Corso Basi di dati su larga scala e data mining
Prof. Valentina Poggioni

Progetto Evalita 2020:
Task B - Stereotype Detection

Università degli Studi di Perugia
Dipartimento di Matematica e Informatica
Corso Magistrale INTELLIGENT AND MOBILE COMPUTING

Studente: Aniello Coscione | Matricola: 327371
Studente: Luca Spaccini | Matricola: 327488

Anno Accademico 2020/2021

Sommario

Introduzione

La seguente relazione per prima cosa si incentrerà ad illustrare gli strumenti utilizzati per svolgere il progetto; successivamente, si concentrerà nella spiegazione del problema da risolvere; ed infine si mostrerà, attraverso gli strumenti messi a disposizione, la soluzione al problema. La soluzione al problema verrà approfondita mostrando, spiegando e confrontando i risultati ottenuti dai modelli.

Obiettivo del Progetto

Il progetto è stato preso tra le task presenti nel sito di EVALITA. EVALITA è una campagna di valutazione periodica del Natural Language Processing (NLP) e degli strumenti vocali per la lingua italiana.

La task scelta si chiama "**HaSpeeDe 2020**".

La prima edizione di HaSpeeDe consisteva nel classificare automaticamente i messaggi nei Social Media con un valore booleano che indica la presenza di Hate Speech (HS). Successivamente la task si è evoluta e HaSpeeDe 2020 si concentra su tre principali fenomeni di HS online che si riflettono lungo tre task.

La **task B** è l'obiettivo del nostro progetto e riguarda il **rilevamento di stereotipi**:

Consiste nella **classificazione binaria** volta a determinare la **presenza o** l'**assenza di uno stereotipo** verso un determinato target (tra immigrati, musulmani o rom).

Come definito nel Merriam Webster Dictionary, lo stereotipo è "un'immagine mentale standardizzata che è tenuta in comune dai membri di un gruppo e che rappresenta un'opinione semplificata, un atteggiamento prevenuto o un giudizio acritico".

Considerando queste caratteristiche, lo Stereotipo può essere utilizzato per esprimere messaggi di odio o offensivi in maniera più sottile, ostacolando il corretto riconoscimento dell'Hate Speech. In questa prospettiva, il Task B mira a potenziare l'indagine sui suoi eventi soprattutto in un contesto odioso.

Quindi in sostanza: dato un messaggio, si deve decidere se il messaggio esprime Stereotipo o meno.

Strumenti e librerie utilizzate

GOOGLE COLAB

Google Colab è uno strumento gratuito presente nella suite Google che consente di scrivere codice python direttamente dal proprio browser.

Una piattaforma online che offre un servizio di cloud hosting per **notebook Jupyter** dove creare ricchi documenti che contengono righe di codice, grafici, testi, link e molto altro.

Si può creare un documento Google Colab direttamente da Google Drive e, proprio come un qualunque documento in G Suite, può essere condiviso con altri utenti che hanno la possibilità di modificarlo e lasciare commenti direttamente nel notebook.

Il notebook Jupiter verrà poi eseguito su macchine virtuali di server Google. Ciò consente di **svincolarsi dalla parte hardware** e di **concentrarci solamente sul codice Python** e sui contenuti che si vuole integrare nel notebook.

Le macchine virtuali messe a disposizione in Google Colab ospitano un ambiente configurato che consente di concentrarsi sin da subito sui progetti di Data Science: sono presenti numerose librerie Python, tra cui moltissime di Data Science come **Keras** e **Tensorflow**, si può usufruire di **GPU** e **TPU** per dare **boost computazionali** importanti ai nostri lavori, per esempio nell'implementazione di reti neurali con Tensorflow.

In Google Colab le risorse che vengono messe a disposizione agli utenti sono limitate, e variano a seconda delle fluttuazioni nella domanda. Per esempio un giorno si potrebbe venire assegnati ad una macchina virtuale con 32 GB di RAM e il giorno successivo ad una macchina virtuale con 12 GB di RAM.

Se si ha bisogno di maggior stabilità, di macchine più performanti o di accedere a GPU e TPU più potenti, è possibile utilizzare la licenza a pagamento, ad oggi disponibile solamente negli Stati Uniti, presente la versione Pro di Google Colab.

Report facilmente realizzabili, disponibilità di moltissime librerie Python di Data Science già presenti, il boost computazionale offerto dalle GPU e TPU uniti alla semplicità di condivisione e collaborazione garantita G Suite: Google Colab è uno strumento utilissimo per tutti quelli che si occupano di Data Science.

PYTHON

Python è un linguaggio di programmazione **dinamico orientato agli oggetti** utilizzabile per molti tipi di sviluppo software. Offre un forte supporto all'integrazione con altri linguaggi e programmi, è fornito di una estesa libreria standard e può essere imparato in pochi giorni.

Perché utilizzare Python nel Machine Learning

Python viene fornito con un'enorme quantità di librerie integrate. Molte librerie sono per Intelligenza Artificiale e Machine Learning. Alcune delle librerie sono **Tensorflow** (che è una libreria di rete neurale di alto livello), **scikit**-learn (per il data mining, l'analisi dei dati e l'apprendimento automatico), **pylearn2** (più flessibile di scikit-learn), ecc.

Python è ampiamente utilizzato nelle comunità scientifiche e di ricerca, perché è facile sperimentare rapidamente nuove idee e prototipi di codice in un linguaggio con una sintassi minima.

Molti programmatori Python possono confermare un sostanziale aumento di produttività e ritengono che il linguaggio incoraggi allo sviluppo di codice di qualità e manutenibilità superiori.

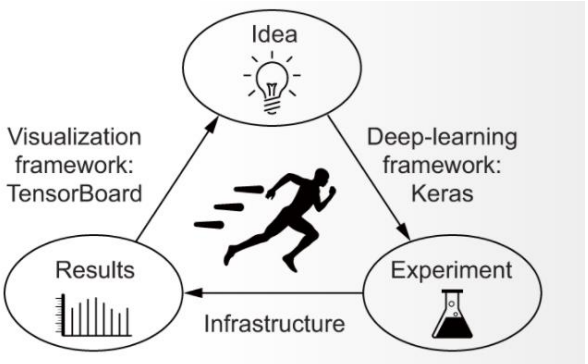
TENSORFLOW 2.5

TensorFlow è una piattaforma **open source end-to-end** per l'apprendimento automatico. Dispone di un ecosistema completo e flessibile di strumenti, librerie e risorse della community che consente ai ricercatori di promuovere lo stato dell'arte in Machine Learning e gli sviluppatori di creare e distribuire facilmente applicazioni basate su Machine Learning.

Crea e addestra facilmente modelli Machine Learning utilizzando **API intuitive** di alto livello come **Keras** con esecuzione rapida, che rende l'iterazione del modello immediata e un facile debug.

KERAS

Keras è una libreria open source per l'apprendimento automatico e le reti neurali, scritta in Python. È progettata come un'interfaccia a un livello di astrazione superiore di altre librerie simili di più basso livello, e supporta come back-end le librerie TensorFlow, Microsoft Cognitive Toolkit (CNTK) e Theano. Progettata per permettere una **rapida prototipazione di reti neurali profonde,** si concentra sulla **facilità d'uso**, la **modularità** e l'**estensibilità**. È stata sviluppata come parte del progetto di ricerca ONEIROS, e il suo autore principale è François Chollet, di Google.



Nel 2017 il team di TensorFlow ha deciso di supportare Keras ufficialmente. Chollet ha spiegato che Keras è stata pensata come un'**interfaccia** e non come una libreria stand-alone. Offre una serie di moduli che permettono di sviluppare reti neurali profonde indipendentemente dal back-end utilizzato, con un linguaggio comune e intuitivo.

Keras è il framework di deep learning più utilizzato tra i primi 5 team vincitori su Kaggle. Poiché Keras semplifica l'esecuzione di nuovi esperimenti, ti consente di provare più idee rispetto alla concorrenza, più velocemente.

SCIKIT-LEARN

Scikit-learn (precedentemente scikits.learn e noto anche come sklearn) è una libreria di machine learning software gratuita per il linguaggio di programmazione Python . Presenta vari **algoritmi** di **classificazione, regressione e clustering** tra cui support vector machines, random forests, gradient boosting, k-means e DBSCAN. È progettato per interagire con le librerie numeriche e scientifiche Python NumPy e SciPy.

Scikit-learn è in gran parte scritto in Python e utilizza NumPy ampiamente per operazioni di algebra lineare e array ad alte prestazioni. Inoltre, alcuni algoritmi di base sono scritti in Cython per migliorare le prestazioni. Le macchine vettoriali di supporto sono implementate da un wrapper Cython attorno a LIBSVM; logistic regression e linear support vector machines da un wrapper simile intorno a LIBLINEAR. In tali casi, potrebbe non essere possibile estendere questi metodi con Python.

Scikit-learn si integra bene con molte altre librerie di Python, come **Matplotlib** e plotly per la stampa, NumPy per array di vettorializzazione, Panda dataframes, SciPy, e molti altri.

NATURAL LANGUAGE TOOLKIT (NLTK)

NLTK è una **piattaforma** leader per la creazione di programmi Python per lavorare con i dati del linguaggio umano. Fornisce interfacce facili da usare per oltre 50 corpora (Un corpus è una collezione di testi selezionati e organizzati per facilitare le analisi linguistiche) e risorse lessicali come WordNet, insieme a una suite **di librerie di elaborazione del testo per classificazione**, **tokenizzazione**, **stemming**, **tagging**, **analisi** e **ragionamento semantico**, wrapper per librerie NLP di livello industriale e un forum di discussione attivo.

Grazie a una guida pratica che introduce i fondamenti della programmazione insieme ad argomenti di linguistica computazionale, oltre a una documentazione API completa, NLTK è adatto allo stesso modo per linguisti, ingegneri, studenti, educatori, ricercatori e utenti del settore. Soprattutto, NLTK è un progetto gratuito, **open source** e **guidato dalla comunità**.

L'elaborazione del linguaggio naturale con Python fornisce un'introduzione pratica alla programmazione per l'elaborazione del linguaggio. Scritto dai creatori di NLTK, guida il lettore attraverso i fondamenti della scrittura di programmi Python, del lavoro con i corpora, della categorizzazione del testo, dell'analisi della struttura linguistica e altro ancora. La versione online del libro è stata aggiornata per Python 3 e NLTK 3.

FASTTEXT

FastText è una libreria open source, gratuita e leggera che consente agli utenti di apprendere rappresentazioni di testo e classificatori di testo. Creata dal laboratorio di ricerca AI di Facebook. Il modello consente di creare un algoritmo di apprendimento non supervisionato o di apprendimento supervisionato per ottenere rappresentazioni vettoriali per le parole.

Tale libreria mette a disposizione vettori di parole pre-addestrati per 157 lingue, addestrati su Common Crawl e Wikipedia utilizzando fastText. Questi modelli sono stati addestrati utilizzando CBOW con pesi di posizione, nella dimensione 300, con caratteri n-grammi di lunghezza 5, una finestra di dimensione 5 e 10 negativi. Distribuiamo anche tre nuovi set di dati sull'analogia delle parole, per francese, hindi e polacco.

I vettori di parole pre-addestrati hanno dimensione 300. Attraverso il riduttore di dimensione è possibile decidere la dimensione che fa più al caso nostro.

Dataset

Il dataset di “HaSpeeDe 2020” include testi rivolti a gruppi minoritari come immigrati, musulmani e comunità rom, i cui relativi problemi sociali alimentano costantemente il dibattito pubblico e politico innescando Hate Speech. In quest'ottica i creatori del dataset hanno raccolto tweet e titoli di testata giornalistica utilizzando parole chiave specifiche relative alle citate minoranze italiane.

Il dataset fornito è un file di valori separati da tabulazioni (TSV) che include i seguenti campi:

1. Un numero che sostituisce l'ID tweet originale
2. Il testo
3. La classe di incitamento all'odio: 1 se il testo contiene incitamento all'odio e 0 in caso contrario
4. La classe degli stereotipi: 1 se il testo contiene stereotipi, 0 altrimenti.

Abbiamo utilizzato Pandas per leggere il Dataset e rimodellarlo in un Data Frame.

In questo caso i delimitatori di fine record sono **\t** oppure **\to**. Quindi abbiamo creato tre colonne: Testo, Odio, Stereotipo e convertito i valori in interi delle etichette.

```
df_cols = ["Testo", "Odio", "Stereotipo"]

df = pd.read_table('/content/drive/MyDrive/Progetto Evalita/haspeede2_dev_taskAB.tsv',
                   delimiter="\t|\t0",encoding='utf-8', names = df_cols, engine='python')
df = df.drop(df.index[0])

df.Stereotipo.replace(to_replace ="1", value = 1, inplace = True)
df.Stereotipo.replace(to_replace ="0", value = 0, inplace = True)

df = df[["Testo","Stereotipo"]]
df
```

Figura 1: Cella lettura e modellazione Dataset

Il Data Frame finale pronto per il pre-processing è composto solo dalla colonna Testo e Stereotipo.

	Testo	Stereotipo
2066	È terrorismo anche questo, per mettere in uno ...	0
2045	@user @user infatti finché ci hanno guadagnato...	0
61	Corriere: Tangenti, Mafia Capitale dimenticata...	0
1259	@user ad uno ad uno, perché quando i migranti ...	0
949	Il divertimento del giorno? Trovare i patrioti...	0
...	...	...
9340	Gli stati nazionali devono essere pronti a rin...	0
9121	Il ministro dell'interno della Germania #Horst...	0
8549	#Salvini: In Italia troppi si sono montati la ...	0
9240	@user @user Chi giubila in buona fede non ha c...	0
8000	I giovani cristiani in #Etiopia sono indotti d...	1

6839 rows × 2 columns

Figura 2: Composizione del Data Frame per l'elaborazione

STATISTICHE

Il numero totale dei record è: **6893**. La distribuzione di classe relativa ai record è:

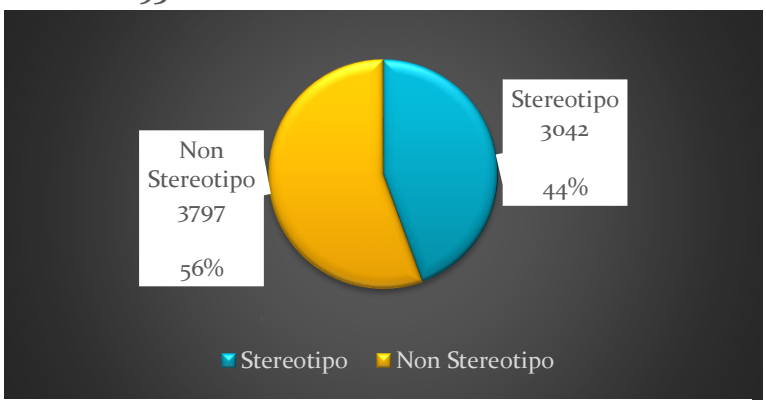
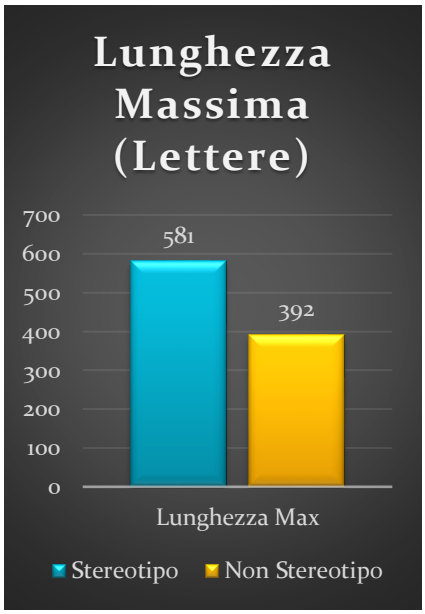
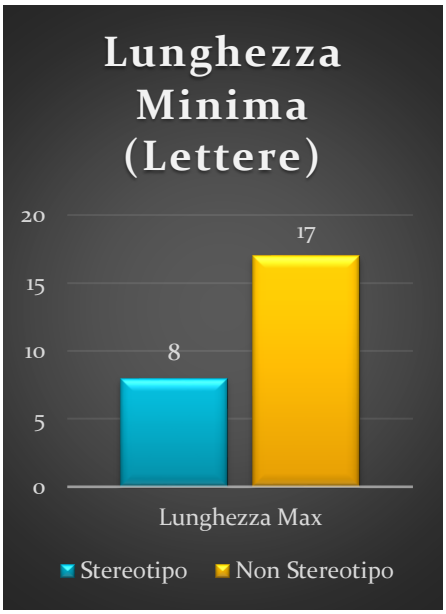


Figura 3: Distribuzione degli stereotipi nel training set

Il testo è rielaborato per cui la menzione dei nomi utente nei tweet e gli URL sono sostituiti con i segnaposto *@utente* e *URL*.





STUDIO DEL DATASET

Dopo aver visto la composizione del dataset e la sua distribuzione abbiamo analizzato alcuni record per verificare la correttezza. Tale operazione è stata essenziale per capire se ci trovavamo a lavorare con un dataset corretto e ben formato. Inoltre ci ha aiutato anche per la pulizia del testo.

Le considerazioni che sono emerse durante lo studio sono che non sempre un record viene classificato correttamente come stereotipo.

Per provare tale affermazione riportiamo alcuni esempi.

- Minorenne rom arrestato dopo furto in appartamento URL
- Allarme terrorismo: mille fantasmi a Milano, senza vera identificazione URL via @user
- Vivevano lungo i binari della ferrovia: sgomberati 13 rom | Il Cittadino di Lodi URL

Pre-processing

Esistono molti fattori che comprendono la qualità dei dati, tra cui accuratezza, completezza, consistenza, tempestività, credibilità e interpretabilità. Tali fattori devono essere sempre presi in considerazione dall'analista dei dati. La fase di pre-processing è un passo fondamentale cui bisogna porre tantissima attenzione perché vi dipendono i risultati di tutte le azioni successive e quindi i **risultati finali sono strettamente dipendenti da tale fase**.

PULIZIA TESTO

La pulizia del testo è stata eseguita tramite una funzione che racchiude essenzialmente tre procedure.

1. Procedura di **pulizia del testo** che riguarda:
    a. Conversione **emoji** in testo italiano così da poterle far prendere in considerazione dal modello
    b. Rimozione della **punteggiatura**, **caratteri speciali** e **simboli**
    c. Conversione del testo in **minuscolo**
    d. Rimozione delle **stopwords** italiane attraverso Corpus della libreria NLTK
    e. Rimozione delle parole con **lunghezza minore di 2** (Questa condizione deriva dallo studio del dataset dove abbiamo notato che la parola "rom" è molto frequente)
    f. Rimozione dei **numeri**
    g. Rimozione delle **combinazioni ricorrenti** caratterizzanti del dataset come per esempio @user, url, # ecc
2. Procedura di **pulizia ulteriore** del testo tramite la libreria **tweet-processor** che mette a disposizione pulizie dei testi in molteplici lingue tra cui l'italiano.
3. Procedura di **stemming** del testo attraverso SnowballStemmer della libreria NLTK.
    Lo stemming è un processo che riconduce una parola dalla sua forma flessa alla radice o tema (in questo post si userà il termine "tema"). Ad esempio, le parole "correre", "corro", "corriamo", "correremo" vengono tutte ricondotte al termine "corr", il tema appunto.
    Ricorriamo allo stemming perché nella classificazione dei testi, dove l'intento è quello di **capire** quale sia **l'argomento** di cui tratta un certo testo. È intuitivo pensare che se in un testo sono presenti le parole "correre", "corro", "corriamo" e "correremo" non si stia parlando di 4 argomenti diversi, ma invece di uno solo: la corsa, che potremmo associare appunto al tema "corr".

Esempio record originale:

"@user @user Questi analfabeti che schifate tanto sono quelli che con le loro tasse e le loro schiene mandano avanti il paese,fate proprio schifo con questo senso di superiorità. Nel frattempo imbarcate migranti ,futuri vostri elettori,tutti universitari! 😂😂😂😂"

Esempio record pulito con stemming:

"analfabet schif tant tass schien mand avant paes fat propr schif sens superior frattemp imbarc migrant futur elettor universitar faccinaconlacrimedigioi faccinaconlacrimedigioi faccinaconlacrimedigioi faccinaconlacrimedigioi"

Esempio record pulito senza stemming:

"analfabeti schifate tanto tasse schiene mandano avanti paese fate proprio schifo senso superiorità frattempo imbarcate migranti futuri elettori tutti universitari faccinaconlacrimedigioia faccinaconlacrimedigioia faccinaconlacrimedigioia faccinaconlacrimedigioia"

SUDDIVISIONE TRAINING/TEST SET

Il dataset una volta effettuata la pulizia è stato sottoposto ad una divisione in train e test set. La divisione è stata fatta attraverso la funzione ***train_test_split*** di **sklearn**. Tale funzione ci permette di divider in modo randomico un dataset in due sottoinsiemi.

I parametri utilizzati sono: **random_state**, che controlla la mescolanza applicata ai dati prima di applicare la suddivisione (assegnare un valore a tale parametro permette di non essere dipendenti dalle esecuzioni in quanto la suddivisione verrà effettuata sempre allo stesso modo); **test_size** impostato a "**0.2**". Questo vuol dire che avremo il **20% (1368) sul test set** e **l'80% (5471) sul training set**.

text_train, text_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

CREAZIONE DEL VOCABOLARIO

Per creare il vocabolario utilizziamo la classe Tokenizer di Keras. Questa classe permette di vettorializzare un corpus di testo, trasformando ogni testo in una sequenza di interi (ogni intero è l'indice di un token in un dizionario) o in un vettore in cui il coefficiente per ogni token potrebbe essere binario, basato sul conteggio delle parole ecc...

Per impostazione predefinita, tutta la punteggiatura viene rimossa, trasformando i testi in sequenze di parole separate da spazi. Queste sequenze vengono quindi suddivise in elenchi di token. Saranno quindi indicizzati o vettorializzati. "o" è un indice riservato che non verrà assegnato a nessuna parola.

Quindi, abbiamo applicato sia al Train che al Test la funzione **fits_on_text** che aggiorna il vocabolario interno sulla base di un elenco di testi.

Successivamente una volta aggiornato il vocabolario possiamo trasformare ogni record in una sequenza di numeri interi, quindi in vettori, tramite **texts_to_sequences**. Dato che "o" è un indice riservato, verranno prese in considerazione solo le prime *num_words-1* parole più frequenti. Perciò verranno prese in considerazione solo le parole conosciute dal tokenizer.

Una vola vettorizzati i record, dato che Keras preferisce input con la stessa lunghezza, effettuiamo il padding su tutti i vettori. La funzione che utilizziamo per fare questa procedura è **pad_sequences**.

```
#TO DO :
#Tokenize and transform to integer index
tokenizer = Tokenizer()

tokenizer.fit_on_texts(text_train)

X_train = tokenizer.texts_to_sequences(text_train)
X_test  = tokenizer.texts_to_sequences(text_test)

#TO DO: get some statistics
vocab_size = len(tokenizer.word_index) + 1 # adding 1 because of reserverd 0 index
```

Figura 4 creazione vocabolario

Caratteristiche del vocabolario e dei vettori di parole

1. Vocabolario con Stemming:
   - Dimensione Vocabolario: **10531**
   - Lunghezza massima vettori di training: **38**
   - Lunghezza massima vettori di test: **33**
   - Lunghezza minima vettori di training: **1**
   - Lunghezza minima vettori di test: **2**
   - Esempio vocabolario con parole **che hanno subito stemming**:


```
'char_level': False,
'document_count': 5471,
'filters': '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
'index_docs': '{"4795": 1, "688": 18, "356": 36, "5": 820, "574": 21, "3": 1039, "53": 127, "32": 176, "1384": 8, "188": 59, "732": 16, "252":
'index_word': '{"1": "rom", "2": "immigr", "3": "migrant", "4": "l", "5": "ital", "6": "islam", "7": "italian", "8": "terror", "9": "camp", "1
'lower': True,
'num_words': None,
'oov_token': None,
'split': ' ',
'word_counts': '{"sbagl": 18, "dov": 60, "direcon": 1, "forner": 8, "ital": 875, "pien": 36, "anzian": 22, "senz": 190, "pension": 49, "migran
'word_docs': '{"direcon": 1, "sbagl": 18, "pien": 36, "ital": 820, "anzian": 21, "migrant": 1039, "pag": 127, "senz": 176, "forner": 8, "dov":
'word_index': '{"rom": 1, "immigr": 2, "migrant": 3, "l": 4, "ital": 5, "islam": 6, "italian": 7, "terror": 8, "camp": 9, "sol": 10, "cas": 11
```


2. Vocabolario No Stemming:
   - Dimensione Vocabolario: **16370**
   - Lunghezza massima vettori di training: **38**
   - Lunghezza massima vettori di test: **31**
   - Lunghezza minima vettori di training: **1**
   - Lunghezza minima vettori di test: **2**
   - Esempio vocabolario con parole **non hanno subito stemming**:


```
'char_level': False,
'document_count': 5471,
'filters': '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
'index_docs': '{"6749": 1, "307": 29, "910": 12, "659": 16, "540": 18, "2": 1008, "24": 176, "1384": 8, "613": 16, "1214": 9, "1383": 8, "4": 752,
'index_word': '{"1": "rom", "2": "migranti", "3": "l", "4": "italia", "5": "immigrati", "6": "roma", "7": "italiani", "8": "stranieri", "9": "solo
'lower': True,
'num_words': None,
'oov_token': None,
'split': ' ',
'word_counts': '{"sbagliato": 8, "doveva": 9, "direcon": 1, "fornero": 8, "italia": 782, "piena": 12, "anziani": 16, "senza": 190, "pensione": 19,
'word_docs': '{"direcon": 1, "pagano": 29, "piena": 12, "anziani": 16, "pensione": 18, "migranti": 1008, "senza": 176, "fornero": 8, "contributi":
'word_index': '{"rom": 1, "migranti": 2, "l": 3, "italia": 4, "immigrati": 5, "roma": 6, "italiani": 7, "stranieri": 8, "solo": 9, "immigrazione":
```


3. Esempio Vettore:
   `sbagliato doveva direcon fornero italia piena anziani senza pensione migranti pagano contributi`
4. Esempio vettore con e senza **padding**:

```
[1383, 1214, 6749, 1384, 4, 910, 659, 24, 540, 2, 307, 613]
```

```
[1383 1214 6749 1384    4  910  659   24  540    2  307  613    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0]
```

Modello ANN

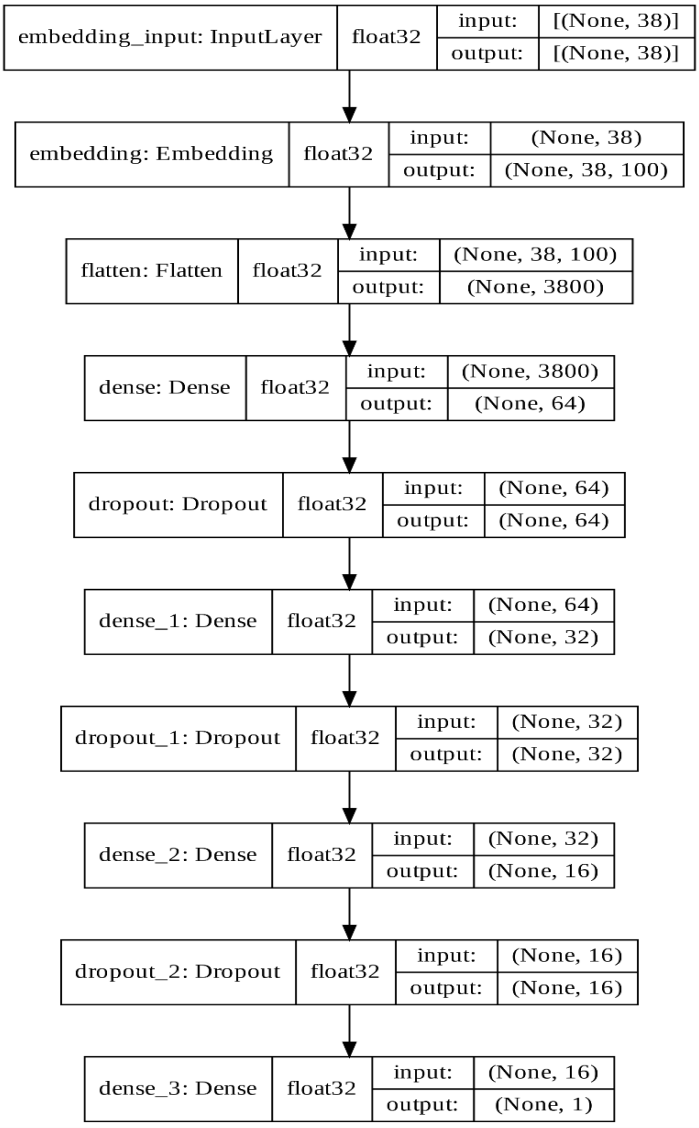
Il seguente modello è derivato dalle varie prove effettuate ed è risultato il migliore in termini delle varie metriche di valutazione che approfondiremo successivamente.

Il modello prende in input il dataset con applicato lo stemming e il vocabolario creato in precedenza.

Dato che in questo caso non si utilizza un vocabolario pre-addestrato, nel layer di Embedding() andremo a mettere il parametro "trainable" uguale a "True" così da poter allenare i pesi. Inoltre la dimensione dell'embedding è stata fissata a "100" ciò che ogni parola ha 100 features.

COMPOSIZIONE

Le caratteristiche del modello sono le seguenti:



- **Layer di Input**
- **Layer di Embedding:** Questo layer prenderà come parametri:
    - Input_dim = vocab_size. Ovvero, la dimensione del vocabolario creato + 1.
    - Output_dim = embedding_dim. Ogni parola sarà rappresentata da un vettore di questa lunghezza (nel nostro caso 100).
    - Input_lenght = maxlen. Dimensione massima che prenderà in input. Ovvero la massima lunghezza di una frase.

tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length = maxlen)

- **Layer Flatten:** Permette di compattare la matrice 2D in un vettore 1D.

tf.keras.layers.Flatten()

- **Layer Dense:** È il normale strato di rete neurale profondamente connesso. È il livello più comune e utilizzato di frequente. Esegue l'operazione seguente sull'input e restituisce l'output.

Nel nostro modello abbiamo 3 layer dense che hanno, a cascata, i seguenti neuroni: 64, 32, 16. La funzione di attivazione per tutti e 3 i layer è la "*relu*", la funzione in questione prende la parte positiva del suo argomento. Il parametro "*bias_initializer*" ed è impostato a "0.1" questo perché dato che usiamo la relu è consigliato da keras per evitare la morte dei neuroni

tf.keras.layers.Dense(units, activation = "relu", bias_initializer=tf.keras.initializers.Constant(0.1)))

- **Layer DropOut:** Il livello Dropout imposta casualmente le unità di input su "0" con una frequenza di frequenza ad ogni passaggio durante il tempo di allenamento, il che aiuta a prevenire l'overfitting.

tf.keras.layers.Dropout(0.3)

- **Layer OutPut:** Ha come parametri 1 neurone, come funzione di attivazione "sigmoid" tale funzione restituisce i valori tra 0 e 1. L'ultimo valore è "bias_inizializer" impsotato a "zeros".

tf.keras.layers.Dense(1, activation = "sigmoid", bias_initializer="zeros")

Il modello è stato configurato nella seguente maniera:

- Come ottimizzazione è stata usata "**adam**", quest'ottimizzazione è un metodo stocastico di discesa del gradiente e implementa l'algoritmo di Adam.
- Come loss function o funzione obiettivo abbiamo usato la "**binary_crossentropy**" che confronta ciascuna delle probabilità predette con l'output effettivo della classe che può essere 0 o 1. Quindi calcola il punteggio che penalizza le probabilità in base alla distanza dal valore atteso. Ciò significa quanto vicino o lontano dal valore effettivo.
- Come metrica da valutare dal modello durante l'addestramento e il test l' "**Accuracy**".

```
model_dense.compile(optimizer = "adam",loss ="binary_crossentropy",metrics =["accuracy"]
```

I parametri risultanti del modello sono:

- **Total params:** 1,273,189
- **Trainable params:** 1,273,189
- **Non-trainable params:** 0

ALLENAMENTO

Come allenamento abbiamo eseguito "10" epoche con un **batch_size** a "1024". Il parametro batch_size permette di definire il numero di campioni che verranno propagati attraverso la rete.

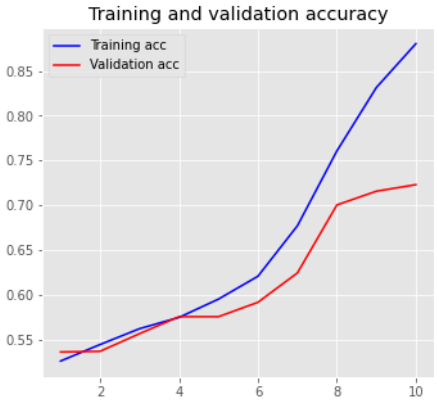
```
history = model_dense.fit(np.array(X_train), np.array(y_train), epochs=10,verbos
e=True, validation_data=(np.array(X_test), np.array(y_test)),batch_size=1024)
```

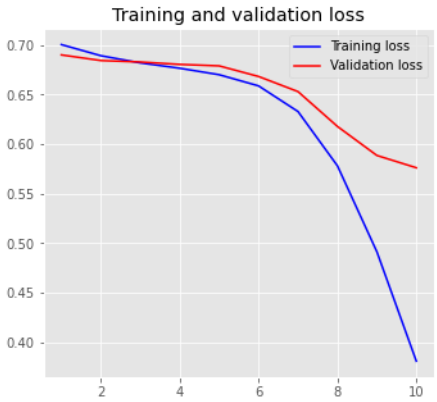
```
Epoch 1/10
6/6 [==============================] - 1s 66ms/step - loss: 0.7005 - accuracy: 0.5257 - val_loss: 0.6901 - val_accuracy: 0.5358
Epoch 2/10
6/6 [==============================] - 0s 37ms/step - loss: 0.6892 - accuracy: 0.5441 - val_loss: 0.6844 - val_accuracy: 0.5365
Epoch 3/10
6/6 [==============================] - 0s 37ms/step - loss: 0.6820 - accuracy: 0.5619 - val_loss: 0.6829 - val_accuracy: 0.5563
Epoch 4/10
6/6 [==============================] - 0s 37ms/step - loss: 0.6766 - accuracy: 0.5745 - val_loss: 0.6805 - val_accuracy: 0.5753
Epoch 5/10
6/6 [==============================] - 0s 39ms/step - loss: 0.6701 - accuracy: 0.5950 - val_loss: 0.6790 - val_accuracy: 0.5753
Epoch 6/10
6/6 [==============================] - 0s 38ms/step - loss: 0.6588 - accuracy: 0.6205 - val_loss: 0.6683 - val_accuracy: 0.5914
Epoch 7/10
6/6 [==============================] - 0s 38ms/step - loss: 0.6329 - accuracy: 0.6770 - val_loss: 0.6530 - val_accuracy: 0.6243
Epoch 8/10
6/6 [==============================] - 0s 38ms/step - loss: 0.5781 - accuracy: 0.7607 - val_loss: 0.6179 - val_accuracy: 0.7003
Epoch 9/10
6/6 [==============================] - 0s 36ms/step - loss: 0.4914 - accuracy: 0.8317 - val_loss: 0.5886 - val_accuracy: 0.7156
Epoch 10/10
6/6 [==============================] - 0s 38ms/step - loss: 0.3811 - accuracy: 0.8806 - val_loss: 0.5762 - val_accuracy: 0.7230
```

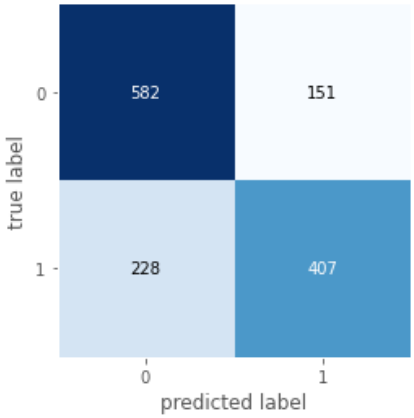
RISULTATI OTTENUTI

Il modello ha prodotto i seguenti risultati:

Durante la fase di training il modello ha raggiunto un'accuracy dell'88% ed una loss dell'38%. Invece durante la validazione abbiamo ottenuto, alla 10° epoca, un'accuracy dell'72,3% ed una loss del 57,62%. I seguenti risultati sembrano buoni ma occorre comunque fare altre valutazioni.







Dalla matrice di confusione si evidenzia che il test è stato eseguito prendendo 635 record di classe 1 (stereotipi) e 733 di classe 0 (non stereotipi). Come si vede in figuri si riesce a classificare bene la classe 1 rispetto alla classe 0.

Nella seguente immagine possiamo notare le previsioni corrette e gli errori commessi inoltre, possiamo vedere le altre metriche utilizzate per valutare il modello derivanti dalla matrice di confusione.

```
Accuratezza:  0.722953216374269

Dimensione training set:  5471
Dimensione test set:  1368

Previsioni Corrette:  989
Errori Commessi:  379
Accuratezza:  72.3 %

              precision    recall  f1-score   support

           0       0.72      0.79      0.75       733
           1       0.73      0.64      0.68       635

    accuracy                           0.72      1368
   macro avg       0.72      0.72      0.72      1368
weighted avg       0.72      0.72      0.72      1368
```

- **Precision:** Questa metrica permette di valutare la precisione con il quale il modello è capace di riconoscere i record di una determinata classe. Nel nostro caso vediamo che la classe 0 e la classe 1 hanno precisione simile (rispettivamente 72% e 73%) quindi possiamo dire che il modello è in grado di distinguere abbastanza bene le due classi. Purtroppo questa metrica tiene conto tiene conto dei falsi positivi ovvero tutti quei record che sono classificati come appartenenti alla classe 1 quando in realtà appartengono alla classe 0.
- **Recall:** Usiamo questa metrica per risolvere il problema della precision poiché invece di tener conto dei falsi positivi essa tiene conto dei falsi negativi. I falsi negativi sono i record che vengono etichettati come appartenenti alla classe 0 ma invece dovrebbero essere etichettati come appartenenti alla classe 1. Come possiamo notare la classe 0 viene predetta correttamente dal modello ma per quanto riguarda la classe 1 il modello inizia ad avere delle debolezze poiché ha molti falsi negativi.
- **F1-score:** Questa metrica sintetizza i risultati di **precision** e recall.
- **Support:** Ci permette di vedere il numero di campioni utilizzati per ogni classe.

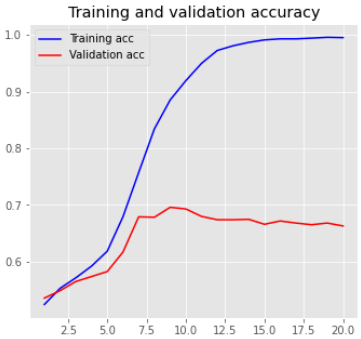
PROVE EFFETTUATE

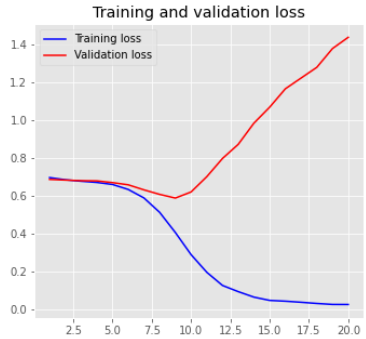
Di seguito si mostreranno le varie prove effettuate per arrivare ai risultati ottenuti in precedenza che sono risultati i migliori nel nostro caso.
Le varie sperimentazioni sono state fatte su vari parametri che sono: **Numero di Epoche**, **batch_size**, modifiche al valore del **layer DropOut** e regolazione della costante "***bias_initializer***".
Queste sperimentazioni ci hanno permesso di vedere i vari problemi del modello, come overfitting e numero di errori commessi da esso, e andarli a risolvere così da poter stabilire la miglior configurazione possibile.

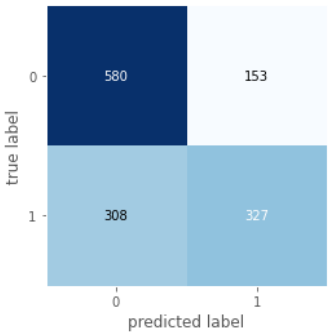
Di seguito si vedrà un'analisi dettagliata:

- OVERFITTING NEL MODELLO

```
history = model_dense.fit(np.array(X_train), np.array(y_train), epochs=20,verbose=True, validati
on_data=(np.array(X_test), np.array(y_test)),batch_size=1024)
```

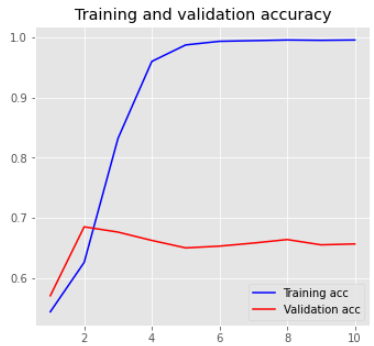


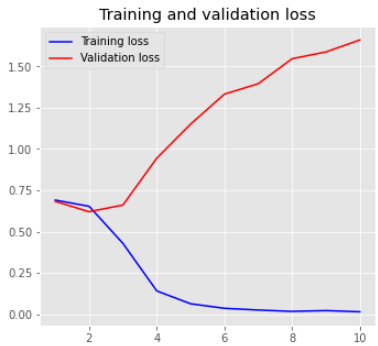


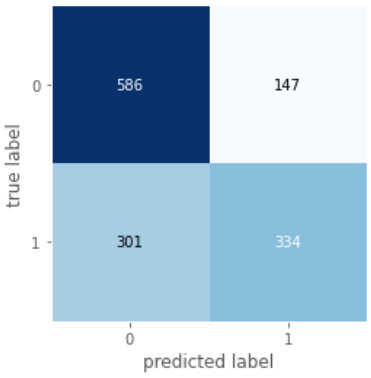


- BATCH_SIZE TROPPO BASSO

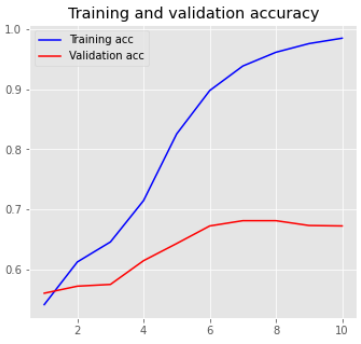
```
history = model_dense.fit(np.array(X_train), np.array(y_train), epochs=10,verbose=True, validati
on_data=(np.array(X_test), np.array(y_test)),batch_size=100)
```

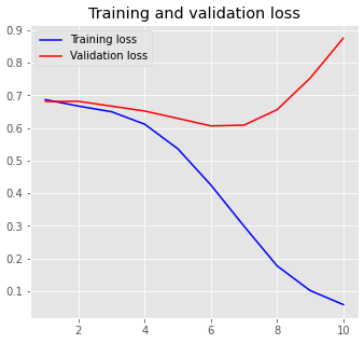


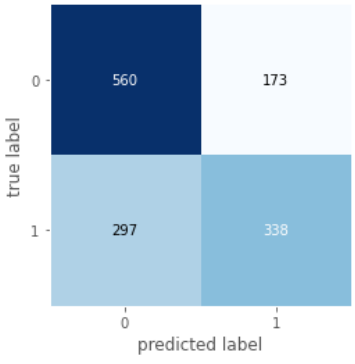




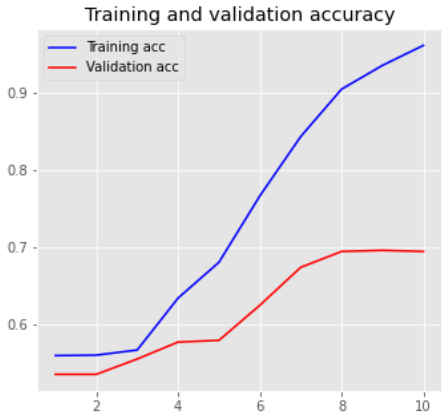
- DROPOUT 0.1

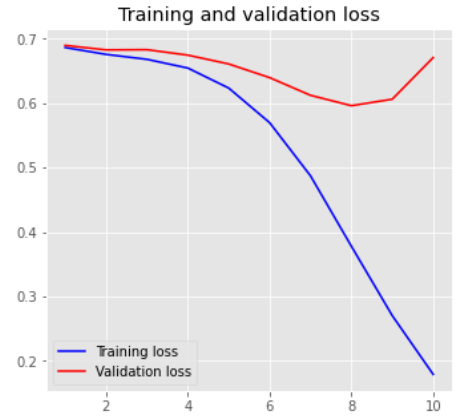


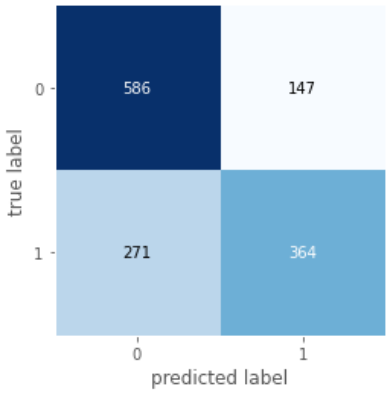


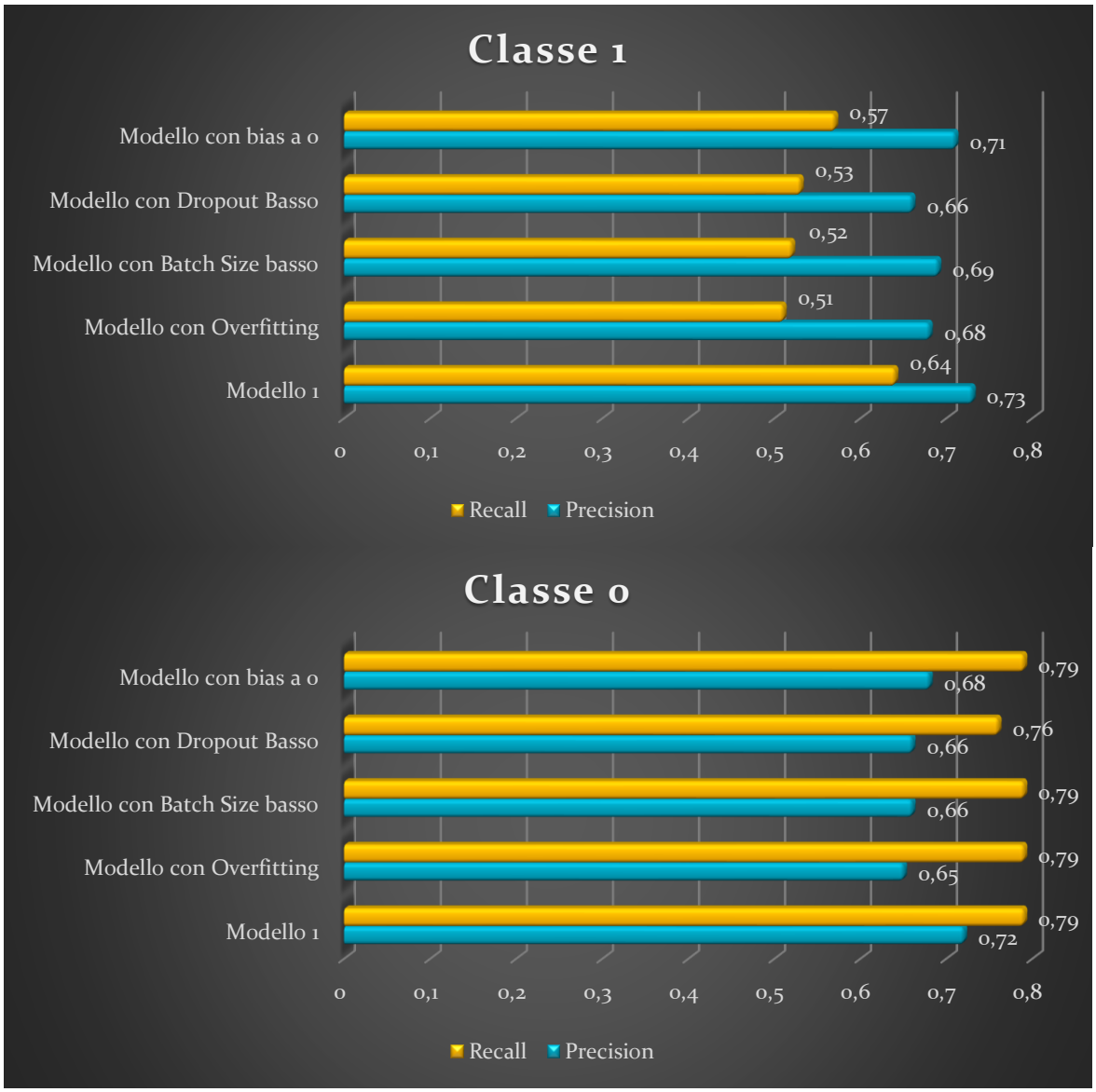


- INIZIALIZZAZIONE DEL BIAS A 0
  Inizializza i bias con un piccolo valore positivo come 0.1. Poiché stiamo usando i neuroni ReLU, è anche buona pratica inizializzarli con un bias iniziale leggermente positivo per evitare "neuroni morti".









DOVE SBAGLIA IL MODELLO

Dopo aver analizzato i vari risultati ed aver notato che il modello faceva degli errori quindi abbiamo cercato un modo per poter controllare con quali frasi sbagliava il nostro modello.

La procedura adottata è stata quella di analizzare i record test classificati in modo errato. Ovvero, tramite una parte di codice, controllare quando la classe predetta è diversa dalla classe effettiva del record.

```
for i in range(0, len(Test_label)):

  x_sample=[Test_record[i]]
  x_sample_token = tokenizer.texts_to_sequences(x_sample)
  x_sample_token = pad_sequences(x_sample_token, padding = "post", maxlen = maxlen)
  predizione = model_dense.predict_classes(x_sample_token)
  if predizione[0][0] != Test_label[i]:
      print("Il modello ha predetto "+str(predizione[0]))
      print("Il record era classe "+str(Test_label[i]))
      print(Test_record[i])

```

I risultati ottenuti sono questi:

```
Il modello ha predetto 0
Il record era classe 1
nomad rinchius follon fatt grav straziant ved ladr rom dietr grat pov donn derub
```

```
Il modello ha predetto 1
Il record era classe 0
rom arrest rapin nomad latit
```

Come si vede in questo esempio dato che abbiamo la parola “rom” sta sia in record di classe 0 che in uno di classe 1 il modello non riesce ad avere una classificazione deterministica della parola.

Modello ANN con FastText (vettori di parole pre-addestrati)

Il seguente modello viene svolto applicando un vocabolario con vettori pre-allenati. In particolare, abbiamo utilizzato FastText scaricando il vocabolario da 300 features ma ridimensionandolo così da ottenere un dizionario con 100 features.

Di seguito è possibile vedere un esempio di parola con 100 features.

```
1 ft.get_word_vector('gatto')
```

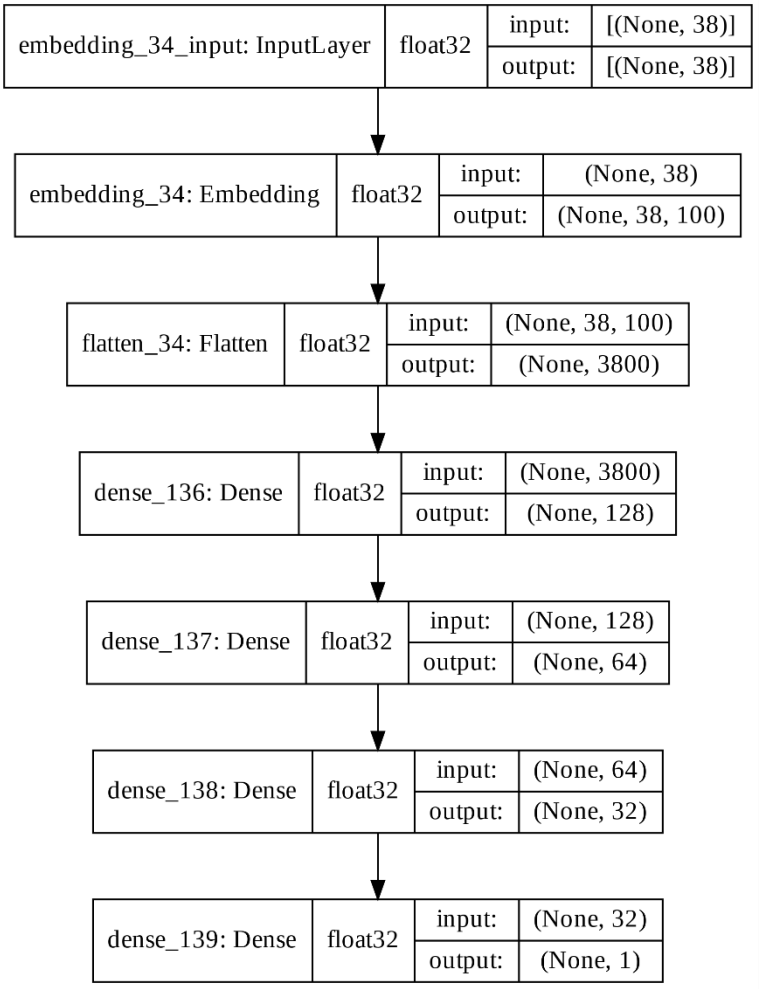
```
array([ 0.08438098,  0.1402936 , -0.06689572,  0.19108896, -0.10038342,
        0.10648277, -0.07903218,  0.25857222,  0.19003722, -0.06537314,
       -0.08662468, -0.02853714, -0.12075889, -0.02077787,  0.0504254 ,
       -0.06068609, -0.04615176, -0.14948761, -0.11595051,  0.10119529,
       -0.08293849, -0.01438684,  0.02754185,  0.01721069, -0.01762934,
        0.08135373, -0.08086737, -0.03874275,  0.00728715, -0.07501822,
        0.01854763,  0.01610843,  0.01007087,  0.03030904,  0.01137639,
        0.05830673, -0.22467397, -0.07957438,  0.17677724, -0.19668782,
       -0.04417606, -0.07363875,  0.05294646, -0.03278863, -0.00569305,
       -0.05957241, -0.07237665, -0.0258616 , -0.07777632, -0.03587773,
       -0.06856329, -0.12835196, -0.09072002,  0.11221576, -0.02786185,
        0.02426682, -0.07915666,  0.07751189,  0.05701655, -0.05395695,
       -0.00047861, -0.21213011,  0.04114397,  0.03428408,  0.00896816,
        0.00605063, -0.03679192, -0.02208775, -0.00389313, -0.04675273,
        0.06354631,  0.07789837, -0.09050789, -0.07066286, -0.00407424,
        0.00354244, -0.00112351, -0.04535507,  0.07013391,  0.0430989 ,
       -0.03602118, -0.06520332,  0.00367676, -0.05652206, -0.02249732,
       -0.04093431,  0.01136662, -0.06171096,  0.00791843,  0.08021972,
        0.05222661,  0.00638712, -0.00622971, -0.00831309, -0.03808447,
        0.02460582,  0.06869251, -0.01006094, -0.02312582,  0.02805201],
      dtype=float32)
```

Per poter lavorare con i pesi pre-addestrati è necessario creare una matrice dei pesi. Essa, viene prima inizializzata a "0" e successivamente riempita aggiornando i pesi con le parole del vocabolario con cui stiamo lavorando.

```
count1 = 0
count2 = 0
for word, index in tokenizer.word_index.items():
  if index > vocab_size-1:
    break
  embedding_vector = ft.get_word_vector(word)
  if embedding_vector is not None:
    embeddings_matrix[index] = embedding_vector
    count1 += 1
  else:
    count2 += 1
print("Parole del nostro vocabolario presenti nella matrice: "+ str(count1))
print("Parole del nostro vocabolario non presenti nella matrice: "+ str(count2))
```

È importante far notare che nelle varie sperimentazioni che abbiamo eseguito abbiamo notato che il modello che utilizza FastText non deve aver effettuato l'operazione di stemming nella fase di pulizia del testo.

COMPOSIZIONE



Le caratteristiche del modello sono le seguenti:

- **Layer di Input**
- **Layer di Embedding:** Questo layer prenderà come parametri:
    - Input_dim = vocab_size. Ovvero, la dimensione del vocabolario creato + 1.
    - Output_dim = embedding_dim. Ogni parola sarà rappresentata da un vettore di questa lunghezza (nel nostro caso 100).
    - weights = [embbeddings_matrix]. Si useranno i pesi inseriti precedentemente nella matrice.
    - Input_lenght = maxlen. Dimensione massima che prenderà in input. Ovvero la massima lunghezza di una frase.
    - trainable = False. Viene impostato a false per non riaddestrare i presi dato che i vettori sono pre-allenati.

  tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=maxlen, weights = [embeddings_matrix], trainable = False))

- **Layer Flatten:** Permette di compattare la matrice 2D in un vettore 1D.

  tf.keras.layers.Flatten()

- **Layer Dense:** In questo modello abbiamo 3 layer dense che hanno, a cascata, i seguenti neuroni: 128, 64, 32.
  La funzione di attivazione per tutti e 3 i layer è la "*relu*", la funzione in questione prende la parte positiva del suo argomento. Il parametro "*bias_initializer*" ed è impostato a "zeros".

  tf.keras.layers.Dense(units, activation = "relu", bias_initializer="zeros"))

- **Layer OutPut:** Ha come parametri 1 neurone, come funzione di attivazione "sigmoid" tale funzione restituisce i valori tra 0 e 1. L'ultimo valore è "bias_inizializer" impsotato a "zeros".

  tf.keras.layers.Dense(1, activation = "sigmoid", bias_initializer="zeros")

Il modello è stato configurato nella seguente maniera:

- Come ottimizzazione è stata usata "**adam**", quest'ottimizzazione è un metodo stocastico di discesa del gradiente e implementa l'algoritmo di Adam.
- Come loss function o funzione obiettivo abbiamo usato la "**binary_crossentropy**" che confronta ciascuna delle probabilità predette con l'output effettivo della classe che può essere 0 o 1. Quindi calcola il punteggio che penalizza le probabilità in base alla distanza dal valore atteso. Ciò significa quanto vicino o lontano dal valore effettivo.
- Come metrica da valutare dal modello durante l'addestramento e il test l' "**Accuracy**".

```
model_dense.compile(optimizer = "adam",loss ="binary_crossentropy",metrics =["accuracy"]
```

I parametri risultanti del modello sono:

Total params: 2,133,897
Trainable params: 496,897
Non-trainable params: 1,637,000

ALLENAMENTO

Come allenamento abbiamo eseguito "4" epoche con un **batch_size** a "128". Il parametro batch_size permette di definire il numero di campioni che verranno propagati attraverso la rete.

```
history = model_prova.fit(np.array(X_train), np.array(y_train), epochs=4,verbose
=True, validation_data=(np.array(X_test), np.array(y_test)),batch_size=128)
```

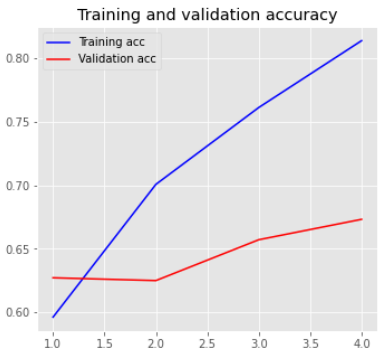
```
Epoch 1/4
43/43 [==============================] - 1s 10ms/step - loss: 0.6622 - accuracy: 0.5962 - val_loss: 0.6386 - val_accuracy: 0.6272
Epoch 2/4
43/43 [==============================] - 0s 7ms/step - loss: 0.5710 - accuracy: 0.7008 - val_loss: 0.6623 - val_accuracy: 0.6250
Epoch 3/4
43/43 [==============================] - 0s 7ms/step - loss: 0.4884 - accuracy: 0.7613 - val_loss: 0.6715 - val_accuracy: 0.6572
Epoch 4/4
43/43 [==============================] - 0s 7ms/step - loss: 0.4068 - accuracy: 0.8139 - val_loss: 0.6803 - val_accuracy: 0.6732
```

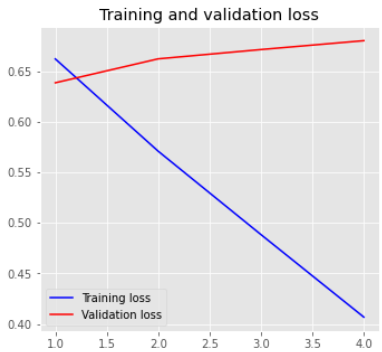
RISULTATI OTTENUTI

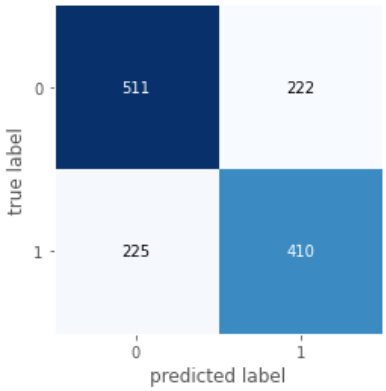
I risultati ottenuti tramite il modello applicato al FastText sono i seguenti:

Il modello ha prodotto i seguenti risultati:

Durante la fase di training il modello ha raggiunto un' accuracy dell'81% ed una loss dell' 40%. Invece durante la validazione abbiamo ottenuto, alla 10° epoca, un' accuracy dell' 67,3% ed una loss del 68%.







Dalla matrice di confusione si evidenzia che il test è stato eseguito prendendo 635 record di classe 1 (stereotipi) e 733 di classe 0 (non stereotipi). Come si vede in figura si riesce a classificare bene la classe 1 rispetto alla classe 0.

Nella seguente immagine possiamo notare le previsioni corrette e gli errori commessi. Inoltre, possiamo vedere le altre metriche utilizzate per valutare il modello derivanti dalla matrice di confusione.

- **Precision:** In questo modello vediamo che questa metrica è più bassa rispetto al modello precedente.
  Per la classe 0 abbiamo 69% e per la classe 1 65%.
- **Recall:** Anche in questo modello, utilizzando questa metrica, vediamo come la classe 0 sia predetta meglio della classe 1 (70% e 65% rispettivamente).
- **F1-score:** Questa metrica sintetizza i risultati di **precision** e **recall**.
- **Support:** Il numero di campioni è lo stesso utilizzato nel modello precedente.

```
Utilizzati come training set:  5471
Utilizzati come test set:  1368

Previsioni Corrette:  921
Errori Commessi:  447
Accuratezza:  67.32 %

              precision    recall  f1-score   support

           0       0.69      0.70      0.70       733
           1       0.65      0.65      0.65       635

    accuracy                           0.67      1368
   macro avg       0.67      0.67      0.67      1368
weighted avg       0.67      0.67      0.67      1368
```

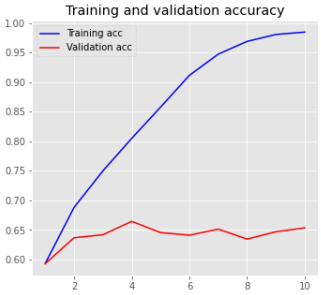
PROVE EFFETTUATE

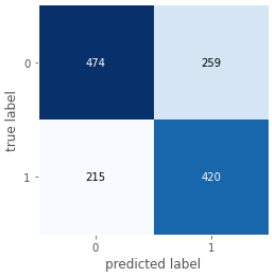
Di seguito si mostreranno le varie prove effettuate per arrivare ai risultati ottenuti in precedenza che sono risultati i migliori nel nostro caso.
Le varie sperimentazioni sono state fatte su vari parametri che sono: **Numero di Epoche**, **batch_size**, modifiche al valore del **layer DropOut** e regolazione della costante "***bias_initializer***". Queste sperimentazioni ci hanno permesso di vedere i vari problemi del modello, come overfitting e numero di errori commessi da esso, e andarli a risolvere così da poter stabilire la miglior configurazione possibile.

Di seguito si vedrà un'analisi dettagliata:

- OVERFITTING NEL MODELLO

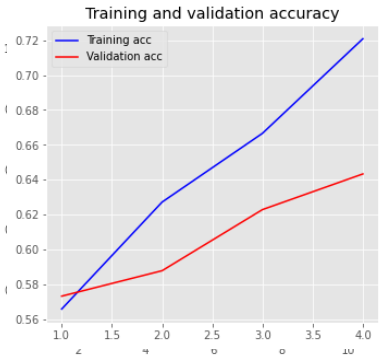
```
history = model_prova.fit(np.array(X_train), np.array(y_train), epochs=10,verbos
e=True, validation_data=(np.array(X_test), np.array(y_test)),batch_size=128)
```

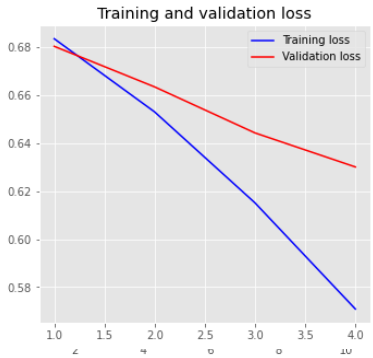


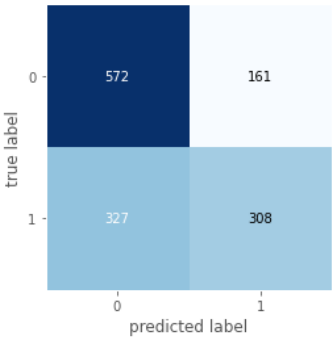


- BATCH_SIZE ALTO

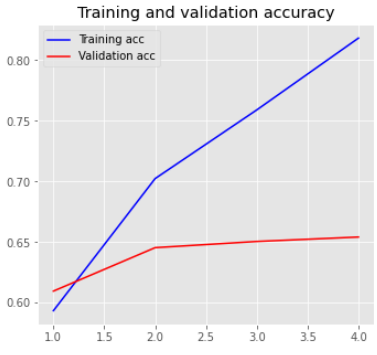
```
history = model_prova.fit(np.array(X_train), np.array(y_train), epochs=4,verbose
=True, validation_data=(np.array(X_test), np.array(y_test)),batch_size=1024)
```

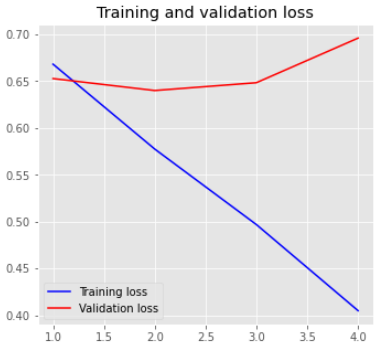


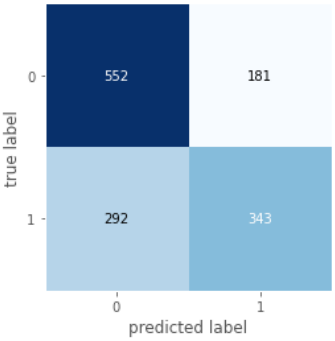




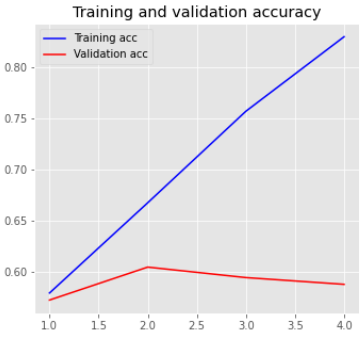
- PRESENZA DEL DROPOUT

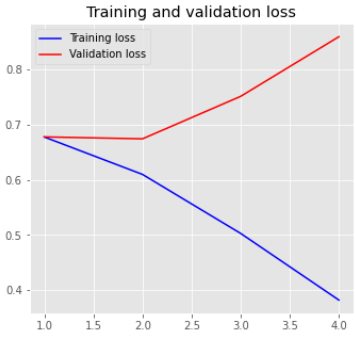


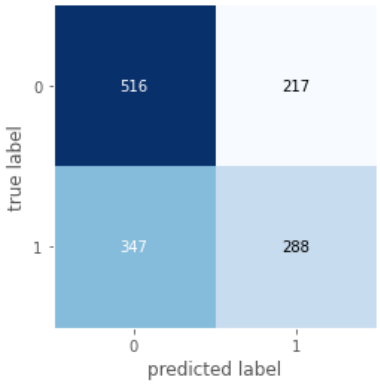


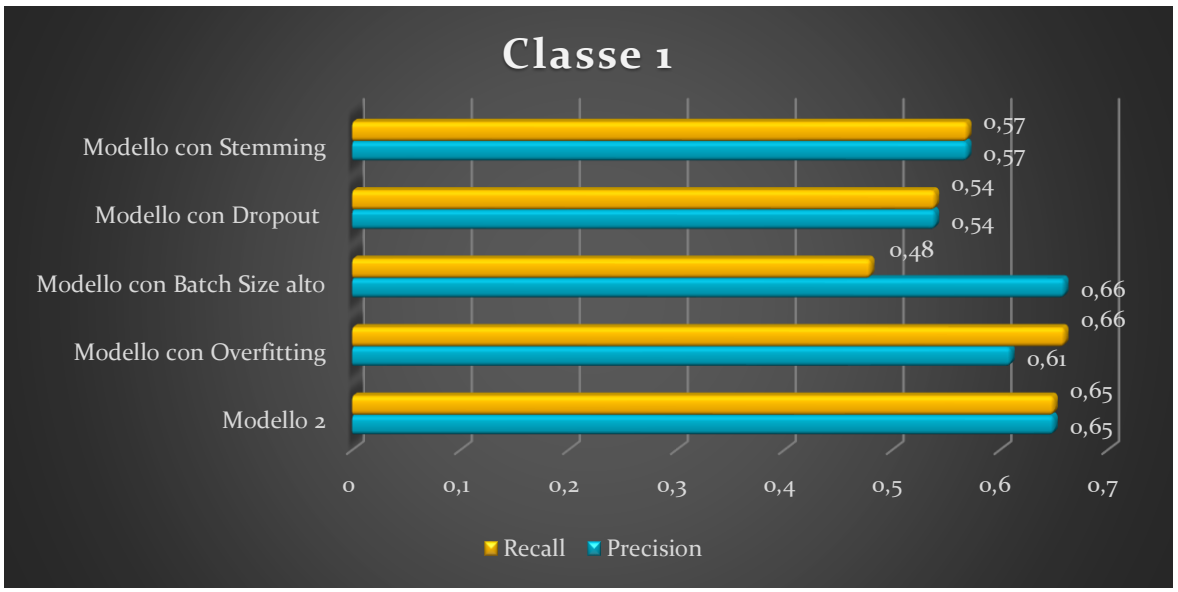


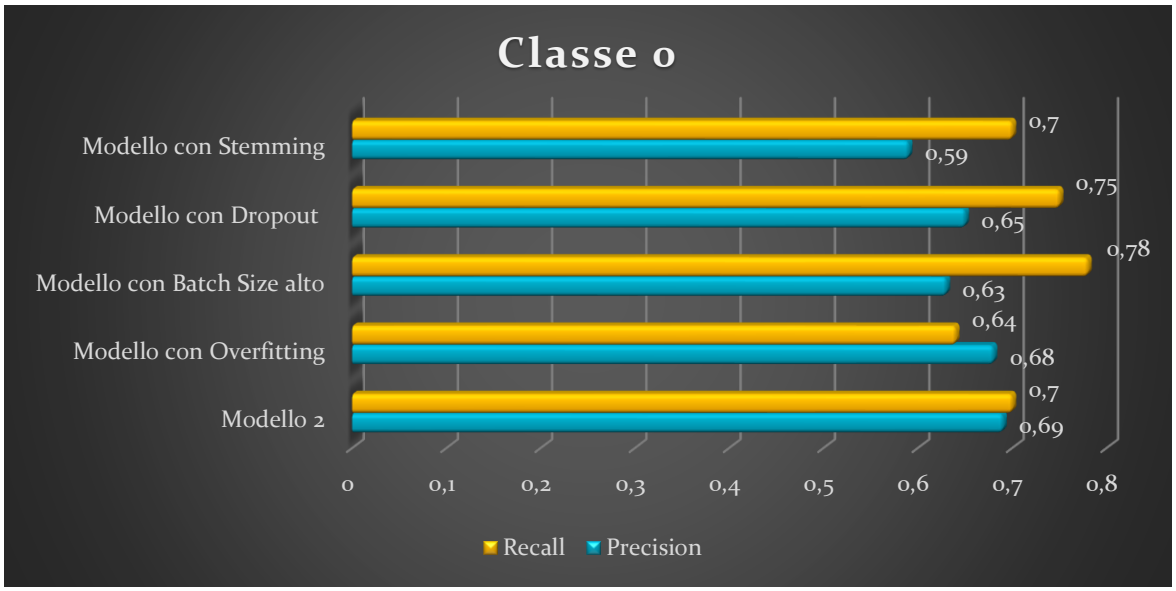
- CON STEMMING











Confronto delle performance dei due modelli

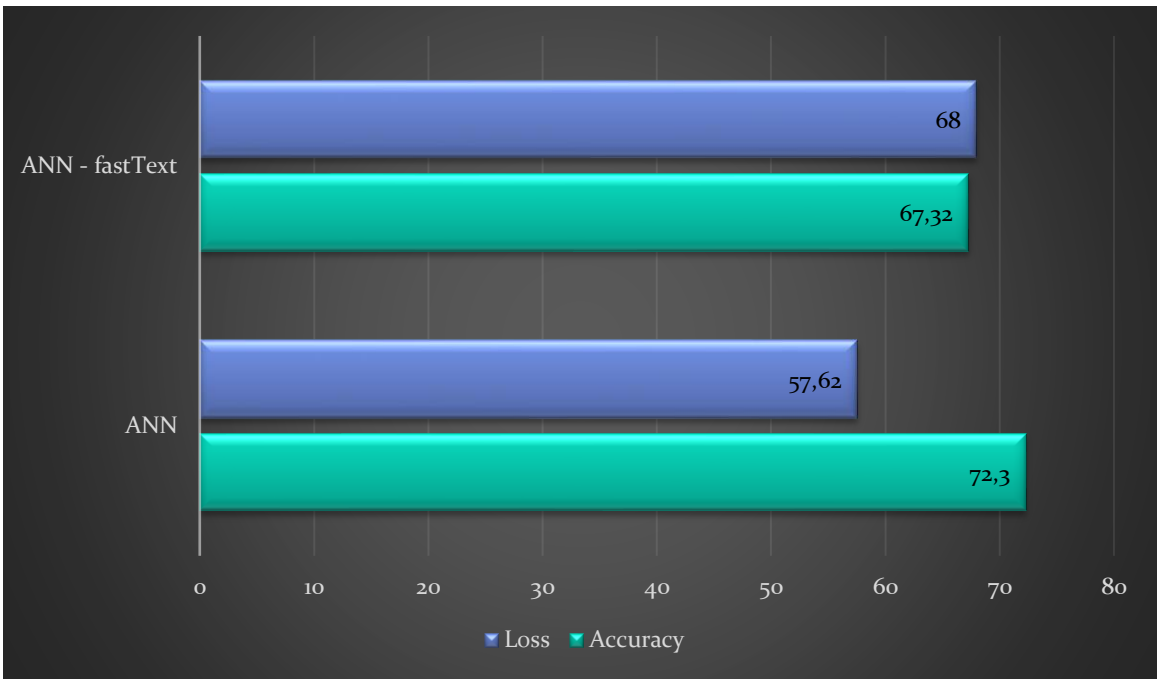
Di seguito confrontiamo il comportamento dei due modelli creati per valutarne il migliore.

Innanzitutto per quanto riguarda l'accuratezza e la loss, possiamo notare che il modello che produce la migliore soluzione al problema è quello che utilizza il vocabolario senza i vettori pre-allenati di fastText.

La differenza sostanziali sta anche nel fatto che il modello con fastText non ha lo stemming e per questo potrebbe incorrere in errori di comprensione del testo. Come abbiamo visto nelle prove utilizzare fastText con stemming non è una buona soluzione. Per questo abbiamo ipotizzato che i vettori delle parole stemmate sono meno precisi e più ambigui delle altre forme.

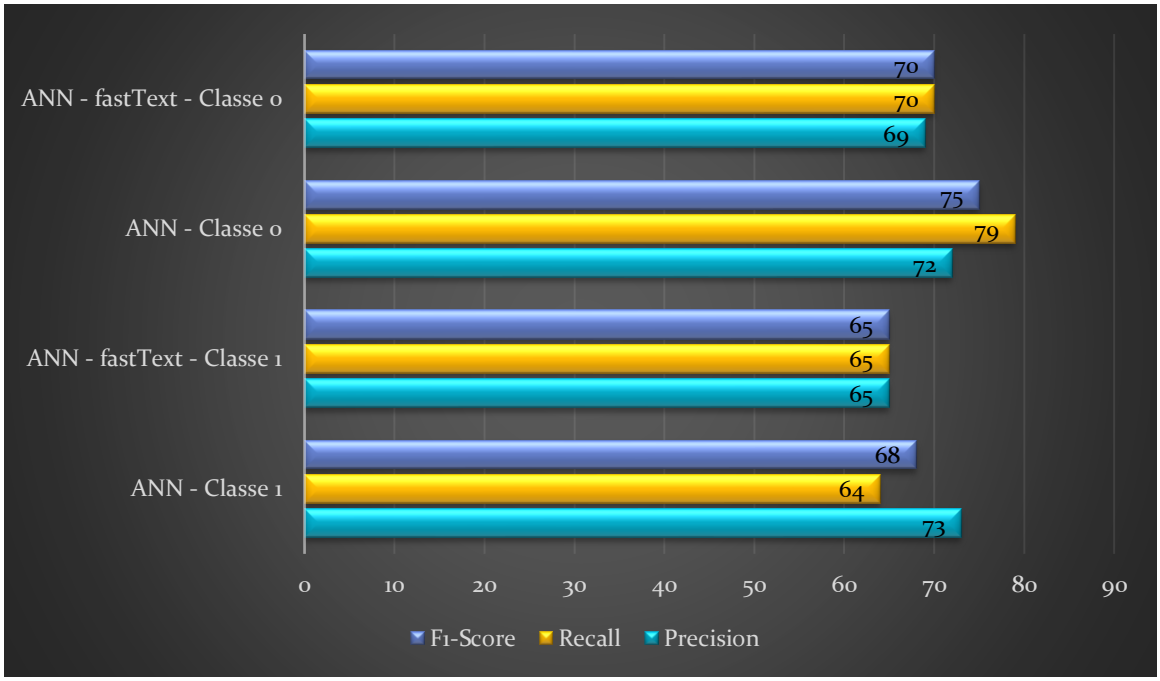
Un'ulteriore differenza è che, pur provando diverse configurazioni, il modello con fastText non permette di essere migliorato.

Per quanto concerne invece l'altro modello possiamo dire che sotto queste metriche è quello che ci dà i risultati migliori.



Per quanto riguarda le altre metriche di valutazione si possono trarre le seguenti conclusioni:

- Il modello ANN con fastText risulta:
  - Peggiore nel predire la Classe 0
  - Mentre nel predire la Classe 1 si ha una recall migliore dell'altro, quindi anche se di poco produce meno falsi negativi
- Il modello ANN risulta:
  - In generale migliore nel predire la Classe 1 e 0



Conclusioni

Le due soluzioni sviluppate attraverso questo progetto riescono a risolvere il task. Ci sono comunque dei possibili miglioramenti che non possono essere applicati ai modelli proposti. Questo perché tali modelli non riescono a cogliere il contesto di una frase poiché anche se le parole nelle frasi venissero mischiate il risultato finale sarà essenzialmente lo stesso.

Bisognerebbe quindi trovare una rete che ci permetta di cogliere la posizione della parola nella frase per estrapolare al meglio il senso

Un modello che si sposerebbe molto bene per risolvere questa task potrebbe essere le Reti Neurali Ricorrenti. Le reti ricorrenti prevedono, in sostanza, collegamenti all'indietro o verso lo stesso livello. Questa caratteristica rende questo tipo di rete neurale molto interessante, perché il concetto di ricorrenza introduce intrinsecamente il concetto di memoria di una rete. In una rete RNN, infatti, l'output di un neurone può influenzare sé stesso, in uno step temporale successivo o può influenzare neuroni della catena precedente che a loro volta interferiranno con il comportamento del neurone su cui si chiude il loop.

Sitografia

1. https://colab.research.google.com/notebooks/intro.ipynb#recent=true
2. https://www.python.org/
3. https://www.tensorflow.org/
4. https://it.wikipedia.org/wiki/TensorFlow
5. https://keras.io/
6. https://it.wikipedia.org/wiki/Keras
7. https://en.wikipedia.org/wiki/Scikit-learn
8. https://scikit-learn.org/stable/
9. https://it.wikipedia.org/wiki/Natural_Language_Toolkit
10. https://www.nltk.org/
11. https://fasttext.cc/
12. https://fasttext.cc/docs/en/crawl-vectors.html

GitHub Progetto: https://github.com/proliuk/TASK-B-EVALITA-HASPEED-2020.git