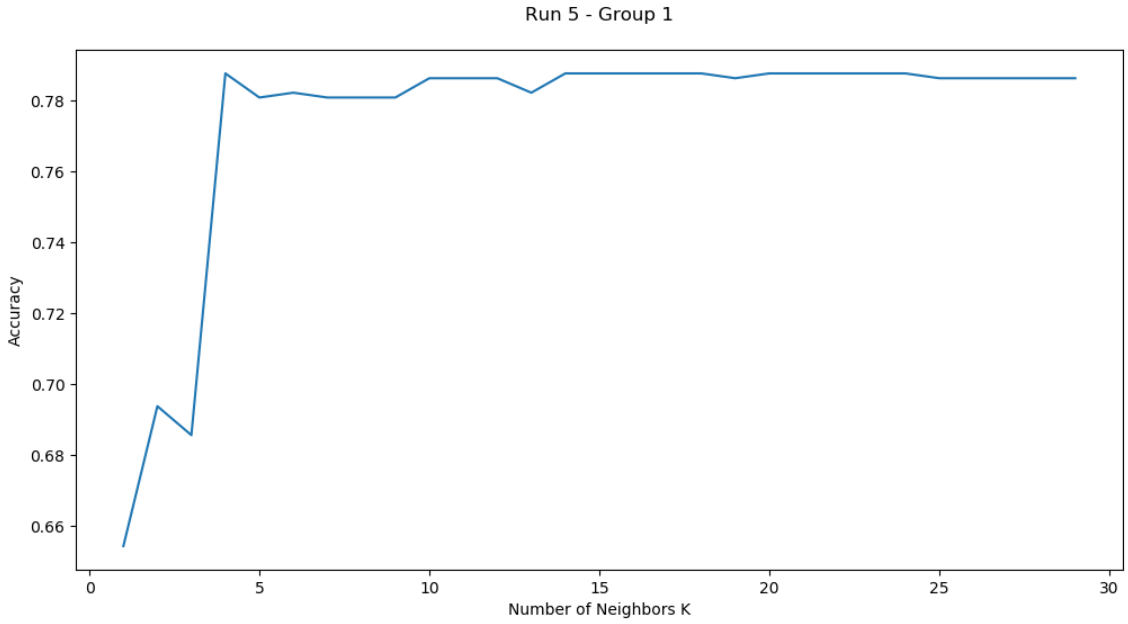

Figura 5.3 – Seconda prova sperimentale su gruppo uno

La Figura 5.3, infatti, presenta un andamento meno regolare delle prestazioni, in base alla scelta del valore K, probabilmente dovuto alla divisione diversa dei dati rispetto al primo *run*, come in Figura 5.2.

Nel caso preso in esame si individua velocemente la quantità ottimale dell'indice, pari infatti a 5.

Infine si ricorda che queste prove sono state eseguite tali che, per ogni ciclo di esecuzione viene trovato il valore ottimale e, solo al termine, si calcola la media delle prestazioni, definite per ogni data set e dunque per ogni modello.

N° Run	Gruppo					
	1	***2***	***3***	***4***	***5***	***6***
1	3	1	1	9	11	14
2	3	9	5	11	25	28
3	1	11	5	12	7	18
4	16	25	1	13	4	17
5	10	17	5	24	21	6
6	3	11	10	11	5	14
7	8	7	1	12	3	20
8	26	11	5	15	5	15
9	16	5	9	11	5	18
10	20	7	15	7	9	3

Tabella 5.4 – Confronto dei valori ottimali per K

La Tabella 5.4 sopra riportata mostra i valori ottimali di K per ogni iterazione, divisi per i gruppi di dati di riferimento.

Queste quantità sono il risultato di tutti i test effettuati in precedenza e illustrati dai grafici come in Figura 5.3.

Viene fissato un intervallo per il confronto dei valori che va da `0` a `30`, in modo da avere una larga quantità di prove da eseguire, ma senza ottenere K molto elevati che appesantirebbero notevolmente le prestazioni del modello.

5.2 Test per l'accuratezza dei suggerimenti

Al termine degli studi sperimentali appena descritti, il classificatore risulta completo ed in grado di etichettare e definire i nuovi messaggi che l'utente gli invia tramite l'applicazione Android, con un'accuratezza piuttosto elevata.
È stato scelto di aumentare l'interazione con l'utente stesso, fornendogli la possibilità di ricevere suggerimenti atti a migliorare l'ipotetico impatto che il suo messaggio avrebbe sul pubblico.
Il procedimento ideato per la realizzazione di essi è stato descritto anche dal capitolo precedente: si categorizza il messaggio dell'utente, attraverso le caratteristiche su cui si basa il modello a classificatore, si isolano i suoi messaggi vicini, da cui studiare gli attributi e, infine, si confronta la media dei valori delle singole caratteristiche dei tweet simili, con la quantità che possiede il messaggio iniziale.
Si ricorda che verrà fornito un suggerimento per ogni caratteristica, tale che possa essere poi l'utente a decidere su quale, secondo lui, è meglio agire.

5.2.1 Analisi delle prestazioni del generatore

In questa sezione vengono analizzate le prestazioni che il generatore di messaggi è riuscito ad ottenere su di essi, andando a sfruttare la memorizzazione dei valori nel vettore `prob_good`, visto nel capitolo precedente, che tiene traccia appunto della variazione della probabilità di ottenere esiti positivi o negativi.
Vengono mostrati di seguito i sei test eseguiti su ognuno dei data set di riferimento, allo scopo di visualizzare l'effettivo andamento dei messaggi, una volta che seguono determinati suggerimenti elaborati dall'algoritmo.
Ogni punto rappresenta il valore medio di tutti i messaggi generati (50, nel caso dell'esempio), dal punto di vista della probabilità stessa.
Il punto `0` raffigura il messaggio originale, quello generato in maniera randomica, mentre gli altri valori sull'asse x rappresentano l'andamento dei tweet dopo l'esecuzione dei suggerimenti.

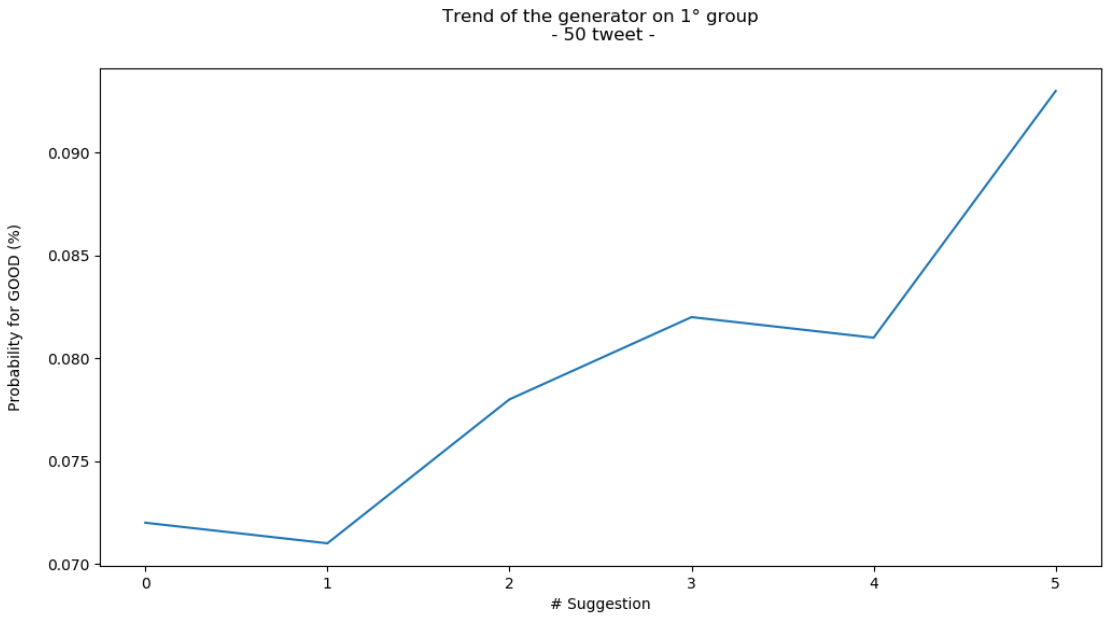

Figura 5.4 – Tendenza messaggi generati sul gruppo 1

Nel caso dell'esempio sopra riportato dalla Figura 5.4 è stato usato il data set del gruppo 1, per riferimento, nel quale si nota una tendenza positiva di sviluppo.

Emerge chiaramente un picco di crescita, tra il quarto ed il quinto suggerimento, ed in generale un trend crescente del grafico stesso.

Tuttavia è possibile affermare un valore piuttosto mediocre per la possibilità di un esito positivo, per i tweet generati in questo gruppo, in quanto comunque raggiungono un apice del 10% di probabilità positiva.

Risultati di questo genere non sono comunque da ritenere deludenti, data la natura completamente randomica dell'algoritmo, sia in fase di creazione dei tweet che nella scelta di suggerimenti.

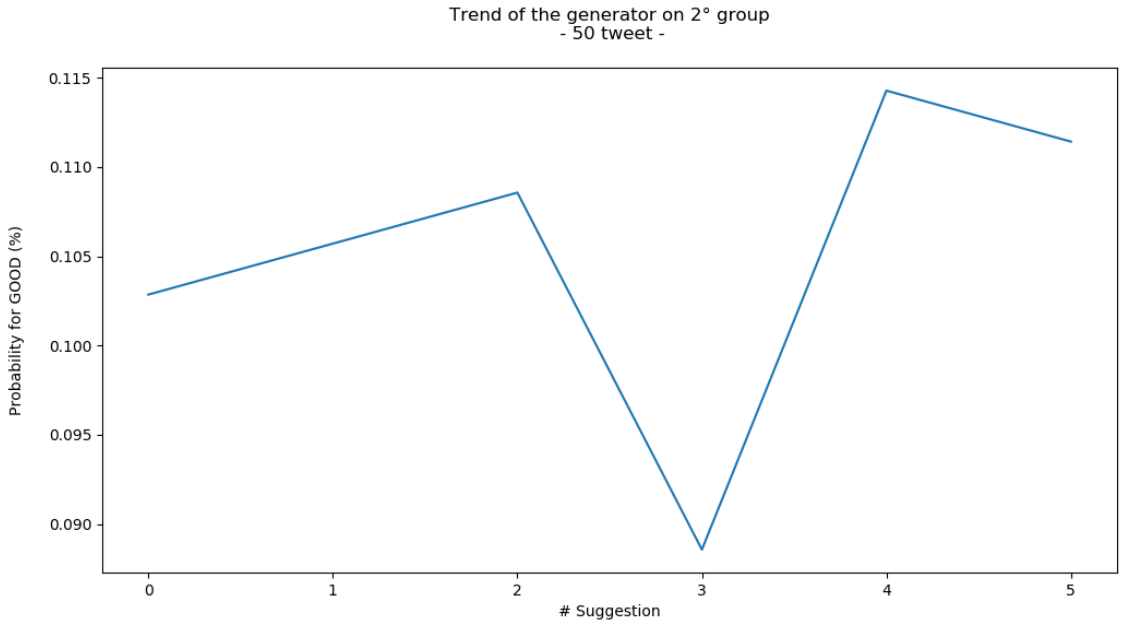

Figura 5.5 – Messaggi generati sul gruppo 2

La Figura 5.5 riporta la tendenza dei messaggi generati sul secondo gruppo di dati.

Sommariamente si afferma un andamento migliore rispetto a quello del gruppo precedente, anche se esso presenta un notevole peggioramento nell'intorno del terzo suggerimento.

Anche in questo caso è un risultato verosimile: rappresenta lo scenario di un utente che sceglie di modificare il proprio tweet in maniera imprevedibile, che però porta ad un drastico calo delle prestazioni del messaggio.

Andamento già in netto miglioramento dal suggerimento successivo, nel quale si ottiene il valore massimo di probabilità per un esito positivo.

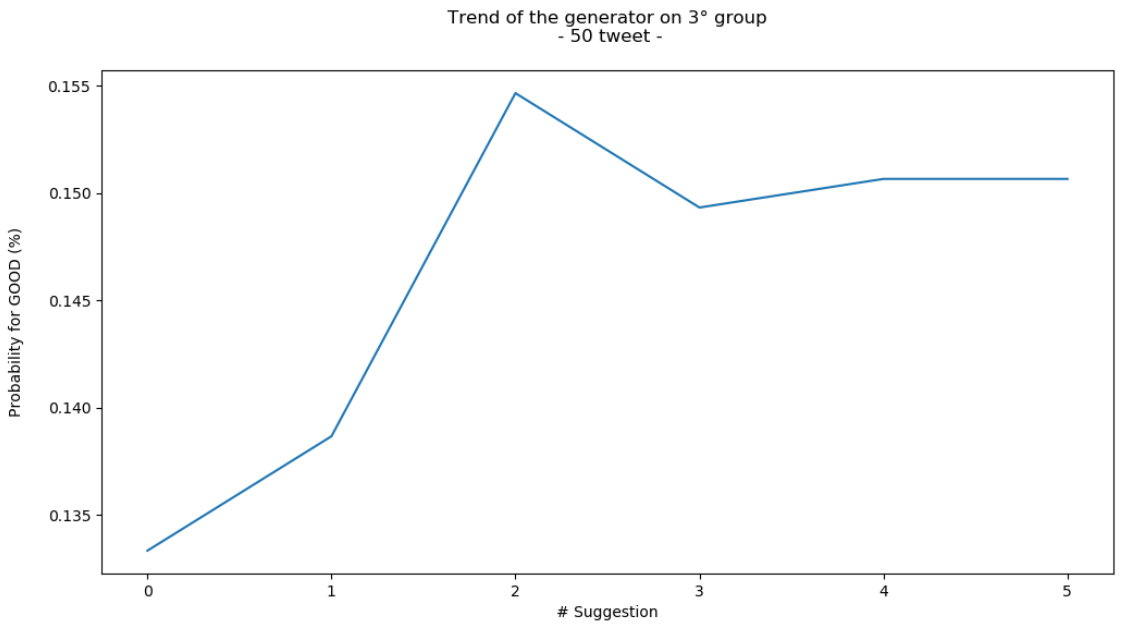

Figura 5.6 – Messaggi generati sul gruppo 3

Nella Figura 5.6 viene, invece, mostrato l'andamento dei messaggi generati che hanno avuto come riferimento il terzo data set.

Si raggiunge una probabilità di esito positivo massima nell'intorno del secondo suggerimento, con un valore del 16%, per poi avere una tendenza negativa o comunque minore nel seguire suggerimenti successivi.

Nel corso di questi esperimenti, non solo cambiano ovviamente i data set, ma anche i 50 messaggi generati ed il loro impatto iniziale: si nota infatti che, per questo insieme di dati, i messaggi iniziali hanno tutti una scarsa probabilità di successo (punto 0 del grafico), mentre in altri esempi ottenevano comunque un valore piuttosto accettabile.

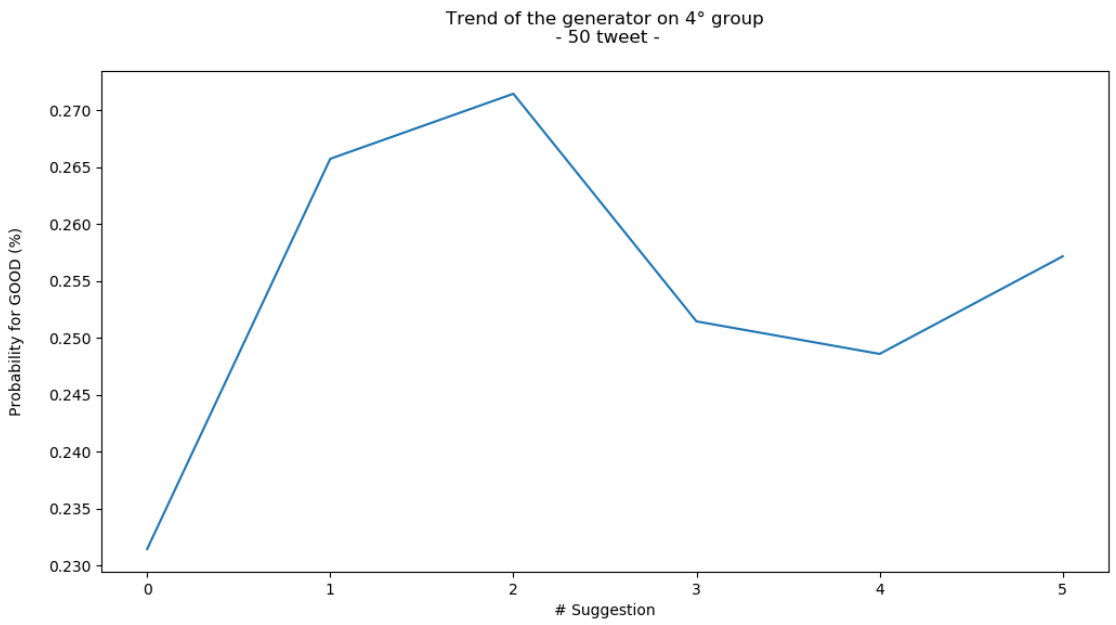

Figura 5.7 – Messaggi generati sul gruppo 4

Il grafico presentato dalla Figura 5.7 mostra la tendenza dei messaggi del generatore con il quarto insieme di dati come riferimento.

Sicuramente spicca una prima porzione di estrema crescita, da parte delle prestazioni dei messaggi stessi, che raggiungono quota 27% di probabilità di ricevere un esito positivo, per poi presentare un netto calo nella sua seconda metà.

Nel caso dell'esempio attuare la terza azione ha provocato una diminuzione di circa 2 punti percentuali, anche se nel complesso presenta una buona prospettiva di miglioramento per i messaggi iniziali, terminando la prova con il 25% di probabilità di risultare positivi.

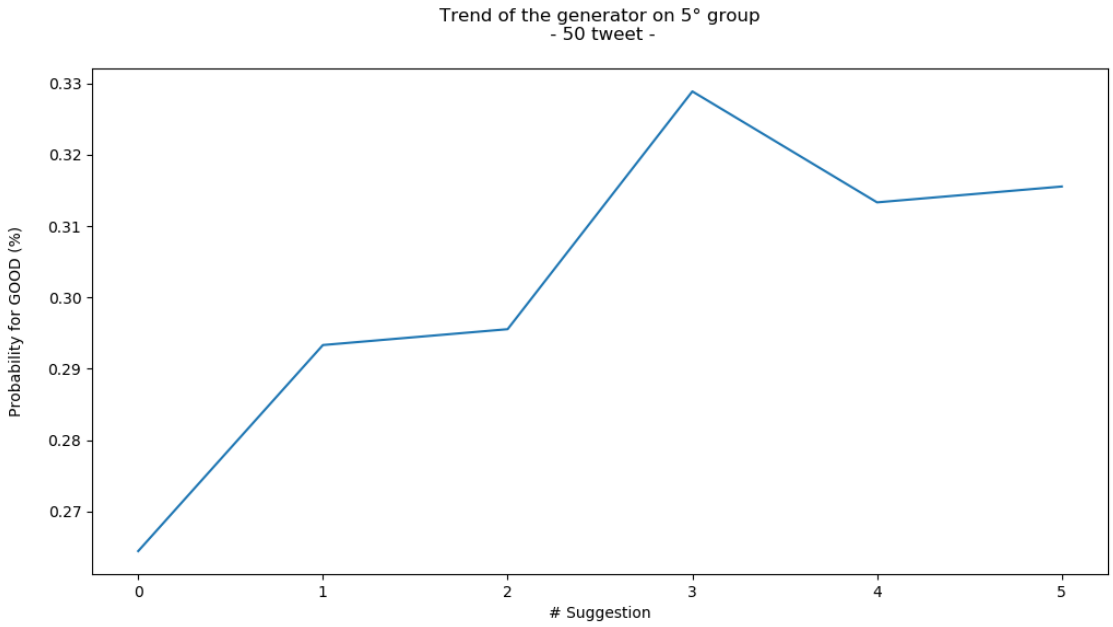

Figura 5.8 – Messaggi generati sul gruppo 5

Il quinto data set è quello che ha portato ad ottenere le prestazioni migliori da parte del generatore, come mostrato in Figura 5.8.

Si ottiene un picco di crescita nell'intorno della terza azione, con la quale si raggiunge il 33% di probabilità di un esito positivo globale dei messaggi.

L'andamento complessivo del grafico è in forte aumento, man mano che vengono seguiti i suggerimenti calcolati dall'algoritmo.

Si ricorda che la scelta delle azioni da seguire è completamente randomica, dunque ottenere una tendenza positiva di crescita significa essere stati in grado di elaborare un algoritmo dei suggerimenti adeguato al compito preposto.

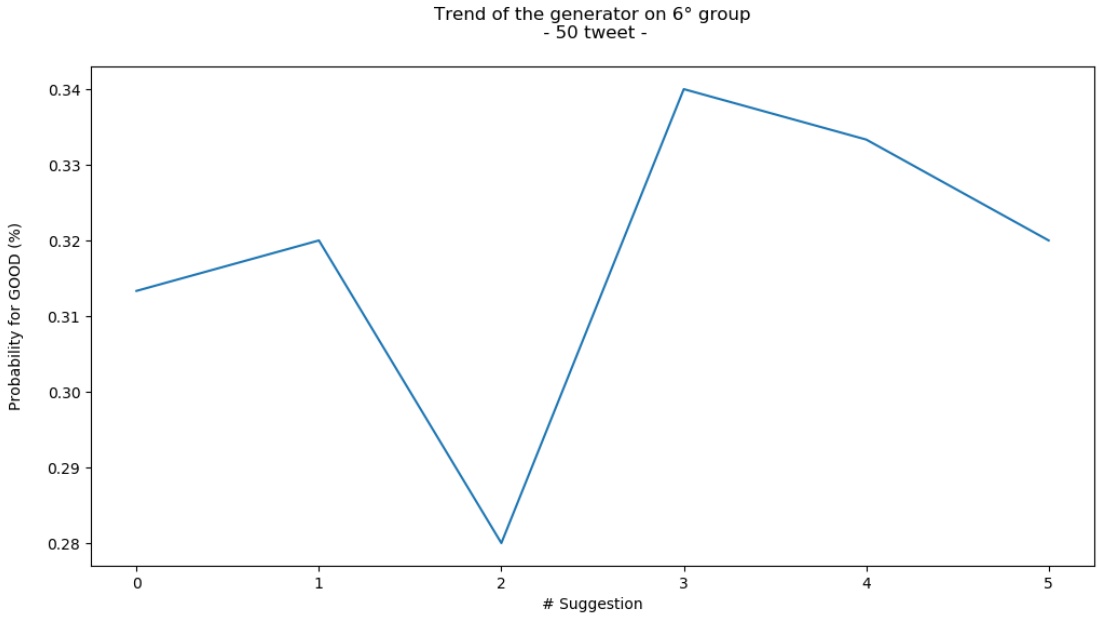

Figura 5.9 – Messaggi generati sul gruppo 6

L'ultimo caso da analizzare risulta essere quello dei messaggi generati sul gruppo 6 di riferimento, come mostrato in Figura 5.9.

La porzione più evidente del medesimo grafico è l'intorno del secondo suggerimento, punto nel quale si registra un forte calo della probabilità di ottenere un esito positivo, memorizzando il risultato sul 28%.

Si riottengono 6 punti percentuali andando ad eseguire la terza azione che, evidentemente, delinea una prospettiva migliore di successo rispetto alle precedenti.

Infine si sottolinea che, nonostante un andamento piuttosto irregolare del grafico, non è stato modificato in modo sostanziale l'esito dei messaggi: essi presentano un 31% di probabilità di essere categorizzati come positivi, già in partenza, risultato estremamente promettente, che però non è stato largamente modificato dai 5 suggerimenti seguiti, che li portano ad ottenere un 32% di probabilità finale.

CAPITOLO 6

Sviluppi futuri

In conclusione al progetto esposto nel presente elaborato, questo capitolo ha lo scopo di recuperare l'intero caso di studio, con i progressi effettuati finora, e di incentivare ulteriori analisi, al fine di ottenere un'applicazione sempre più performante.
Essa, infatti, ha dimostrato di riuscire nell'intento preposto in fase iniziale, ovvero quello di essere d'ausilio ai manager di musei che hanno la necessità di potenziare la propria presenza online, specialmente su Twitter.
Mediante i chiarimenti sui test effettuati nel capitolo precedente, è stato possibile dimostrare quanto il classificatore ideato sia performante ma anche flessibile, dato che cambia il proprio modello in base al data set di riferimento, in base al gruppo scelto.
Esso, non solo classifica il messaggio dell'utente, ma permette analisi ed osservazioni degli elementi simili (quindi vicini, nello spazio vettoriale) ad esso, in modo da apprendere da loro e definire potenziali suggerimenti.
È proprio questa capacità di adattamento che rende il modello a classificatore, quindi l'analisi testuale più in generale, un settore di studio estremamente innovativo.
In primo luogo andando ad orientare i propri esami sul potenziamento del classificatore, specificando sempre più la divisione dei dati, in modo da darne una visione più chiara e delineata, ad esempio inserendo più classi di categorizzazione, più ***label***.
Nel progetto proposto, infatti, il raggruppamento si può definire "binario": esistono solo due macro categorie, ovvero i messaggi con esito positivo o negativo.
Inserire più gradi di classificazione significherebbe avere un modello molto più preciso, seppur più complesso da definire; potrebbe essere possibile inserire anche un esito *neutrale*, per quei messaggi non estremizzati né in un caso né nell'altro, oppure dare dei *gradi* di positività o negatività, in modo da rendere sempre meno netta la differenza tra una categoria e l'altra e, così facendo, essere più inclusivi.

Un secondo aspetto che potrebbe essere portato avanti, nel corso di perfezionamento del modello proposto, pone l'attenzione sulle **immagini** utilizzate nei tweet.
Esse, infatti, come visto in precedenza, non hanno alcuna rilevanza in fase di classificazione del messaggio, in cui viene memorizzato il loro semplice numero.
Inserire un processo di elaborazione delle immagini potrebbe portare alla formazione di suggerimenti estremamente accurati, catturando l'attenzione dei propri lettori tramite strategie di marketing e comunicazione visiva, diffondendo marchi aziendali e fino alla fidelizzazione del proprio pubblico.

Questo modello a classificatore, inoltre, è stato progettato per agire su uno dei principali social network presenti, ovvero Twitter.
Esso può avere un ruolo meno marginale nell'applicazione, andando ad includere uno studio più mirato e in tempo reale.
Più nel dettaglio è possibile andare ad istruire il server con quelli che sono gli **hashtag** in tendenza, al momento della scrittura del tweet, in modo da favorire quelli per i suggerimenti.
Questa piccola accuratezza incrementerebbe notevolmente le prestazioni del modello, anche in termini di flessibilità dello stesso, e renderebbe i tweet molto più visibili, avendoli collegati agli argomenti in tendenza in quel periodo, raggiungendo più persone in meno tempo, dunque molto più efficaci per la pubblicità online.

Si conclude comunque ricordando le buone prestazioni già calcolate finora, viste nel dettaglio nel capitolo precedente, tali da dimostrare che lo studio e la manipolazione del testo risultino strumenti fondamentali per campagne pubblicitarie efficienti.
È stato, dunque, possibile creare un dispositivo in grado di aiutare manager di musei ad essere più incisivi su Twitter e, così facendo, ad aumentare il proprio pubblico.

Bibliografia

[1] REST API

https://phpenthusiast.com/blog/what-is-rest-api

[2] Formato JSON

https://json.org/json-it.html

[3] Apprendimento automatico

https://it.wikipedia.org/wiki/Apprendimento_automatico

[4] Tipi di Machine Learning

https://jacopokahl.com/i-tre-principali-tipi-di-machine-learning-apprendimento-supervisionato-non-supervisionato-e-per-rinforzo/

[5] Clustering

https://it.wikipedia.org/wiki/Clustering

[6] Apprendimento con rinforzo

http://www.andreaminini.com/ai/machine-learning/apprendimento-con-rinforzo

[7] Text Classification

https://monkeylearn.com/text-classification/

[8] Distanza di Euclide

https://en.wikipedia.org/wiki/Euclidean_distance

[9] Classificatore di Naïve-Bayes

https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

[10] Algoritmo KNN

https://www.datacamp.com/k-nearest-neighbor-classification-scikit-learn

[11] Definizione Sistema Operativo

https://it.wikipedia.org/wiki/Sistema_operativo

[12] Sistema Android

https://it.wikipedia.org/wiki/Android

[13] Definizione di SDK

https://it.wikipedia.org/wiki/Software_development_kit

[14] Definizione di file APK

https://it.wikipedia.org/wiki/APK

[15] Definizione di ambiente di sviluppo

https://it.wikipedia.org/wiki/Integrated_development_environment

[16] Framework Flask

https://it.wikipedia.org/wiki/Flask

[17] Definizione di Web Framework

https://it.wikipedia.org/wiki/Framework_per_applicazioni_web

[18] Android Studio

https://developer.android.com/studio/intro

[19] Ciclo di vita di un'attività

https://docs.microsoft.com/it-it/android/app-fundamentals/activity-lifecycle/

[20] Libreria Volley

https://developer.android.com/training/volley

[21] Libreria Pandas

https://pandas.pydata.org/

[22] Libreria Scikit-learn

https://scikit-learn.org/stable/

[23] Libreria Matplotlib e Seaborn

https://seaborn.pydata.org/

[24] Twitter Development Kit

https://github.com/twitter-archive/twitter-kit-android/wiki/Getting-Started