

# Progetto di Linguistica Computazionale (modulo B)

Caccamo Mariachiara - Piemonte Alessio

A.A. 2023/2024

# Introduzione

Il codice fornito implementa un sistema per rilevare le fake news utilizzando tecniche di apprendimento automatico e di analisi del linguaggio naturale (NLP). Utilizza il dataset di Mohammad Javad Pirhadi per il rilevamento delle fake news, la libreria spacy per l'NLP e il modello SentenceTransformer per trasformare i testi in vettori. Le tecniche di apprendimento automatico utilizzate includono il classificatore NearestNeighbors e un classificatore di rete neurale MLP (multi-layer perceptron).

# Librerie utilizzate

- **Sentence Transformers (SBERT)**: libreria utilizzata per l'elaborazione del linguaggio naturale che fornisce modelli pre-addestrati di trasformatori di frase per l'embedding di testo
- **spaCy**: package alternativo a **nltk** che permette di effettuare operazioni di manipolazione avanzate sul linguaggio naturale (per esempio per il POS tagging o la NER)
- **Scikit-learn**: libreria per l'apprendimento automatico in Python che fornisce una vasta gamma di algoritmi per la classificazione, la regressione, il clustering e molti altri compiti di apprendimento automatico. In particolare sono stati utilizzati le seguenti classi e funzioni:
  - **NearestNeighbors**: una classe per la ricerca dei vicini
  - **MLPClassifier**: una classe per la classificazione con perceptron multistrato
  - **train\_test\_split**: una funzione per dividere i set di dati in training e testing
  - **accuracy\_score**, **precision\_score**, **recall\_score**, **f1\_score**: funzioni per valutare le prestazioni dei modelli di apprendimento automatico
- **NumPy**: libreria utilizzata per lavorare con gli array
- **Matplotlib**: è una libreria utilizzata per la creazione di grafici per il linguaggio di programmazione Python e la libreria matematica NumPy

# Fase 1

**Caricamento del dataset, del modello di fake-news detection, del modello NLP in inglese da SBERT e applicazione del modello per la trasformazione delle frasi in vettori (Caccamo)**

```
dataset = load_dataset("mohammadjavadpirhadi/fake-news-detection-dataset-english")
data = DataFrame(dataset['train'])[['text", "subject", "label"]]
nlp = spacy.load("en_core_web_sm")
model = SentenceTransformer("distiluse-base-multilingual-cased-v1")
embeddings = model.encode(data["text"])
```

In questa fase è stato caricato il Dataset, costruito un DataFrame di Pandas dalla divisione del train del dataset estraendo le colonne di “text”, “subject” e “label”, che contengono rispettivamente il testo degli articoli, il topic e la label di fake o non fake-news. Dopo aver caricato il modello spaCy pre-addestrato “en\_core\_web\_sm” questo è stato applicato al DataFrame per calcolare gli embedding di frase per la colonna “text”.

# Fase 2

## Creazione di un'istanza del classificatore NearestNeighbors e addestramento del classificatore sui dati (Piemonte)

```
nbrs = NearestNeighbors(n_neighbors=5)  
nbrs.fit(embeddings)
```

In questa fase è stato creata un'istanza della classe `NearestNeighbors` dal modulo `neighbors` di Scikit-learn. Questa classe viene utilizzata per l'apprendimento dei vicini più vicini non supervisionato. Il parametro `n_neighbors` è impostato su 5, per selezionare i 5 vicini più vicini per ogni punto dato. È stata poi addestrata l'istanza di `NearestNeighbors` sul Dataset di `embeddings`. Attraverso il metodo `fit` viene poi preso in input il Dataset e costruita una struttura di dati che consente di eseguire ricerche di vicini più vicini efficienti. In questo caso, `embeddings` è un array 2D dove ogni riga rappresenta un punto dati (ad esempio, una frase o un documento) e ogni colonna rappresenta una caratteristica (ad esempio, una dimensione di embedding di parole). Il metodo `fit` indicizza gli `embeddings` in modo da poter trovare rapidamente i vicini più vicini per un punto dati di query.

# Fase 3

## Test e training (Caccamo)

- Divisione del dataset in un training set e in un test set:
  - Il parametro `test_size=0.2` indica che il 20% del dataset sarà utilizzato per il test, mentre il resto sarà utilizzato per l'addestramento
  - Il parametro `random_state=42` assicura che la divisione dei dati sia sempre la stessa, anche se il codice viene eseguito più volte
- Codifica delle frasi del training set e addestramento della rete neurale MLP
- Codifica delle frasi e predizione delle label delle frasi del test set
- Valutazione delle prestazioni del classificatore, in particolare i risultati ottenuti sono stati:
  - **Accuracy:** 0.9983296213808464
  - **Precision:** 0.9986835176408636
  - **Recall:** 0.9981578947368421
  - **F1-score:** 0.9984206370097395

```
X = data[["text"]]
y = data["label"]
X_train, X_test, y_train, y_test = train_test_split(*arrays: X, y, test_size=0.2, random_state=42)

embeddings_train = model.encode(X_train["text"].tolist())

mlp = MLPClassifier(hidden_layer_sizes=[1000])
mlp.fit(embeddings_train, y_train)

embeddings_test = model.encode(X_test["text"].tolist())

y_pred = mlp.predict(embeddings_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

predictions = mlp.predict(embeddings_test)
```

# Fase 4

```
for index, frase in data.iterrows():
    embedding = model.encode([frase["text"]]).reshape(1, -1)
    distances, indices = nbrs.kneighbors(embedding)
    nearest_index = indices[0][0]
    topic_prediction = data["subject"][nearest_index]
    fake_news_embedding = model.encode([frase["text"]]).reshape(1, -1)
    fake_news_prediction = mlp.predict(fake_news_embedding)
    data3 = nlp(frase["text"])

    pos_count = defaultdict(int)
    ner_count = defaultdict(int)
    for frase_count in data3:
        pos_count[frase_count.pos_] += 1
    for frase1_count in data3.ents:
        ner_count[frase1_count.label_] += 1
    pos = [frase2.pos_ for frase2 in data3]
    ner = [frase3.label_ for frase3 in data3.ents]
    topic = frase["subject"]
    if frase["label"] == 1: # fake news
        pos_fake.extend(pos)
        ner_fake.extend(ner)
        topic_fake[topic] = topic_fake.get(topic, 0) + 1
        word_count_fake.append(len(frase["text"].split()))
    else: # real news
        pos_non_fake.extend(pos)
        ner_non_fake.extend(ner)
        topic_non_fake[topic] = topic_non_fake.get(topic, 0) + 1
        word_count_non_fake.append(len(frase["text"].split()))
```

## Calcolo degli embedding, predizione dei topic (Piemonte) e delle label (Caccamo), POS, NER, NearestNeighbors

- Il codice utilizza un *ciclo for* per iterare sul dataset **data** utilizzando il metodo *iterrows()* che restituisce un iteratore che produce coppie di valori (*index*, *frase*), dove *index* è l'indice della riga del dataset e *frase* è la riga stessa
- Per ogni frase calcola l'embedding utilizzando il modello di SentenceTransformer precedentemente caricato. L'embedding risultante è un array 1D, che è ridimensionato in un array 2D con forma (1, -1) utilizzando il metodo *reshape*. Questo perché il metodo *kneighbors* si aspetta un array 2D come input (Piemonte)
- Utilizzando l'oggetto **nbrs** (che è un oggetto NearestNeighbors di Scikit-learn) il codice calcola le distanze e gli indici dei vicini più vicini dell'embedding della frase. Il metodo *kneighbors* restituisce due array:
  - **Distances**: contiene le distanze tra l'embedding della frase e gli embeddings dei vicini più vicini
  - **Indices**: contiene gli indici dei vicini più vicini
- Individuato l'indice del più vicino, questo viene utilizzato per predire il topic della frase, selezionato dalla colonna *subject* del dataset all'indice *nearest\_index*
- In sintesi, il codice utilizza la classe NearestNeighbors per trovare i 5 vicini più vicini per ogni campione di testo nel dataset, e utilizza l'etichetta di argomento del vicino più vicino come previsione dell'etichetta di argomento per il campione di testo corrente (Piemonte)
- Utilizzando l'oggetto **mlp** (che è un oggetto MLPClassifier di Scikit-learn) il codice predice per ciascuna frase se questa è fake news e tramite il metodo *predict* del classificatore restituisce la lista di etichette predette (Caccamo)
- Utilizzando l'oggetto **nlp** (che è un oggetto Language di SpaCy) il codice analizza la frase, restituendo un oggetto che contiene informazioni sulla frase, come ad esempio il POS tagging (Piemonte) e la Name Entity Recognition (Caccamo)

The background of the slide features a dynamic, abstract design composed of several overlapping semi-circles in various shades of green. These circles overlap in a way that creates a sense of depth and movement across the entire frame.

Rappresentazione grafica dei  
risultati ottenuti:

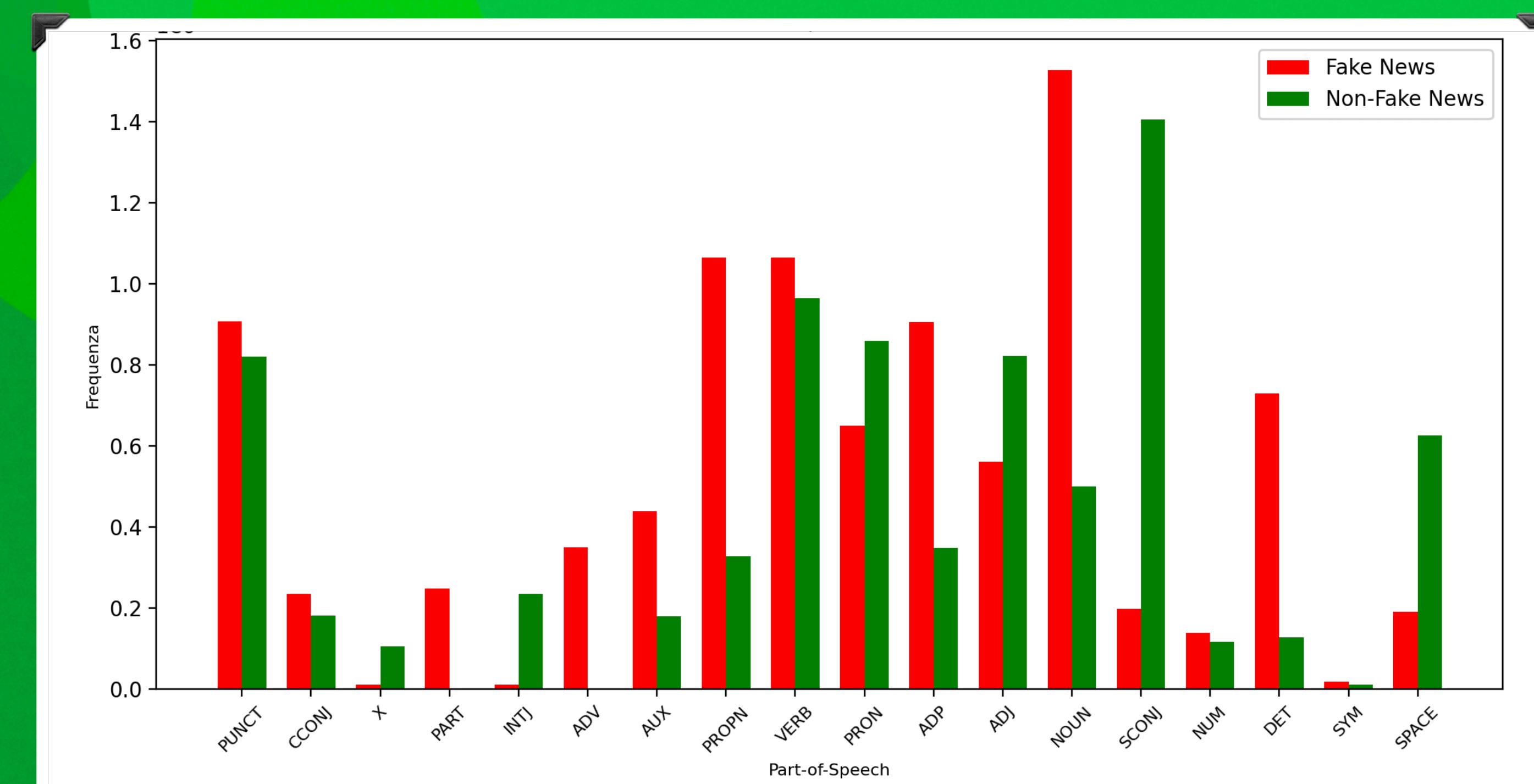
# Grafico 1

## Distribuzione POS per fake e non fake-news (Piemonte)

Il grafico evidenzia come la maggior parte delle **Part of Speech (POS)** sia maggiormente distribuita nelle *Fake News*, eccetto per:

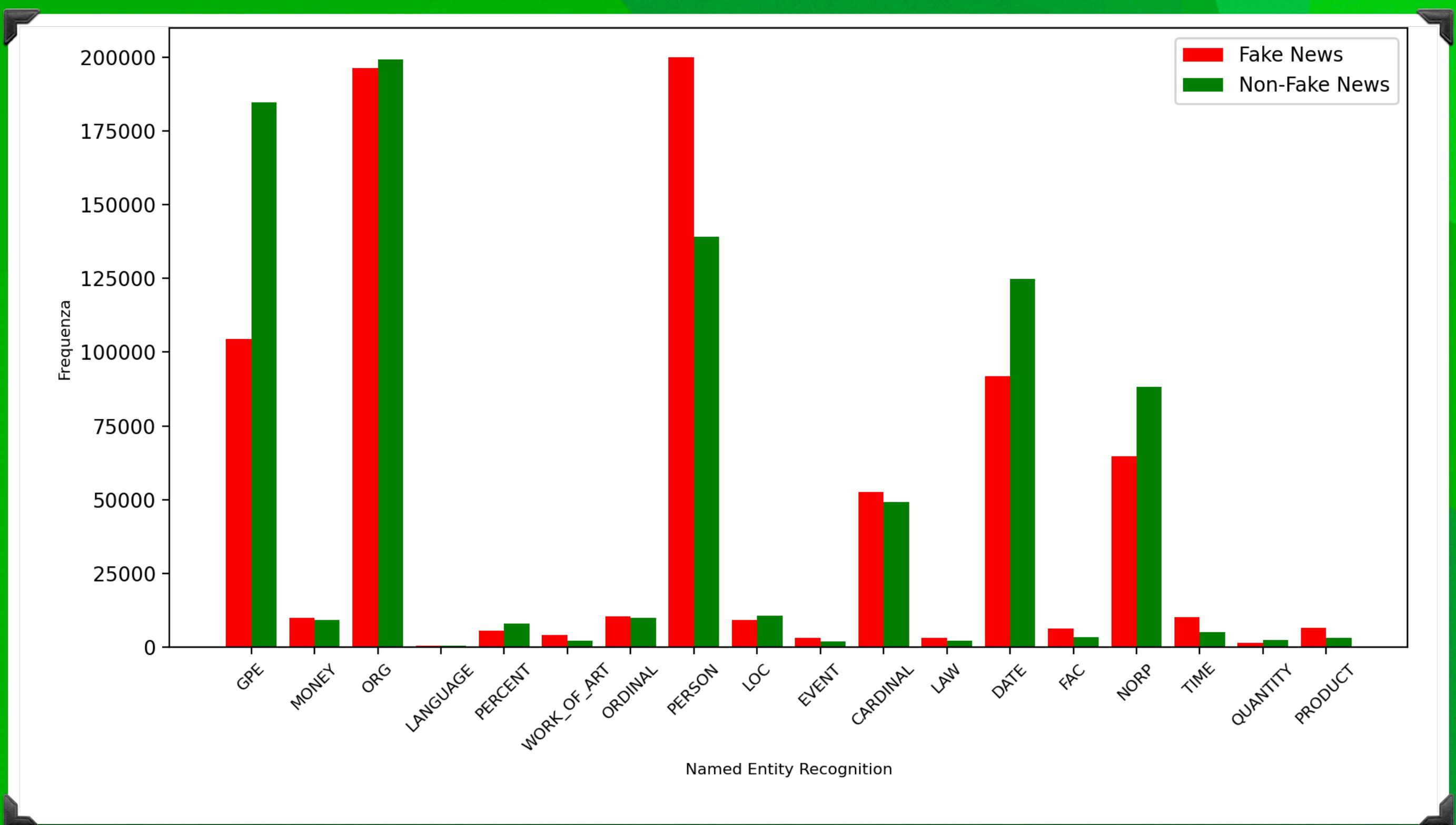
- SCONJ (congiunzioni subordinate)
- ADJ (aggettivi)
- PRON (pronomi)
- INTJ (interiezioni)
- X (unknown)

Questi risultati potrebbero suggerire che le fake-news tendono utilizzare uno stile più informale o colloquiale, mentre le notizie non fake sembrerebbero propendere a uno stile più formale.



# Grafico 2

## Distribuzione NER per fake e non fake-news (Caccamo)



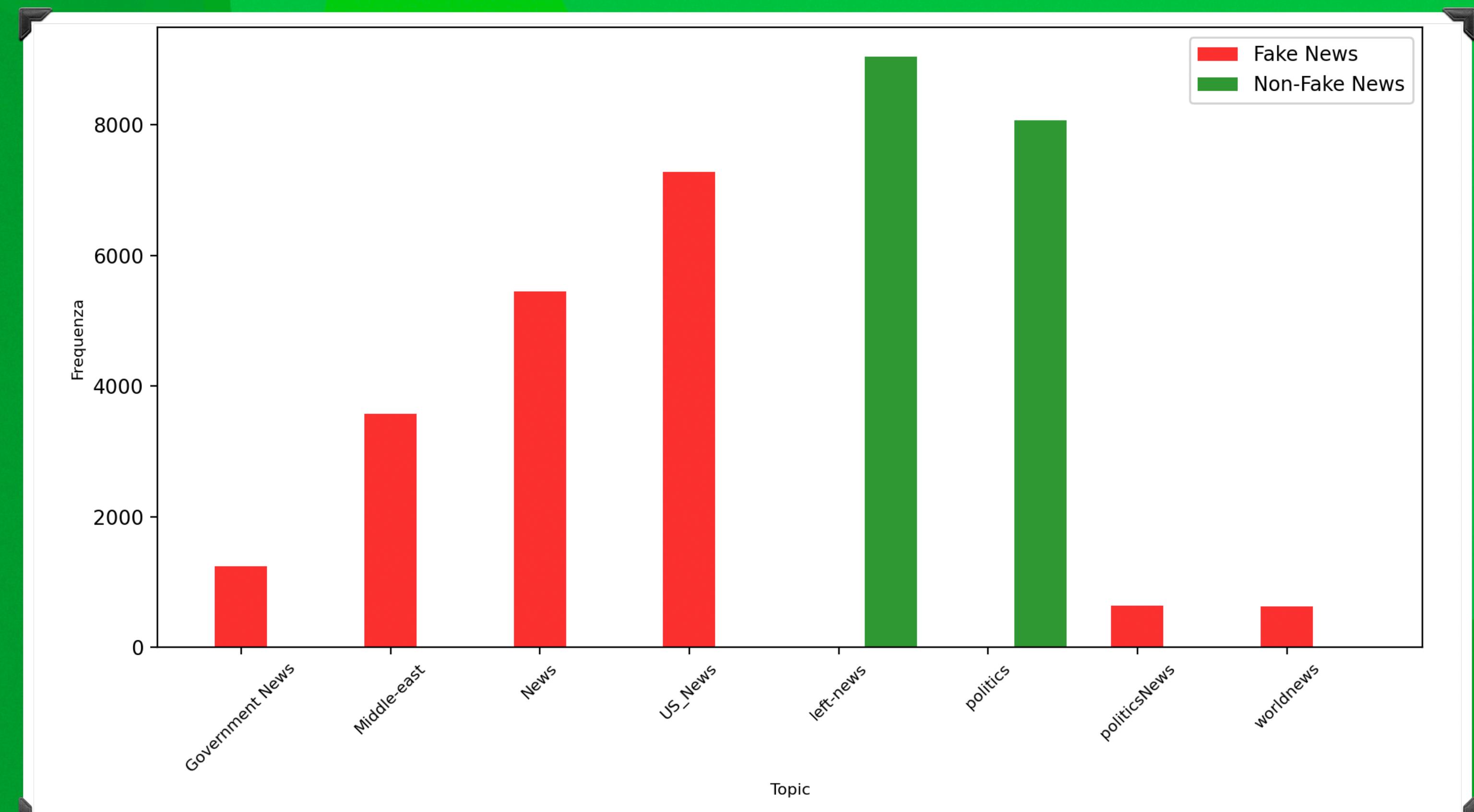
Il grafico mostra la distribuzione NER per fake e non-fake news. Le Name Entities sono pezzi di testo che si riferiscono a persone, luoghi, organizzazioni o cose specifiche. Si può notare che alcune *Name Entities* sono più comuni nelle fake news rispetto alle non-fake news, ad esempio:

- **PERSON:** questo risultato potrebbe essere conseguenza del fatto che le fake news utilizzano strategie come la citazione di "esperti" o "testimoni" falsi per dare credibilità alle loro affermazioni, o perché inventano storie che coinvolgono persone specifiche per renderle più convincenti
- **FAC (Facilities):** le fake news potrebbero usare più riferimenti a strutture fisiche (ad esempio luoghi istituzionali) per dare l'impressione di presunta accuratezza
- **MONEY:** le fake news potrebbero includere riferimenti a dati economici e cifre monetarie, ad esempio per promuovere teorie del complotto o per creare una sensazione di urgenza

# Grafico 3

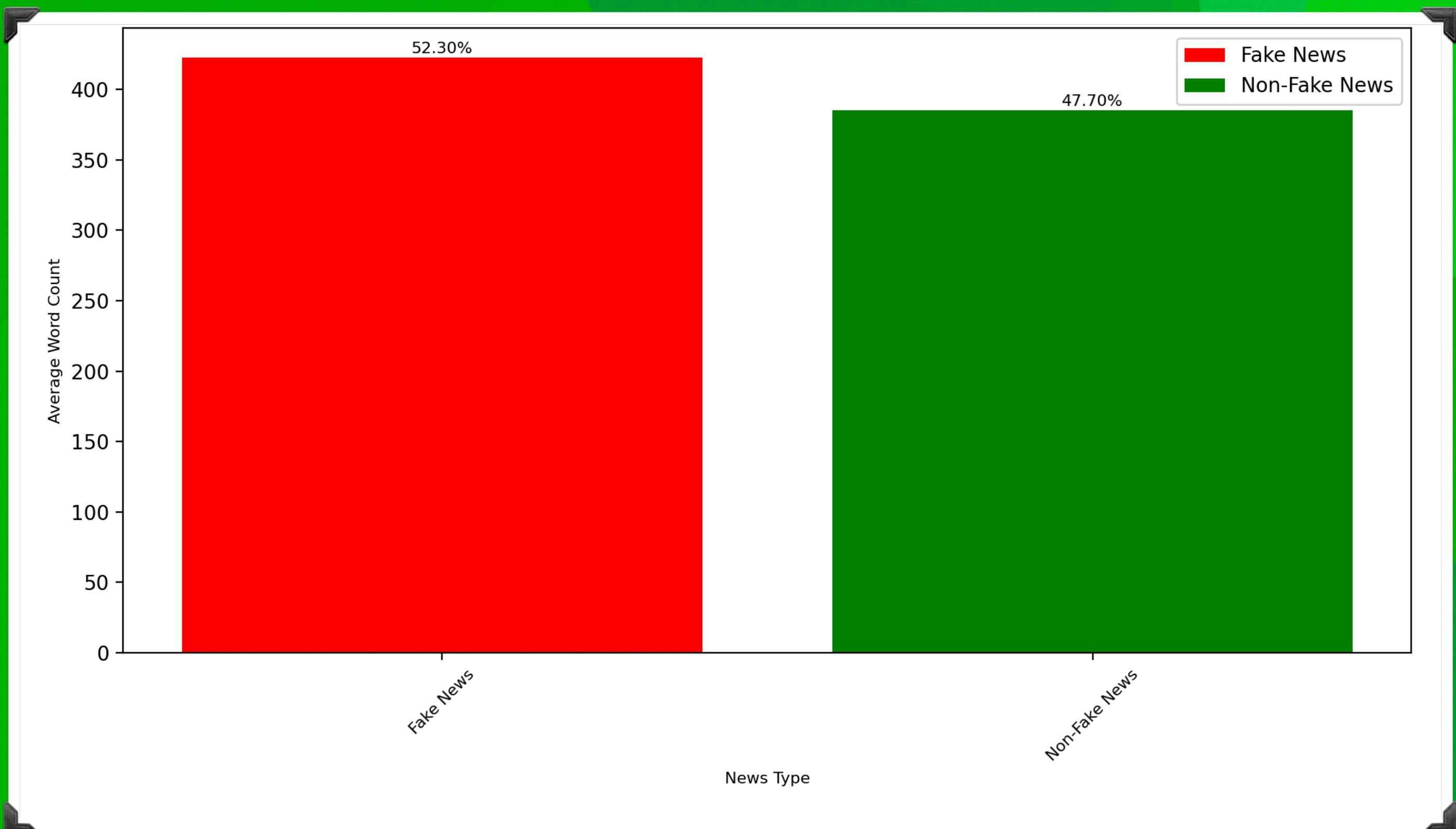
## Distribuzione topic per fake e non fake-news (Piemonte)

Il grafico mostra la distribuzione dei topic per fake e non fake-news mettendo in mostra il *bias* del dataset: ogni topic presenta una polarizzazione verso un'unica label. Ovvero, un topic presenta o solo notizie reali o solo notizie fake.



# Grafico 4

## Conteggio medio delle parole per le fake e non fake-news (Piemonte)



- **Conto delle parole medio per fake news:** 422.4532546266752

- **Conto delle parole medio per non fake news:** 385.36566553698725

Il conteggio medio delle parole è più alto nelle fake news questo perché probabilmente esse utilizzano spesso un linguaggio più sensazionalistico ed emotivo per attirare l'attenzione e manipolare i lettori.

Inoltre, le fake news potrebbero includere più citazioni, dichiarazioni o testimonianze di "esperti" o "testimoni" per rendere le loro affermazioni più credibili, il che contribuirebbe inevitabilmente ad aumentare il conteggio delle parole.

# Grafico 5

## Confronto tra le etichette reali e quelle previste per fake e non fake-news (Caccamo)

Il grafico confronta i risultati di predizione delle label di fake e non fake-news con i risultati reali. Gli esiti soddisfano le previsioni suggerendo che il modello riesce ad identificare correttamente le label.

