1. INTRODUZIONE

Al giorno d'oggi i dati rivestono una grandissima importanza nelle attività umane; mai come oggi l'acquisizione e l'analisi dei dati sono state così redditizie, in tutti gli ambiti. In particolare, un campo in cui l'analisi dei dati è sempre stata praticamente indispensabile è la politica. Un buon politico per definizione deve conoscere i bisogni dei suoi elettori, e possibilmente di tutti i cittadini, per cercare di soddisfarli; il miglior modo per farlo è proprio raccogliere dati su di essi. In altre parole, un buon politico deve sapere cosa vuole il suo elettore "tipo", quali sono le sue idee e cercare di assecondarle, per fare i suoi interessi.

Il problema affrontato in questo progetto è correlato a ciò: il politico deve innanzitutto identificare il suo elettore tipo, ed esaminare le opinioni espresse da esso, per comprendere i suoi bisogni.

Nell'epoca "analogica", l'unico modo per raccogliere dati sulle preferenze politiche era quello dei sondaggi politici: veniva scelto un campione, più eterogeneo possibile, di persone (circa un migliaio) per intervistarli personalmente e ottenere dei dati con alta significatività (solitamente circa 95%).

Con l'avvento dei social media, il cittadino ha ottenuto uno spazio infinitamente più grande per poter esprimere le proprie opinioni, i propri bisogni e le proprie rimostranze. Potenzialmente, tutto il mondo può visualizzare ciò che esprime, sottoforma di testo, audio o video. In particolare, lo spazio preferito dal cittadino occidentale per esprimere opinioni politiche è il social media "X" (ex Twitter), che permette di condividere globalmente testi di breve lunghezza (fino a 280 caratteri); un politico (o un partito) che vuole saperne di più su cosa pensa la gente farebbe bene ad acquisire ed estrapolare da X. Rispetto ai sondaggi realizzati su campioni di poche migliaia di persone, qui il campione è immenso: miliardi di tweet scritti da milioni di persone.

Nello specifico, il nostro scopo è quello di raccogliere un dataset di tweet condivisi sulla piattaforma nel periodo delle elezioni statunitensi del 2020 e, in base sia al testo che alla descrizione, riuscire a dedurre il partito che l'utente in questione voterà alle elezioni (in caso di voto). In pratica, si riduce tutto ad un problema di text classification binaria.

Un politico, associando un partito a delle opinioni espresse dai cittadini, può far eseguire delle analisi su questi dati e capire come impostare la sua campagna elettorale, anche per "contrastare" i partiti concorrenti.

2. DATI

Il dataset utilizzato in questo progetto è disponibile al seguente link.

Precisamente, il dataset è diviso in due file .csv:

- "hashtag_donaldtrump.csv" contiene 970919 righe (cioè 970919 tweet), che contengono l'hashtag "#donaldtrump". Sono tutti i tweet in cui si parla di Donald Trump, candidato del Partito Repubblicano alle elezioni presidenziali USA del 2020 e ai tempi presidente in carica;
- "hashtag_joebiden.csv" contiene 775054 righe (cioè 775054 tweet), che contengono l'hashtag "#joebiden". Sono tutti i tweet in cui si parla di Joe Biden, candidato del Partito Democratico alle elezioni presidenziali USA del 2020, da lui vinte.

In totale, si tratta di un dataset con 1745973 righe.

Le feature del dataset sono le seguenti:

- created_at: Data e ora della creazione del tweet
- tweet_id: ID unico del tweet (chiave primaria)
- tweet: Il testo del tweet
- likes: Numero di like
- retweet_count: Numero di retweet
- source: Client usato per inviare il tweet
- user_id: ID dell'utente
- user_name: Username del profilo
- user_screen_name: Nome dell'utente associato al profilo
- user_description: Bio dell'utente
- user_join_date: Data di iscrizione a X dell'utente
- user_followers_count: Numero di followers dell'utente
- user_location: Provenienza dell'utente
- lat: Latitudine del posto inserito in user_location
- long: Longitudine del posto inserito in user_location
- city: Citt corrispondente ad user_location
- country: Nazione corrispondente ad user_location
- state: Stato (solo USA) corrispondente ad user_location
- state_code: Codice dello stato (solo USA) corrispondente ad user_location
- collected_at: Data, ora, minuti, secondi in cui il tweet è stato acquisito dall'autore del dataset

Per la nostra analisi, ci concentreremo in particolare sul testo e sulla descrizione; altre feature saranno utilizzate durante il data cleaning.

Come spiega anche l'autore del dataset, i dati sono stati acquisiti in momenti diversi tra il 15 ottobre 2020 e l'8 novembre 2020, utilizzando le API di X statuses_lookup e snsscrape, ora disponibili soltanto previo pagamento di 100$ mensili. Soprattutto per questo motivo (lo scraping di dati da X senza API è molto difficoltoso), si è deciso di non raccogliere i dati in autonomia, ma sfruttare un dataset già pronto.

2.1 DATA CLEANING

Ad una prima occhiata, si può vedere come molte delle righe del dataset non sono adatte a ciò che andremo a fare; sarà operato un data cleaning piuttosto corposo, prima, durante e dopo le operazioni comprese nella classica NLP pipeline.

Prima di tutto, vengono scartati tutti i record contenenti valori NA e i duplicati. Decidiamo successivamente di scartare tutti i tweet non scritti da utenti singoli, bensì da profili rappresentanti testate giornalistiche e organizzazioni varie: gli schieramenti politici di giornali e aziende sono già note al pubblico; è per noi interessante predirre lo schieramento politico del singolo cittadino, della persona fisica. Riusciamo a filtrare le testate giornalistiche dal fatto che nella stragrande maggioranza dei casi non utilizzano i classici client X disponibili per l'utente normale, ma utilizzano dei client personalizzati con funzionalità aggiuntive. Si scartano quindi tutti i record in cui il client utilizzato non è l'app X per iPhone/iPad/Android/Windows oppure la Web App (il sito, via browser).

In [ ]:
```python
def filter_papers(tweet_source):
    if tweet_source != "Twitter for iPhone" and tweet_source != "Twitter for Android" \
        and tweet_source != "Twitter Web App" and tweet_source != "Twitter for iPad" and tweet_source != "Twitter Web Client":
        return False

self.dataset["single_author"] = self.dataset["source"].apply(lambda source: lib_util.filter_papers(source))
self.dataset = self.dataset.drop(self.dataset[self.dataset['single_author'] == False].index)
```

L'analisi effettuata deve riguardare soltanto persone che hanno effettivamente possibilità di votare alle elezioni USA: non avrebbe senso includere cittadini di altri paesi. Ogni tweet in cui la feature "country" è diversa dagli Stati Uniti è scartato.

Le librerie Python per NLP da noi utilizzate lavorano meglio con l'inglese rispetto ad altre lingue: ci serviamo della libreria langdetect per rilevare la lingua del tweet, e scartarlo se non è in inglese.

In [ ]:
```
pip install langdetect
```

In [ ]:
```python
from langdetect import detect

def detect_lang(tweet):
    try:
        return detect(tweet)
    except langdetect.lang_detect_exception.LangDetectException as e:
        return None

self.dataset = self.dataset.drop(self.dataset[self.dataset['country'] != "United States of America"].index)
self.dataset["lang"] = self.dataset["tweet"].apply(lambda tweet: lib_util.detect_lang(tweet))
self.dataset = self.dataset.drop(self.dataset[self.dataset['lang'] != "en"].index)
```

Infine, dopo aver applicato la NLP pipeline ad ogni tweet (tokenizzazione, lemmatizzazione, rimozione di punteggiatura, link, simboli non alfanumerici e stop words) si decide, per facilitare le prossime analisi (sarà tutto spiegato nella sezione successiva), di scartare da ogni sottodataset tutti i tweet in cui viene menzionato anche l'altro candidato. Il preprocessing viene effettuato anche sulle descrizioni degli utenti, ci serviranno dopo.

N.B. Sempre per facilitare le prossime analisi, dalla lista delle stop words vengono rimosse le negazioni, e la parola "but".

In [ ]:
```python
if self.candidate == "biden":
    self.dataset = self.dataset.drop(self.dataset[self.dataset['tweet'].str.contains("trump")].index)
else:
    self.dataset = self.dataset.drop(self.dataset[self.dataset['tweet'].str.contains("biden")].index)
```

3. METODO UTILIZZATO

Per allenare e validare il nostro classificatore necessitiamo ovviamente di un dataset con tuple già etichettate col partito d'appartenenza dall'autore dal tweet. Problema: il dataset non è etichettato in questo modo, è etichettato con il candidato di cui si parla nel tweet. Avendo più di un milione di record, è praticamente impossibile etichettarle a mano; bisogna trovare un modo per etichettarle automaticamente.

3.1 SENTIMENT ANALYSIS

3.1.1 VADER

La prima parte del progetto sarà quindi una parte di sentiment analysis, utilizzando il modello VADER (Valence Aware Dictionary and sEntiment Reasoner), particolarmente adatto ad analizzare tweet e in generale testi presi dai social media. Il modello VADER si basa su un lexicon (dizionario) di parole a cui è stato assegnato un punteggio di polarità a priori, che va da -4 a 4. La polarità totale (compound) di una frase viene calcolata con la formula

$$x = \frac{z}{\sqrt{z^2 + \alpha}}$$

dove

$$z = \sum_i s(w_i)$$

dove s è il punteggio associato alla parola nel lexicon, ed α è un iperparametro, solitamente settato uguale a 15. Il punteggio viene poi modificato in base a 5 euristiche:

- Punteggiatura: se il sentimento compound della frase è positivo, viene aggiunto 0.292 per ogni punto esclamativo, e 0.18 per ogni punto interrogativo. Se è negativo, questi valori vengono sottratti;
- Maiuscole: per ogni maiuscola, se il sentimento è positivo/negativo viene aggiunto/sottratto 0.733;
- Modificatori di polarità: alcune parole particolari messe accanto ad una parola "normale" contribuiscono a sommare/sottrarre 0.293 dal punteggio della parola "normale", a seconda se il suo punteggio è positivo/negativo;
- Cambiamento di polarità: la parola "but" (ma) indica cambio di polarità rispetto alla prima parte della frase; il sentimento delle parole che vengono prima/dopo viene abbassato/aumentato del 50%;
- Negazioni: per ogni parola, vengono controllate le tre parole precedenti: se è presente una negazione tra esse, il sentimento della parola viene diminuito di 0.74.

N.B. L'ultimo punto ci fa capire il motivo per cui abbiamo rimosso le negazioni dalle stop words.

3.1.2 IMPLEMENTAZIONE

La libreria NLTK implementa il modello VADER in Python, con tanto di lexicon modificabile: lo modificheremo, aggiungendo nuovi termini, per aumentare la precisione delle etichettature. Creiamo un oggetto SentimentIntensityAnalyzer ed eseguiamo il metodo polarity_scores, passandogli ogni tweet, e memorizziamo il risultato in una nuova colonna "label".

In [ ]:
```python
    def analyze(self):
        labels = ["Very negative", "Negative", "Neutral", "Positive", "Very positive"]
        labeled_dataset = self.dataset.copy()
        labeled_dataset['label'] = self.dataset['tweet'].apply(lambda tweet: self.analyzer.polarity_scores(tweet)['compound'])

        i = 0
        for row in labeled_dataset.iterrows():
            row[1]["label"] = lib_util.compound(row[1]["tweet"], row[1]["label"], self.candidate)
            i+=1

        labeled_dataset['label'] = pd.cut(labeled_dataset['label'], bins=5, labels=labels)
        labeled_dataset = labeled_dataset.where(labeled_dataset['label'] != "Neutral").dropna()
        labeled_dataset = labeled_dataset.where(labeled_dataset['label'] != "Positive").dropna()
        labeled_dataset = labeled_dataset.where(labeled_dataset['label'] != "Negative").dropna()
        if self.candidate == "biden":
            labeled_dataset["label"] = labeled_dataset["label"].replace({"Very positive": "DEM"})
            labeled_dataset["label"] = labeled_dataset["label"].replace({"Very negative": "GOP"})
        else:
            labeled_dataset["label"] = labeled_dataset["label"].replace({"Very positive": "GOP"})
            labeled_dataset["label"] = labeled_dataset["label"].replace({"Very negative": "DEM"})
        return labeled_dataset
```

Siamo consci che non possiamo etichettare con buona precisione soltanto in base allo score ritornato dal modello VADER: un tweet potrebbe essere non abbastanza polarizzato, oppure alcune parole che esprimono un forte sentimento soltanto in particolari contesti hanno un punteggio basso nel lexicon VADER. Decidiamo di sfruttare anche la descrizione del profilo associato al tweet, perchè la bio di un utente potrebbe rivelare molto sulle sue ideologie politiche, e modificare di conseguenza lo score di polarità.

Creiamo quindi due altri lexicon: uno contenente parole associate al partito democratico, l'altro contenente parole associate al partito repubblicano, con un punteggio assegnato in base a quanto è decisiva nel farci comprendere l'orientamento politico dell'autore. Esempio: se nella bio è presente l'hashtag "voteblue" è chiaro che l'autore è di orientamento DEM, quindi alla parola è associato un punteggio alto, mentre se è presente la parola "christian", ci fa capire che l'autore è di religione cristiana; una buona parte dei cristiani praticanti in USA è repubblicana, ma ovviamente non è possibile fare assunzioni forti, quindi la parola avrà un punteggio basso.

I due lexicon si applicano specificatamente alle bio. Dopo averli applicati, ogni bio avrà assegnato un punteggio "DEM" e un punteggio "GOP" (fino a 0.5); sarà preso in considerazione il punteggio più alto. Se il candidato è dello stesso partito, il punteggio sarà aggiunto allo score compound ottenuto da VADER, altrimenti sarà sottratto. Se il punteggio finale risulta maggiore di 1/minore di -1, viene lasciato a 1/-1.

Per avere una buona probabilità che le tuple siano state etichettate correttamente, selezioniamo soltanto le tuple abbastanza polarizzate, cioè con un punteggio compound abbastanza vicino ad 1/-1.

Tutti i tre lexicon si trovano nel file "lexicon_update.py".

In [ ]:
```python
def compound(text, row_compound, candidate):
    pts_dem = 0
    text = text.lower()
    for word in dem_lexicon():
        if word in text:
            pts_dem += dem_lexicon()[word]
    pts_gop = 0
    for word in gop_lexicon():
        if word in text:
            pts_gop += gop_lexicon()[word]
    if candidate == "biden":
        if pts_dem > pts_gop:
            if pts_dem > 0.5:
                row_compound += 0.5
            else:
                row_compound += pts_dem
        elif pts_dem < pts_gop:
            if pts_gop > 0.5:
                row_compound -= 0.5
            else:
                row_compound -= pts_gop
    else:
        if pts_gop > pts_dem:
            if pts_gop > 0.5:
                row_compound += 0.5
            else:
                row_compound += pts_gop
        elif pts_gop < pts_dem:
            if pts_dem > 0.5:
                row_compound -= 0.5
            else:
                row_compound -= pts_dem

    if row_compound > 1.0:
        row_compound = 1.0
    elif row_compound < -1.0:
        row_compound = -1.0

    return row_compound
```

3.2 CLASSIFICATORE

3.2.1 WORD/SENTENCE EMBEDDING

Per creare gli embedding dei tweet proviamo ad utilizzare un modello di word embedding Word2Vec, precisamente un modello preallenato GLOVE ("glove-twitter-200") allenato su un dataset di 1 mld di tweet, in cui ogni embedding è un vettore di 200 elementi. Per ottenere un embedding dell'intero tweet, si calcola la media dei vettori-parola, per ogni parola componente il tweet. Al momento dei test sul classificatore, notiamo che l'accuracy offerta da qualunque modello provato è abbastanza bassa. Pensiamo allora ad utilizzare un sentence embedding preallenato basato su transformers, cioè SBERT (Sentence BERT), una variante del modello BERT allenato specificatamente su frasi e testi di poche centinaia di parole (esattamente quello che serve in questo caso); che permette di ottenere un classificatore molto più preciso, come si vedrà nella sezione dei risultati. Maggiori informazioni su SBERT possono essere trovate nel paper originale, lasciato in allegato.

3.2.2 TRAINING E TEST CLASSIFICATORE

Dopo una serie di test, confrontando K-NN, SVM, e diversi modelli ensemble come AdaBoost e Random Forest, il classificatore migliore risulta SVM, con parametri (C=10, gamma=1, kernel="rbf"). Dopo aver allenato e validato il modello, ci accorgiamo di un problema: la recall e la precision della classe 0 (GOP) sono basse (di conseguenza anche F1-score); in altre parole, la maggior parte dei record GOP viene associata alla classe DEM, mentre la maggior parte dei record DEM viene classificata correttamente come DEM.

SVM (C=10, gamma=1, kernel="rbf"), WORD2VEC, NO SMOTE

1960	4368
612	8569

Si può facilmente vedere che il dataset finale risultante dalla sentiment analysis è moderatamente sbilanciato, con circa 28k record etichettati GOP e 49k record etichettati DEM. A costo di perdere precisione sulla classe DEM, dobbiamo cercare di aumentare l'F1-score della classe GOP. Decidiamo quindi di sfruttare un modello di over-sampling (creare delle tuple fittizie della classe di minoranza per bilanciare il dataset), precisamente il SVMSMOTE, implementato nella libreria imbalanced-learn.

L'algoritmo di resampling SMOTE non duplica alcune delle tuple selezionate casualmente, ma ne crea di "artificiali" attraverso l'interpolazione. La variante da noi utilizzata sfrutta il classificatore SVM per dividere in classi ("noise", "in danger", "safe") le tuple, e generare le nuove interpolando quelle esistenti che sono più vicine ai bordi delle classi (quelle "in danger"). Questo migliora le performance del classificatore che sarà allenato su questi dati. (fonte)

Dopo aver creato gli embedding, si applica SVMSMOTE e si resampla il dataset; infine si allena il classificatore sul dataset resamplato. La validazione avviene comunque sul test-set originale (20% del dataset), con soli embedding "reali".

Passando da Word2Vec a SBERT, si nota che i risultati con SMOTE e senza SMOTE sono così simili tra loro che non è giustificato il suo impiego (lo vedremo nei risultati).

In [ ]:
```python
import pickle
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.utils import shuffle
from sentence_transformers import SentenceTransformer
from imblearn.over_sampling import SVMSMOTE
import numpy as np
import matplotlib.pyplot as plt

class TweetClassifier:
    def __init__(self, dataset):
        dataset = shuffle(dataset)
        self.embedder = SentenceTransformer("all-MiniLM-L6-v2")
        self.train, self.test = train_test_split(dataset, test_size=0.2, stratify=dataset["label"], random_state=42)
        self.model = self._train_model()
        self._test_model()

    def _train_model(self):
        model = SVC(C=10, gamma=1, kernel="rbf")
        list_texts = self.train["tweet"].tolist()
        embedded_train = self.embedder.encode(list_texts)
        labels = np.array(self.train["label"])
        model.fit(embedded_train, labels)
        return model

    def _test_model(self):
        list_texts = self.test["tweet"].tolist()
        embedded_test = self.embedder.encode(list_texts)
        predictions = self.model.predict(embedded_test)
        cm = confusion_matrix(self.test["label"], predictions)
        ConfusionMatrixDisplay(cm).plot()
        plt.show()
        print(classification_report(self.test["label"], predictions))


    def save_model(self):
        with open('tweet_classifier.pkl','wb') as f:
            pickle.dump(self.model,f)
```

4. RISULTATI

In questa sezione verranno mostrati tutti i risultati dei vari classificatori ed embedding provati, mostrando per ognuno di essi la matrice di confusione, accuracy, precision, recall, f1-score (tutti con SMOTE, tranne se specificato il contrario).

MATRICI DI CONFUSIONE:

K-NN (5 vicini), SBERT

5953	375
3857	5324

K-NN (5 vicini), WORD2VEC

3989	2339
3490	5691

SVM (C=10, kernel=rbf, gamma=1), SBERT

5334	994
875	8306

SVM (C=10, kernel=rbf, gamma=1), SBERT, NO SMOTE

5355	973
825	8356

SVM (C=10, kernel=rbf, gamma=1), WORD2VEC

4174	2154
2914	6267

RANDOM FOREST (200 estimators), SBERT

5115	1213
1652	7529

RANDOM FOREST (200 estimators), WORD2VEC

3502	2826
2088	7093

]

ADABOOST (100 estimators), SBERT

4556	1772
2657	6524

ADABOOST (100 estimators), WORD2VEC

3758	2570
3355	5826

METRICHE

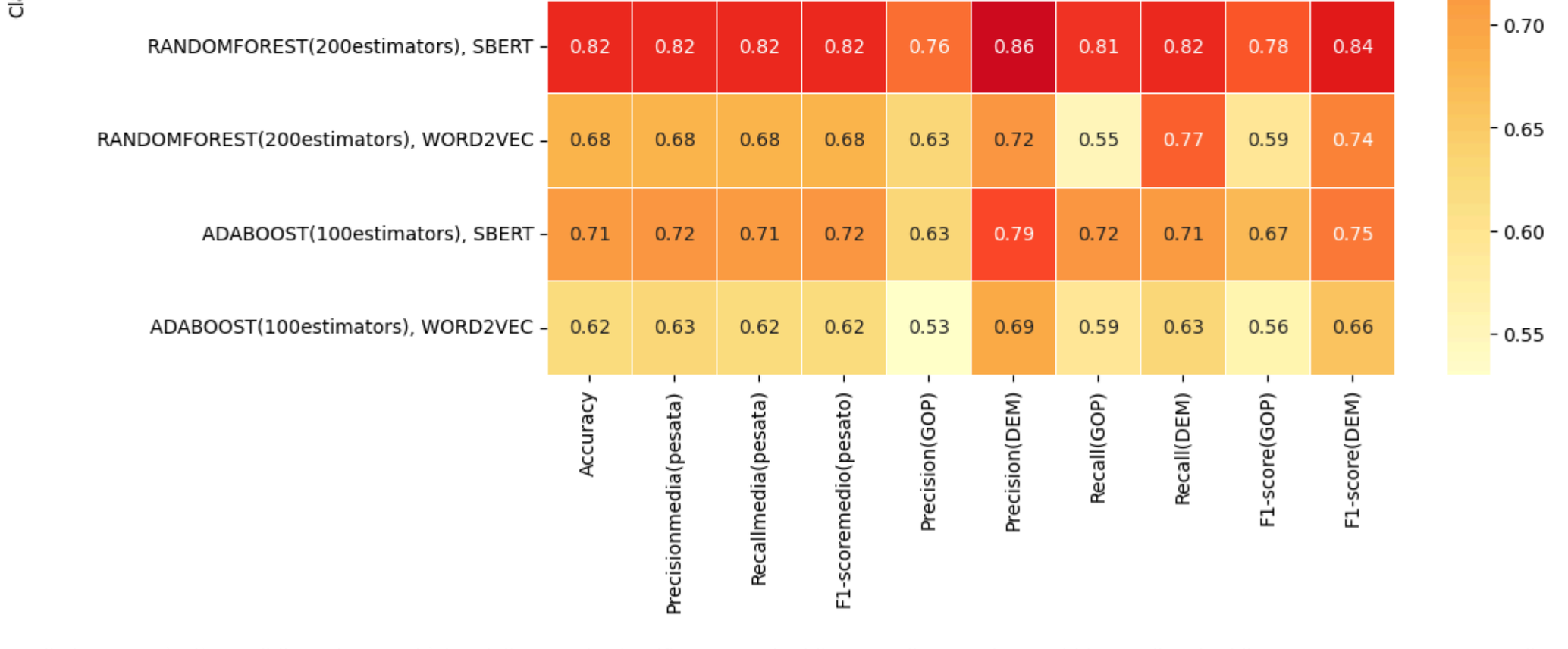
Supporto classe DEM: 9181
Supporto classe GOP: 6328

Classificatore	Embedding	Accuracy	Precision media (pesata)	Recall media (pesata)	F1-score medio (pesato)	Precision (GOP)	Precision (DEM)	Recall (GOP)	Recall (DEM)	F1-score (GOP)	F1-score (DEM)
K-NN (5 vicini)	SBERT	0.73	0.80	0.73	0.72	0.61	0.93	0.94	0.58	0.74	0.72
K-NN (5 vicini)	WORD2VEC	0.62	0.64	0.62	0.63	0.53	0.71	0.63	0.62	0.58	0.66
SVM (C=10, kernel=rbf, gamma=1)	SBERT	0.88	0.88	0.88	0.88	0.86	0.89	0.84	0.90	0.85	0.90
SVM (C=10, kernel=rbf, gamma=1)	WORD2VEC	0.67	0.68	0.67	0.68	0.59	0.74	0.66	0.68	0.62	0.71
SVM (C=10, kernel=rbf, gamma=1), NO SMOTE	SBERT	0.88	0.88	0.88	0.88	0.87	0.90	0.85	0.91	0.86	0.90
RANDOM FOREST (200 estimators)	SBERT	0.82	0.82	0.82	0.82	0.76	0.86	0.81	0.82	0.78	0.84
RANDOM FOREST (200 estimators)	WORD2VEC	0.68	0.68	0.68	0.68	0.63	0.72	0.55	0.77	0.59	0.74
ADABOOST (100 estimators)	SBERT	0.71	0.72	0.71	0.72	0.63	0.79	0.72	0.71	0.67	0.75
ADABOOST (100 estimators)	WORD2VEC	0.62	0.63	0.62	0.62	0.53	0.69	0.59	0.63	0.56	0.66

In [ ]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


table = pd.read_json("tabella.json")
table["Classificatore"] = table["Classificatore"] + ", " + table["Embedding"]
table.set_index("Classificatore", inplace=True)
heatmap_data = table.drop(columns=["Embedding"])

plt.figure(figsize=(10, 8))
heatmap = sns.heatmap(heatmap_data, annot=True, cmap="YlOrRd", linewidths=0.5, fmt=".2f")
plt.title('Metriche')
plt.show()
```



Dalla heatmap è già possibile vedere qual è la miglior coppia classificatore-embedding, per distacco: l'SVM abbinato all'embedding SBERT, con un'accuracy di 0.88 e tutte le metriche che oscillano tra 0.84 e 0.90; l'unico che riesce ad avvicinarsi a queste prestazioni è il Random Forest, sempre abbinato all'embedding SBERT. Sarebbe stato interessante vedere come si comportasse Random Forest aumentando il numero di estimatori, ma i limiti prestazionali della macchina utilizzata l'hanno impedito.

Intanto, possiamo osservare il problema descritto sopra, quello per cui abbiamo dovuto usare l'SVMSMOTE: i classificatori soffrono particolarmente con la classe "GOP", che ha metriche sempre inferiori rispetto all'altra. Nonostante l'introduzione del resampler, con alcuni classificatori permane il problema (anche se mitigato). Nel classificatore da noi scelto, il problema viene risolto quasi del tutto.

L'embedding SBERT migliora in tutti i casi le prestazioni, ma le migliora particolarmente quando abbinato con un classificatore che già di suo ottiene buone performance, ad esempio SVM (accuracy aumenta da 0.67 a 0.88) e il Random Forest (da 0.68 a 0.82). Infatti si osserva che il peggior classificatore con SBERT (K-NN) ottiene comunque prestazioni di poco migliori rispetto al miglior classificatore con Word2Vec (SVM); molto probabilmente questo embedding riesce a distinguere i tweet abbastanza da permettere al classificatore di dividere bene le zone dello spazio, nonostante facciano tutti parte dello stesso topic (politica). Come notato durante la realizzazione di questo progetto, l'embedding Word2Vec soffre nel rappresentare le differenze tra i vari tweet: anche se di polarità diverse, la similarità del coseno risultante tra due di essi è sempre molto vicina ad 1. Da ciò possiamo concludere che Word2Vec, essendo basato sulla matrice di co-occorrenza e sul contesto, riesce bene a distinguere diversi topic, ma non a distinguere due testi dello stesso topic.

Interessante notare che con Word2Vec il Random Forest si comportasse poco meglio dell'SVM, ma usando SBERT, SVM migliori molto più di quanto migliora Random Forest.

Infine, selezioniamo SVM senza SMOTE, alla luce dei risultati ottenuti.

5. CONCLUSIONI

Come si può vedere dalla sezione dei risultati, siamo riusciti ad ottenere un classificatore con buone prestazioni, sfruttando modelli di embedding e modelli di classificazioni ad alte performance, cioè SBERT e SVM (con kernel RBF, che ci permette di lavorare anche con dati non linearmente separabili).

Il primo problema è stato quello di filtrare ed etichettare il dataset: la maggior parte dei record erano completamente inutili per il nostro scopo per vari motivi, e dovevano essere scartati. Rimanendo comunque con decine di migliaia di record, l'etichettatura sarebbe stato un lavoro impossibile da svolgere per una sola persona, e non si dispongono delle risorse economiche per sfruttare il servizio Mechanical Turk. L'unico modo è quello di automatizzare l'etichettatura, sfruttando un qualche modello (VADER) di sentiment analysis. L'idea di eseguire sentiment analysis viene da un semplice assunto: chi elogia il candidato (sentiment positivo) probabilmente voterà per lui (quindi supporta il suo partito), chi lo critica (sentiment negativo), probabilmente voterà per l'altro, quindi supporta l'altro partito. I risultati della sentiment analysis sono stati arricchiti utilizzando due lexicon, per aumentare la precisione dell'etichettatura e avvicinarla a quella ottenuta da un operatore umano.

Ottenute le etichette, il lavoro in teoria dovrebbe semplificarsi: basta creare gli embedding, e testare diversi classificatori con cross-validation per selezionare il migliore. Sopraggiunge però il problema dello sbilanciamento del dataset, che fa ottenere un cattivo F1-score: per risolverlo, bisogna resamplare il dataset, lo facciamo inizialmente con SVMSMOTE.

Il problema successivo è quello di scegliere il modello di embedding che massimizzi le metriche prese in considerazione. Non soddisfatti dei risultati ottenuti con Word2Vec, decidiamo di cercare un modello tra quelli basati su trasformatori (su Hugging Face), trovando il modello SBERT, pensato appositamente per questo scopo (embedding di frasi/piccoli testi), al contrario di Word2Vec che è pensato per le singole parole (otteniamo il sentence embedding facendo una semplice media). I risultati finali ci hanno dato ragione: SBERT ha migliorato tutte le metriche, e ci ha permesso di scegliere un modello di classificazione con buone prestazioni (SVM). Il modello SBERT ci permette anche di fare a meno di SMOTE.

In conclusione, utilizzando un modello di sentence embedding abbastanza performante, è possibile classificare con buone performance dei tweet politici, individuando il partito votato/supportato dall'autore del tweet, nonostante il task venga complicato dal fatto che i testi facciano parte dello stesso topic.