UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica
Corso di Machine Learning

Rilevamento di Annunci di Lavoro Fraudolenti mediante Machine Learning

Fraud Job Postings Detection Pipeline

Docenti:
Prof. Giuseppe Polese
Prof.ssa Loredana Caruccio

Studenti:
Alessandro D.
Gaetano A.

Anno Accademico 2025/2026

Abstract

Questo report presenta un'analisi completa di un sistema di classificazione binaria per la rilevazione di annunci di lavoro fraudolenti. Il dataset contiene 17.880 annunci di lavoro con 18 caratteristiche, tra cui dati testuali e metadati. L'approccio proposto combina tecniche di elaborazione del linguaggio naturale (NLP) con feature engineering avanzato e ensemble methods. Il modello migliore (Random Forest) raggiunge un F1-Score di 0.7041 sul test set con threshold standard a 0.5, migliorabile a **0.8023** mediante *threshold tuning* a 0.20.

Indice

Capitolo 1

Introduzione

1.1 Contesto e Motivazione

Il fenomeno delle truffe negli annunci di lavoro online è divenuto sempre più pervasivo negli ultimi anni, esponendo aziende, piattaforme di recruiting e candidati a rischi crescenti. Le modalità di attacco sono eterogenee e spaziano da campagne di phishing mirate al furto di identità, fino a complesse frodi finanziarie.

In tale scenario, lo sviluppo di un sistema automatico di rilevazione risulta fondamentale sotto molteplici aspetti. Primariamente, esso funge da barriera per **proteggere i candidati**, evitando che vengano esposti a tentativi di frode che potrebbero comprometterne la sicurezza dei dati personali ed economici. Parallelamente, l'automazione contribuisce a **mantenere la fiducia** nell'ecosistema del recruiting, garantendo la credibilità delle piattaforme che ospitano le offerte. Dal punto di vista aziendale, l'adozione di modelli predittivi permette di **ridurre significativamente i costi operativi** legati alla moderazione manuale, consentendo al contempo di **analizzare i pattern di frode** emergenti per comprendere e anticipare l'evoluzione delle strategie degli attaccanti.

1.2 Obiettivi del Progetto

L'obiettivo primario del progetto è lo sviluppo di una pipeline completa di Machine Learning dedicata alla classificazione binaria degli annunci. Il lavoro prende avvio dall'esplorazione e dal preprocessamento di un dataset reale, affrontando sfide critiche come la gestione di una notevole quantità di valori mancanti.

Nello specifico, l'approccio metodologico punta su un **feature engineering avanzato**, che combina tecniche di Natural Language Processing (NLP) per l'analisi dei testi con l'estrazione di metadati numerici e categorici. La fase sperimentale prevede l'addestramento e il confronto di diversi modelli di classificazione, la cui efficacia viene massimizzata attraverso tecniche di ottimizzazione come il *threshold tuning*. Infine, grande enfasi viene posta sull'analisi dei risultati e sull'interpretabilità dei modelli, elemento essenziale per comprenderne le decisioni in un contesto di sicurezza informatica.

1.3 Struttura del Report

Il report è organizzato come segue:

Capitolo 2 Analisi Esplorativa dei Dati (EDA)

Capitolo 3 Data Cleaning e Preprocessing

Capitolo 4 Feature Engineering

Capitolo 5 Preparazione Features e Splitting

Capitolo 6 Modelli e Metodologie di Training

Capitolo 7 Risultati e Valutazione

Capitolo 8 Analisi Dettagliata del Miglior Modello

Capitolo 9 Conclusioni e Lavori Futuri

Capitolo 2

Analisi Esplorativa dei Dati (EDA)

2.1 Descrizione Dataset

Il dataset contiene informazioni su annunci di lavoro provenienti da una piattaforma di recruiting online. Le statistiche principali sono:

Tabella 2.1: Statistiche Descrittive del Dataset

Metrica	**Valore**
Numero di campioni	17.880
Numero di caratteristiche	18
Numero di annunci reali	17.014 (95.16%)
Numero di annunci fraudolenti	866 (4.84%)

2.2 Descrizione Features

Il dataset è caratterizzato da una struttura eterogenea che combina dati non strutturati (testo libero) con metadati strutturati. Le variabili a disposizione possono essere suddivise in tre macro-categorie principali in base alla loro natura e al trattamento necessario in fase di preprocessing.

2.2.1 Features Testuali

Le variabili testuali costituiscono la parte più ricca di informazioni semantiche e richiedono tecniche di Natural Language Processing (NLP) per essere analizzate. Il nucleo dell'annuncio è rappresentato dal `title` e dalla `description`, che forniscono i dettagli essenziali sulla posizione aperta. A questi si affiancano campi specifici che delineano il contesto aziendale (`company_profile`), le competenze necessarie per la candidatura (`requirements`) e gli eventuali vantaggi offerti al dipendente (`benefits`).

2.2.2 Features Categoriche

Le variabili categoriche forniscono il contesto strutturale e classificatorio dell'offerta di lavoro. La dimensione geografica e organizzativa è catturata rispettivamente dalle feature `location` e `department`. Per quanto riguarda i dettagli contrattuali e professionali, il dataset include informazioni sulla tipologia di impiego (`employment_type`), sul livello di esperienza richiesto (`required_experience`) e sul grado di istruzione necessario (`required_education`). Infine, il posizionamento di mercato dell'azienda è definito dal settore industriale di appartenenza (`industry`) e dalla specifica funzione lavorativa (`function`).

2.2.3 Features Numeriche e Binarie

Quest'ultima categoria include indicatori diretti e metadati specifici. L'unica variabile contenente informazioni quantitative sull'offerta economica è `salary_range`, che specifica l'intervallo retributivo previsto. Le restanti feature sono indicatori binari (flag) che segnalano la presenza di determinate caratteristiche: la possibilità di lavorare da remoto (`telecommuting`), la presenza del logo aziendale nell'annuncio (`has_company_logo`) e l'esistenza di un questionario di screening per i candidati (`has_questions`).

2.3 Analisi Missing Values

Il dataset presenta un numero significativo di valori mancanti. La distribuzione è la seguente:

Tabella 2.2: Percentuale Missing Values per Colonna

Colonna	**Missing (%)**
salary_range	83.96%
department	64.58%
required_education	45.33%
benefits	40.34%
required_experience	39.43%
function	36.10%
industry	27.42%
employment_type	19.41%
company_profile	18.50%
requirements	15.08%
location	1.94%

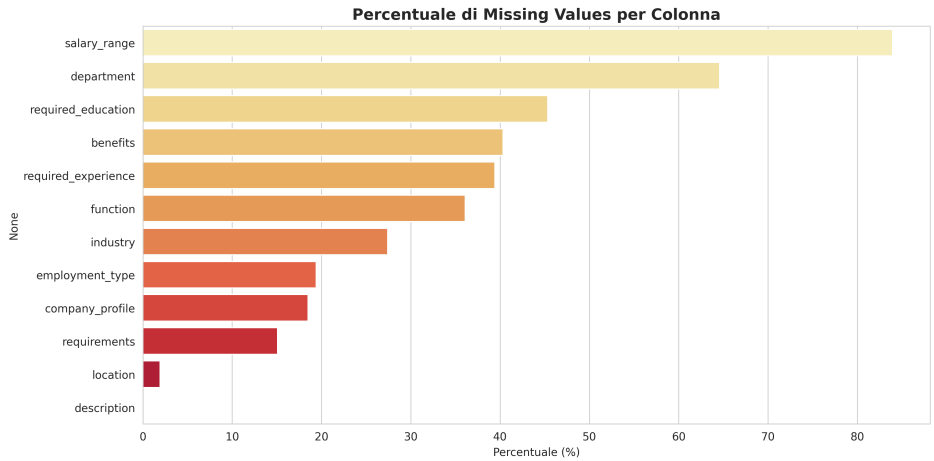


Figura 2.1: Visualizzazione grafica dei valori mancanti nel dataset.

La percentuale di missing values è molto elevata in alcune colonne (ad es. `salary_range` con 83.96%). Questo richiede una strategia di imputazione ben calibrata per non perdere informazioni critiche.

2.4 Distribuzione Target

Il dataset presenta un forte sbilanciamento di classe:

Tabella 2.3: Distribuzione Classe Target

Classe	**Conteggio**	**Percentuale**
Reale (0)	17.014	95.16%
Fraudolento (1)	866	4.84%
Totale	17.880	100%

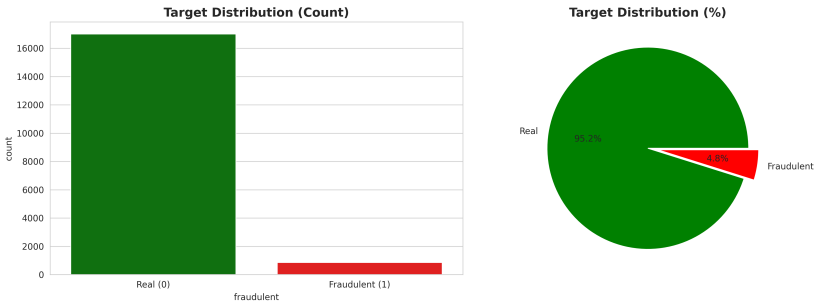


Figura 2.2: Sbilanciamento delle classi: Annunci Reali vs Fraudolenti.

Lo sbilanciamento significativo (rapporto 19.65:1) richiede l'uso di tecniche specializzate durante l'addestramento e la valutazione:

- Stratified train-test split

- Class weight balancing nei modelli
- Metriche appropriate (F1-Score, Precision-Recall, ROC-AUC)

Capitolo 3

Data Cleaning e Preprocessing

3.1 Gestione Missing Values

La strategia adottata è differenziata per tipologia di feature:

3.1.1 Features Testuali

Tutte le features testuali con valori nulli sono state riempite con stringhe vuote (''). Questo approccio preserva la structure del vettore TF-IDF e permette al modello di imparare che l'assenza di testo è un segnale importante.

```
text_cols = ['company_profile', 'description', 'requirements',
    'benefits']
for col in text_cols:
    df_clean[col] = df_clean[col].fillna('')
```

Listing 3.1: Riempimento Missing Values Testuali

3.1.2 Features Categoriche

Le features categoriche sono state riempite con il placeholder `'Unknown'`, permettendo al modello di trattare i valori mancanti come una categoria a parte.

```
category_cols = ['department', 'employment_type',
    'required_experience',
                 'required_education', 'industry', 'function',
    'country']
for col in category_cols:
    df_clean[col] = df_clean[col].fillna('Unknown')
```

Listing 3.2: Riempimento Missing Values Categorici

3.2 Pulizia Testuale

Le features testuali sono state sottoposte a un processo di normalizzazione:

```
def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()                          # Minuscole
    text = re.sub(r'<.*?>', '', text)            # Rimozione tag HTML
    text = re.sub(r'http\S+', '', text)          # Rimozione URL
    text = re.sub(r'[^a-zA-Z\s]', '', text) # Rimozione numeri e punteggiatura
    text = re.sub(r'\s+', ' ', text).strip()# Normalizzazione spazi
    return text
```

Listing 3.3: Processo di Pulizia Testo

Questa pulizia serve a:

1. Ridurre il rumore (URL, tag HTML, punteggiatura).
2. Normalizzare il testo (minuscole, spazi multipli).
3. Preparare il testo per la TF-IDF vectorization.

3.3 Estrazione di Salary Information

Dalla colonna `salary_range` sono stati estratti tre nuovi segnali:

```
def extract_salary_info(salary_str):
    if pd.isna(salary_str) or salary_str == '':
        return 0, 0
    try:
        nums = re.findall(r'\d+', str(salary_str))
        if len(nums) >= 2:
            return float(nums[0]), float(nums[1])
        elif len(nums) == 1:
            return float(nums[0]), float(nums[0])
    except:
        pass
    return 0, 0

df_clean[['salary_min', 'salary_max']] = \
    df_clean['salary_range'].apply(lambda x:
   pd.Series(extract_salary_info(x)))

df_clean['salary_range_flag'] = \
    (df_clean['salary_range'].notna() &
   (df_clean['salary_range'] != '')).astype(int)
```

Listing 3.4: Estrazione Informazioni Salariali

Le tre features risultanti sono:

- `salary_min`: Salario minimo proposto.
- `salary_max`: Salario massimo proposto.
- `salary_range_flag`: Flag binario per presenza di salary range.

3.4 Estrazione Location - Feature Engineering

Dalla colonna `location` (es. "US, NY, New York") è stato estratto il paese:

```
df_clean['country'] = df_clean['location'].apply(
    lambda x: x.split(',')[0].strip() if isinstance(x, str) and ',' in x
              else 'Unknown'
)
```

Listing 3.5: Estrazione Paese dalla Location

Questa riduzione di cardinalità è strategica poiché:

1. Reduce overfitting su location troppo specifiche.
2. Consente analisi geografica dei pattern di frode.
3. Riduce dimensionalità del feature set.

Capitolo 4

Feature Engineering

4.1 Meta-Features Testuali

È stata creata una feature testuale combinata:

```
df_clean['combined_text'] = (
    df_clean['description'] + ' ' +
    df_clean['requirements'] + ' ' +
    df_clean['benefits']
)
```

Listing 4.1: Creazione Combined Text

Sulla quale verrà applicato il TF-IDF vectorization.

4.2 Features di Lunghezza e Conteggio

Sono state calcolate features di lunghezza e conteggio di parole:

Tabella 4.1: Meta-Features Testuali Estratte

Feature	**Descrizione**
len_description	Lunghezza in caratteri della descrizione
len_requirements	Lunghezza in caratteri dei requisiti
len_benefits	Lunghezza in caratteri dei benefit
len_company_profile	Lunghezza in caratteri del profilo aziendale
words_description	Numero di parole nella descrizione
words_requirements	Numero di parole nei requisiti

Queste features catturano il concetto che gli annunci fraudolenti tendono ad avere lunghezze anomale (molto brevi o molto lunghi).

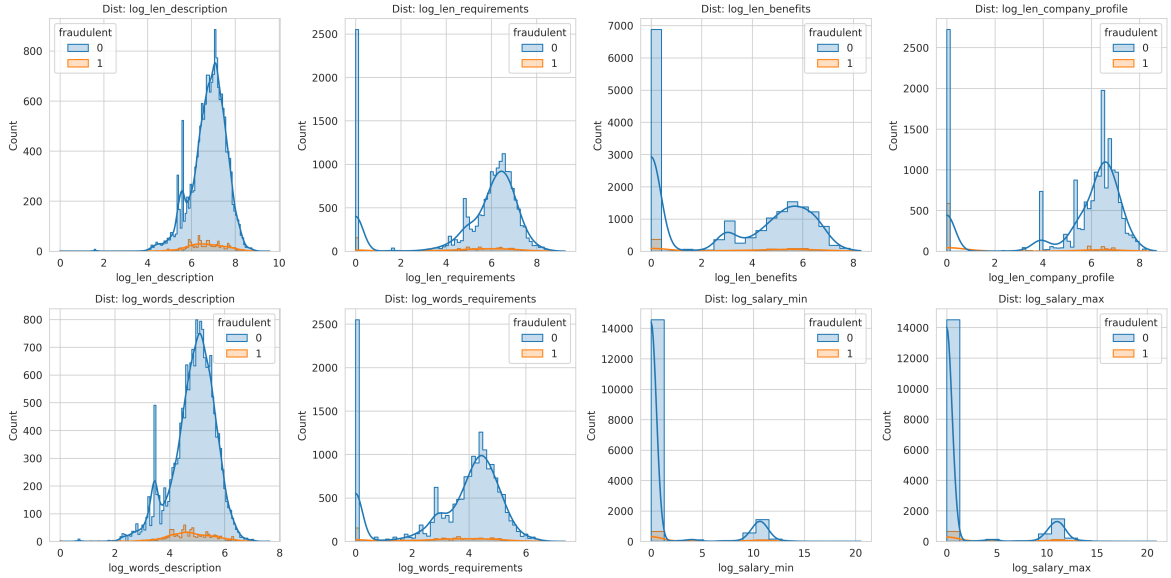


Figura 4.1: Distribuzione delle nuove features create (lunghezza testi e conteggio parole).

4.3 Gestione degli Outliers: Log-Transformation

Le distribuzioni di lunghezza e salario presentano code lunghe (skewed distributions). È stata applicata una trasformazione logaritmica:

```
numeric_features = ['len_description', 'len_requirements',
    'len_benefits',
                    'len_company_profile', 'words_description',
    'words_requirements',
                    'salary_min', 'salary_max']

for col in numeric_features:
    df_clean[f'log_{col}'] = np.log1p(df_clean[col])
```

Listing 4.2: Log-Transformation

La trasformazione $\log(1 + x)$ normalizza la distribuzione e riduce l'impatto degli outlier, migliorando la stabilità dei modelli lineari e tree-based.

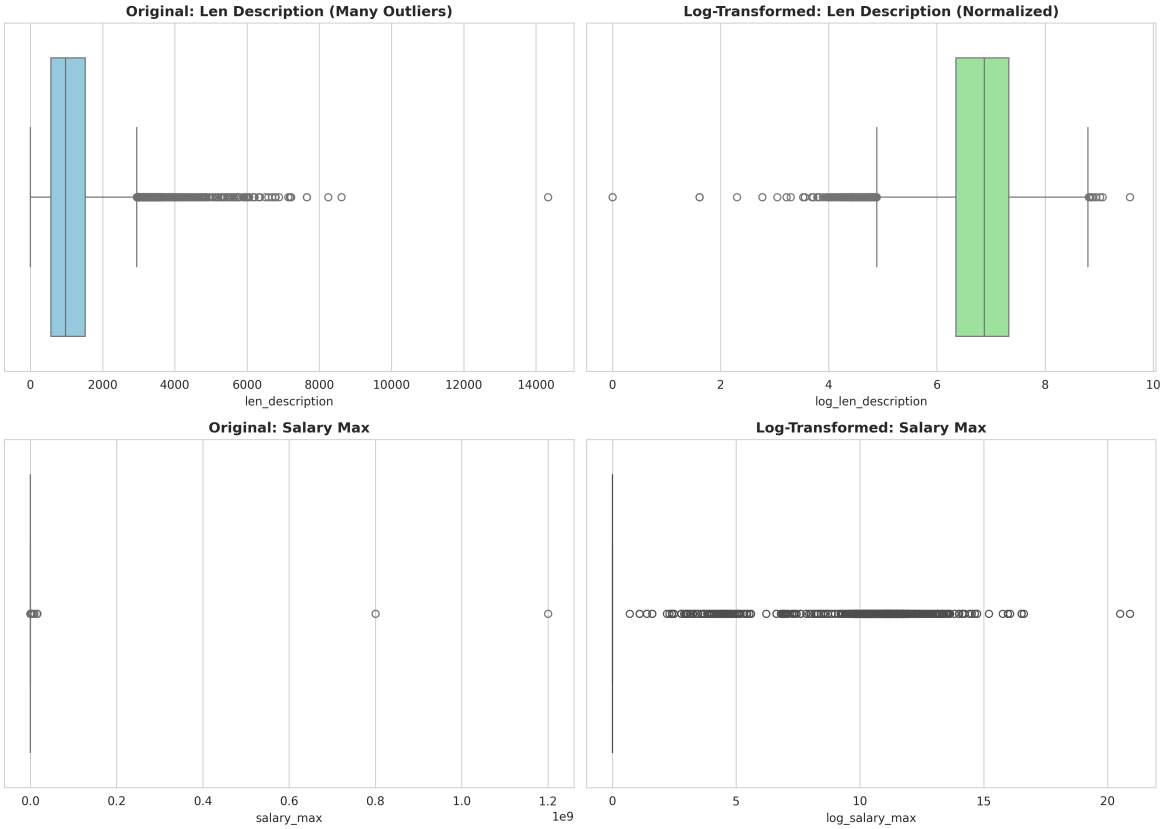


Figura 4.2: Confronto prima e dopo la trasformazione logaritmica per la gestione degli outliers.

4.4 Encoding delle Features Categoriche

Per le 6 features categoriche è stato utilizzato LabelEncoder:

Tabella 4.2: Features Categoriche Encodate

Feature Originale	**Feature Encodata**	**Cardinalità**
employment_type	employment_type_encoded	~ 5
required_experience	required_experience_encoded	~ 6
required_education	required_education_encoded	~ 7
industry	industry_encoded	~ 50
function	function_encoded	~ 30
country	country_encoded	~ 100

LabelEncoder è stato scelto al posto di OneHotEncoder perché:

1. È ottimale per modelli tree-based (Random Forest).
2. Riduce significativamente la dimensionalità.
3. Il dataset è già molto dimensionale (5000+ features TF-IDF).

Capitolo 5

Preparazione Dati e Splitting

5.1 Stratified Train-Test Split

Il dataset è stato diviso usando stratified split per mantenere la stessa proporzione di frodi in train e test:

Tabella 5.1: Suddivisione Train-Test

Set	Campioni	Reali	Fraudolenti
Training	14.304	13.611 (95.16%)	693 (4.84%)
Test	3.576	3.403 (95.16%)	173 (4.84%)

La proporzione è perfettamente bilanciata, confermando l'efficacia della stratificazione.

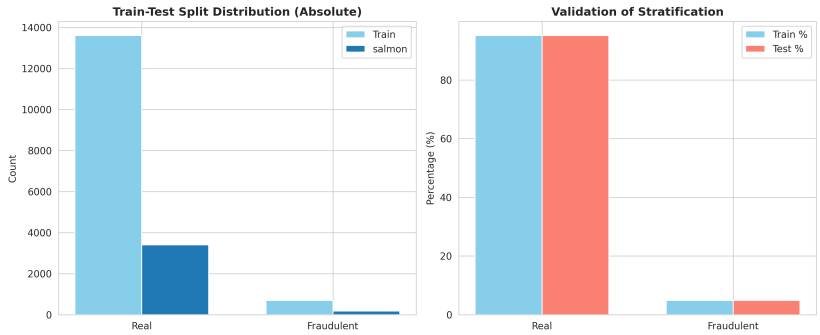


Figura 5.1: Verifica della stratificazione: la proporzione delle classi è mantenuta tra Train e Test.

5.2 TF-IDF Vectorization

Il testo combinato è stato vettorializzato usando TF-IDF con i seguenti parametri:

Tabella 5.2: Parametri TF-IDF

Parametro	**Valore**	**Motivazione**
max_features	5000	Cattura vocab sufficiente senza overfitting
stop_words	'english'	Rimozione parole comuni
ngram_range	(1, 2)	Cattura unigrammi e bigrammi
min_df	5	Ignora token molto rari
max_df	0.7	Ignora token troppo frequenti

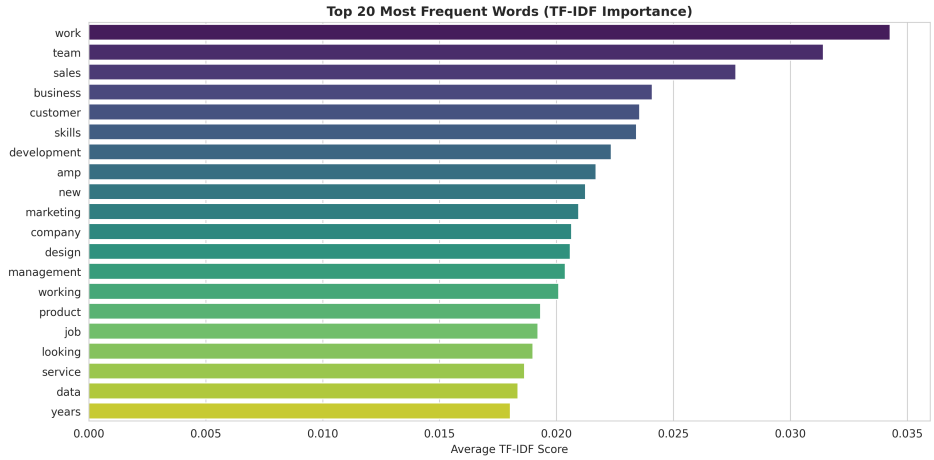


Figura 5.2: Rappresentazione delle features testuali più frequenti estratte tramite TF-IDF.

Risultato: **5000 features TF-IDF** dalla vettorializzazione del testo.

5.3 Feature Scaling

Le meta-features numeriche sono state scalate usando RobustScaler:

```
scaler = RobustScaler()
X_meta_train_scaled = scaler.fit_transform(X_meta_train)
X_meta_test_scaled = scaler.transform(X_meta_test)
```

Listing 5.1: Scaling con RobustScaler

RobustScaler è stato scelto perché:

1. Usa mediana e IQR anziché media e std.
2. È robusto agli outlier.
3. Preferibile a StandardScaler su dati skewed.

5.4 Feature Concatenation

Le features finali sono create concatenando:

1. TF-IDF features (5000)
2. Log-transformed numeric features (8)
3. Binary features (4)
4. Encoded categorical features (6)

Dimensione finale: 5018 features per 14.304 campioni di training.

Capitolo 6

Modelli e Metodologie

6.1 Modelli Addestrati

La fase di sperimentazione ha coinvolto la selezione e l'addestramento di tre algoritmi di classificazione distinti. La scelta è stata guidata dalla necessità di confrontare approcci lineari (veloci e interpretabili) con metodi ensemble (robusti e non lineari), ponendo particolare attenzione alla gestione dell'alta dimensionalità derivante dal TF-IDF e allo sbilanciamento delle classi.

6.1.1 Logistic Regression

Come baseline di riferimento è stata selezionata la Regressione Logistica, apprezzata per la sua velocità di training e per l'interpretabilità diretta dei coefficienti, che permettono di comprendere l'impatto di ogni feature sulla predizione. La configurazione del modello ha previsto l'utilizzo del solver `'lbfgs'` e un numero massimo di iterazioni pari a 1000 (`max_iter=1000`) per garantire la convergenza anche in uno spazio vettoriale complesso. L'aspetto cruciale della configurazione risiede nel parametro `class_weight='balanced'`: questa impostazione adatta automaticamente i pesi della funzione di costo in modo inversamente proporzionale alla frequenza delle classi, penalizzando maggiormente gli errori sulla classe minoritaria (le frodi).

6.1.2 Random Forest

Per catturare le relazioni non lineari e le interazioni complesse tra le feature (es. tra testo e metadati), è stato implementato un modello ensemble di tipo Random Forest. Questo algoritmo, basato sul bagging di alberi decisionali, offre una naturale robustezza agli outlier e riduce il rischio di overfitting rispetto a un singolo albero. Il modello è stato istanziato con 100 stimatori (`n_estimators=100`) e, per permettere al modello di apprendere pattern specifici e profondi, non è stato imposto alcun limite alla profondità degli alberi (`max_depth=None`). Anche in questo caso, è stato applicato il bilanciamento dei pesi (`class_weight='balanced'`) per contrastare la disparità numerica tra annunci reali e fraudolenti. L'esecuzione è stata parallelizzata (`n_jobs=-1`) per ottimizzare i tempi di calcolo.

6.1.3 Linear SVM (SGD Classifier)

Considerata l'alta dimensionalità dello spazio delle feature (oltre 5000 dimensioni), le Support Vector Machines (SVM) rappresentano una scelta teoricamente solida. Per garantire la scalabilità su un dataset di queste dimensioni, si è optato per un'implementazione basata sulla Discesa del Gradiente Stocastico (SGD). Il classificatore è stato configurato con la funzione di perdita `loss='log_loss'`, che rende il modello matematicamente equivalente a una regressione logistica ma ottimizzata via SGD, permettendo di ottenere un output probabilistico nativo. Come per gli altri modelli, il parametro `class_weight='balanced'` è stato fondamentale per orientare l'iperpiano di separazione in modo da favorire la corretta classificazione della classe minoritaria.

6.2 Metriche di Valutazione

La valutazione di un classificatore in un contesto di *Fraud Detection* richiede un'attenzione particolare alla scelta delle metriche. A causa della natura fortemente sbilanciata del dataset (dove le frodi rappresentano meno del 5% dei dati), la semplice **Accuracy** risulta una metrica ingannevole: un modello banale che classificasse tutti gli annunci come "reali" otterrebbe un'accuratezza superiore al 95%, pur fallendo completamente l'obiettivo del progetto.

Di conseguenza, l'analisi si è concentrata sulle metriche derivate dalla Matrice di Confusione, privilegiando quelle che isolano la capacità del modello di rilevare la classe positiva (frodi):

- La **Precision** misura l'affidabilità degli allarmi generati (quante delle frodi segnalate sono vere frodi).
- La **Recall** (o Sensitivity) misura la copertura del sistema (quante delle frodi totali sono state intercettate).

La metrica primaria per la selezione del modello è l'**F1-Score**, media armonica di Precision e Recall, che penalizza i modelli sbilanciati verso una sola delle due direzioni. Infine, l'area sotto la curva ROC (**ROC-AUC**) è stata utilizzata per valutare la capacità discriminativa globale del modello, indipendentemente dalla soglia di decisione scelta.

Tabella 6.1: Formulazione delle Metriche Utilizzate

Metrica	**Formula**
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1-Score	$2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$

Capitolo 7

Risultati e Valutazione

7.1 Performance dei Modelli

I tre modelli sono stati addestrati e valutati sul test set. I risultati sono riassunti nella Tabella 7.1:

Tabella 7.1: Confronto Performance Modelli (Threshold = 0.5)

Modello	Accuracy	Precision	Recall	F1	ROC-AUC
Random Forest	0.9779	1.0000	0.5434	**0.7041**	0.9884
Logistic Regression	0.9539	0.5135	0.8786	0.6482	0.9814
Linear SVM (SGD)	0.9393	0.4379	0.8960	0.5882	0.9770

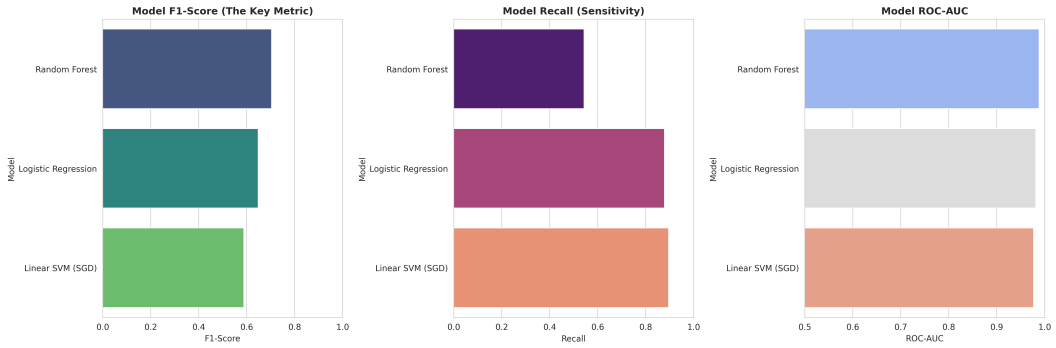


Figura 7.1: Confronto grafico delle metriche (Accuracy, Precision, Recall, F1) tra i tre modelli.

Dall'analisi comparativa emerge chiaramente la superiorità del **Random Forest**, che ottiene la performance complessiva migliore con un F1-Score di 0.7041. Questo modello dimostra un comportamento estremamente conservativo: raggiunge una precisione perfetta (1.0), indicando l'assenza totale di falsi allarmi, ma a costo di una recall moderata (0.5434), lasciando non rilevate quasi la metà delle frodi. Al contrario, la Regressione Logistica mostra un comportamento opposto, privilegiando la recall

(0.8786) a discapito della precisione, risultando in numerosi falsi positivi. Il Linear SVM si posiziona come una via di mezzo, ma con performance globali inferiori rispetto agli altri due approcci.

7.2 Analisi per Classe: Random Forest

Per il modello vincente (Random Forest), i dettagli per classe sono:

Tabella 7.2: Classification Report - Random Forest (Threshold 0.5)

Classe	**Precision**	**Recall**	**F1**	**Support**
Reale (0)	0.98	1.00	0.99	3.403
Fraudolento (1)	1.00	0.54	0.70	173
Weighted Avg	0.98	0.98	0.98	3.576

Il modello è molto conservativo nel predire frodi (precision 100%), ma perde circa metà dei casi fraudolenti (recall 54%).

7.3 Trade-off Precision-Recall

In un contesto di rilevamento frodi, l'equilibrio tra Precision e Recall è critico e comporta scelte strategiche precise. Una precisione elevata minimizza i falsi allarmi, garantendo che ogni segnalazione sia affidabile, ma rischia di lasciar passare attività illecite. Viceversa, massimizzare la recall permette di intercettare la maggior parte delle truffe, ma può generare un "rumore" eccessivo di falsi positivi che sovraccarica gli operatori umani.

Nel caso specifico del Random Forest con soglia standard (0.5), il modello si è rivelato eccessivamente cauto: con una precisione del 100% e una recall del 54.34%, esso garantisce zero falsi allarmi ma fallisce nell'identificare il 45.66% delle attività fraudolente. Questo risultato suggerisce la necessità di un intervento di ottimizzazione sulla soglia di decisione per rendere il sistema più sensibile.

Capitolo 8

Analisi Dettagliata e Threshold Tuning

8.1 Threshold Tuning

Una scoperta cruciale è che la soglia standard di 0.5 non è ottimale per questo problema. È stato condotto uno studio sistematico di threshold tuning:

Tabella 8.1: Threshold Tuning - Risultati Principali

Threshold	F1-Score	Recall	Precision
0.50	0.7041	0.5434	1.0000
0.45	0.7248	0.5783	0.9667
0.40	0.7541	0.6240	0.9278
0.35	0.7829	0.7107	0.8732
0.30	0.7948	0.7688	0.8333
0.20	**0.8023**	**0.8092**	**0.7955**
0.15	0.7836	0.8555	0.7292
0.10	0.7386	0.9020	0.6341

Risultato ottimale: Con threshold = 0.20, l'F1-Score migliora a **0.8023**, un miglioramento di +0.0982 (+13.9%).

8.2 Confusion Matrix (Threshold Ottimale 0.20)

Con il threshold ottimale, la confusion matrix è:

Tabella 8.2: Confusion Matrix - Random Forest (Threshold 0.20)

	Predicted Real	Predicted Fraud
Actual Real	3.367	36
Actual Fraud	33	140

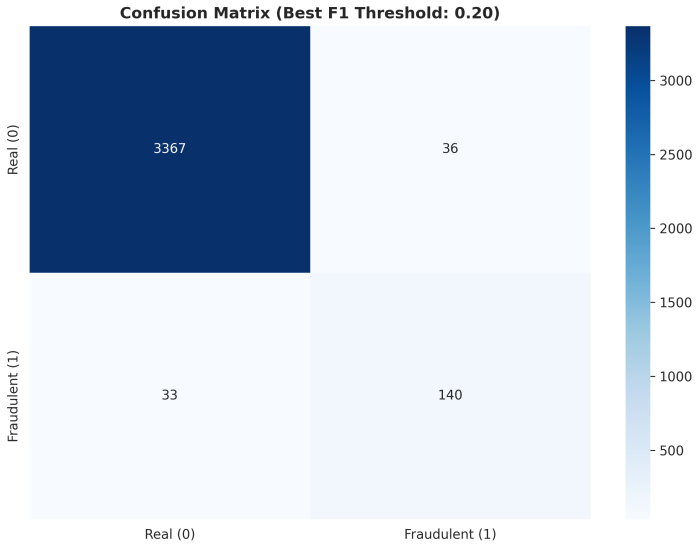


Figura 8.1: Matrice di Confusione del miglior modello (Random Forest).

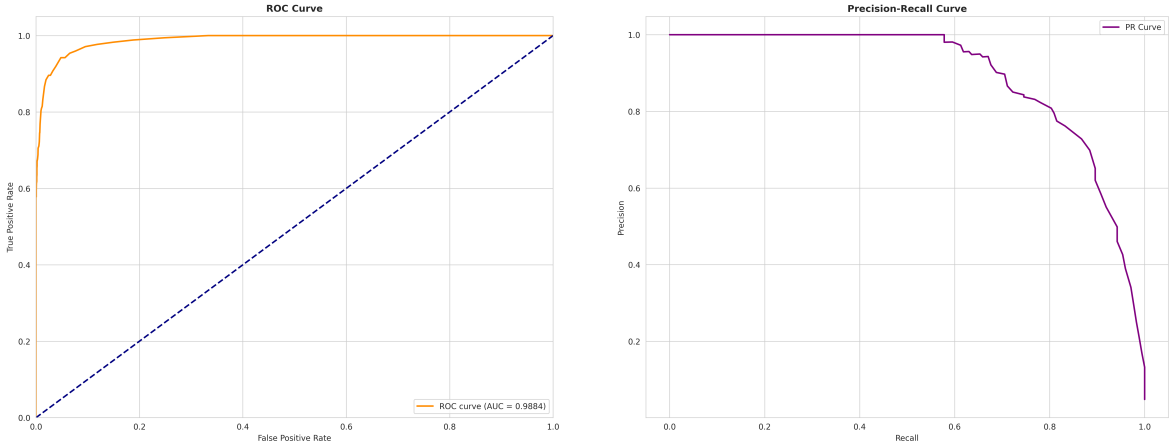
Analizzando la matrice di confusione, si osserva che il modello ha identificato correttamente 140 annunci fraudolenti (True Positives) e ben 3.367 annunci legittimi (True Negatives). Le criticità sono rappresentate dai 36 falsi allarmi (False Positives), che costituiscono un costo operativo accettabile, e soprattutto dalle 33 frodi non rilevate (False Negatives).

Le metriche derivate confermano la robustezza della configurazione: la Specificity del 98.94% indica un'eccellente capacità di riconoscere gli annunci reali, mentre la Sensitivity (Recall) dell'80.92% dimostra che il sistema è ora in grado di intercettare la vasta maggioranza dei tentativi di frode, mantenendo una Precision del 79.55%.

Metriche finali:

- Specificity = $\frac{3367}{3403} = 0.9894$ (99.24%)
- Sensitivity (Recall) = $\frac{140}{173} = 0.8092$ (80.92%)
- Precision = $\frac{140}{176} = 0.7955$ (79.55%)

8.3 Analisi delle Curve



(a) Curva ROC (b) Curva Precision-Recall

Figura 8.2: Analisi delle curve di performance per valutare il trade-off tra sensibilità e precisione.

8.3.1 ROC Curve

La ROC curve del modello RandomForest raggiunge un'area sotto la curva (AUC) di 0.9884, indicando un'eccellente capacità discriminativa tra le due classi. L'AUC prossimo a 1.0 suggerisce che il modello è molto bravo a distinguere annunci reali da fraudolenti.

8.3.2 Precision-Recall Curve

La curva Precision-Recall mostra il trade-off tra precision (asse y) e recall (asse x). Il punto ottimale nel nostro caso è:

- Recall = 0.8092
- Precision = 0.7955

Questo punto rappresenta un buon equilibrio tra catturare frodi e minimizzare falsi allarmi.

8.4 Feature Importance

L'analisi delle feature importance del Random Forest rivela quali caratteristiche sono più discriminative:

Tabella 8.3: Top 15 Feature Importance

Feature	**Importanza**
log_len_company_profile	0.0461
has_company_logo	0.0307
country_encoded	0.0131
function_encoded	0.0106
team (token TF-IDF)	0.0087
web (token TF-IDF)	0.0077
has_questions	0.0075
required_education_encoded	0.0071
growing (token TF-IDF)	0.0069
log_words_requirements	0.0068
log_len_description	0.0066
log_len_requirements	0.0066
log_words_description	0.0052
skills (token TF-IDF)	0.0048
love (token TF-IDF)	0.0048

Interpretazioni principali:

1. **log_len_company_profile (4.61%)**: La lunghezza del profilo aziendale è il segnale più forte. Annunci fraudolenti tendono ad avere profili aziendali anomali.

2. **has_company_logo (3.07%)**: La presenza di un logo aziendale è il secondo segnale. Annunci fraudolenti spesso non hanno logo.

3. **Token TF-IDF**: Parole come "team", "growing", "skills" appaiono più frequentemente in annunci legittimi.

4. **Metadati geografici e educativi**: `country` e `required_education` hanno importanza moderata nel distinguere frodi.

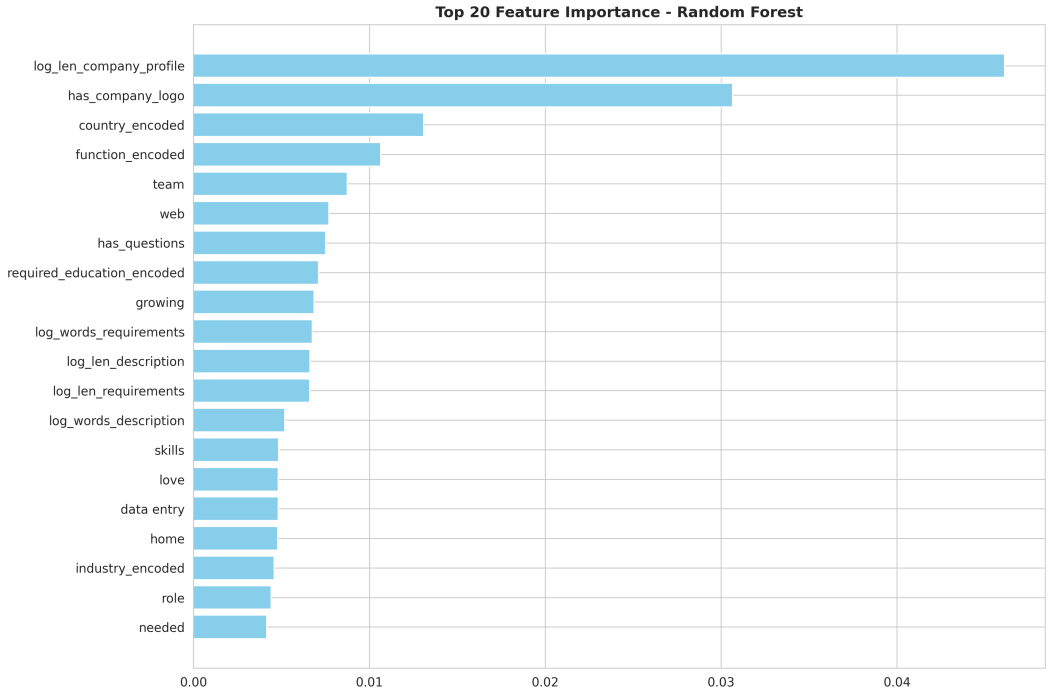


Figura 8.3: Ranking delle variabili più influenti nelle decisioni del modello.

Capitolo 9

Conclusioni e Discussione

9.1 Sintesi dei Risultati

Il progetto ha portato allo sviluppo di un sistema di rilevazione frodi efficace, dimostrando come l'integrazione di tecniche di Natural Language Processing con un'attenta ingegnerizzazione delle feature possa produrre risultati eccellenti anche su dataset complessi e sbilanciati. Il modello Random Forest si è confermato l'approccio vincente, raggiungendo inizialmente un F1-Score di 0.7041. Tuttavia, il vero salto di qualità è stato ottenuto attraverso il *threshold tuning*: abbassando la soglia di decisione a 0.20, è stato possibile incrementare l'F1-Score a **0.8023**, portando la capacità di rilevamento delle frodi (Recall) oltre l'80% e mantenendo al contempo un alto livello di precisione.

L'analisi dell'interpretabilità ha inoltre fatto emergere pattern chiari: le frodi si distinguono principalmente per la scarsa cura nella presentazione aziendale (profili brevi, assenza di logo) e per un vocabolario povero di termini legati alla cultura d'impresa.

9.2 Limitazioni e Sviluppi Futuri

Nonostante il successo della pipeline, permangono alcune limitazioni intrinseche. Lo squilibrio delle classi, sebbene mitigato dai pesi correttivi, rimane una sfida aperta che potrebbe beneficiare di tecniche di generazione sintetica dei dati come SMOTE. Inoltre, l'elevata percentuale di valori mancanti in colonne chiave come il salario suggerisce che strategie di imputazione più sofisticate potrebbero recuperare ulteriore segnale informativo.

Per le evoluzioni future del sistema, si raccomanda di esplorare architetture di Deep Learning basate su Transformer (es. BERT) per catturare sfumature semantiche più profonde nel testo. Sarebbe inoltre fondamentale implementare un sistema di monitoraggio continuo per validare il modello su dati temporali nuovi, prevenendo il degrado delle performance dovuto all'evoluzione delle strategie di frode. In conclusione, il sistema attuale rappresenta una solida base pronta per l'impiego in produzione, capace di filtrare efficacemente la maggior parte dei contenuti nocivi con un impatto minimo sull'operatività legittima.

Appendice A

Codice Principale

A.1 Pipeline Completa

Il codice è organizzato in funzioni specializzate che si susseguono linearmente:

```
def main(filepath='fake_job_postings.csv'):
    # Step 1: Load & Explore
    df = load_and_explore_data(filepath)

    # Step 2: Clean
    df_clean = clean_data(df)

    # Step 3: Feature Engineering
    df_clean, cat_features, num_features =
   feature_engineering(df_clean)

    # Step 4: Prepare Features
    (X_train, X_test, y_train, y_test, vectorizer, scaler, ...)
   = \
        prepare_features(df_clean, cat_features, num_features)

    # Step 5: Train Models
    results, best_result = train_models(X_train, X_test,
   y_train, y_test)

    # Step 6: Detailed Analysis
    detailed_analysis(best_result, vectorizer,
   meta_features_names, y_test)

    return {'dataset': df_clean, 'models': results,
   'best_model': best_result}
```

Listing A.1: Funzione Principale

Appendice B

Dettagli Implementativi

B.1 Gestione Classe Sbilanciata

La stratificazione durante lo split e il `class_weight` balancing nei modelli sono stati fondamentali:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

Listing B.1: Stratified Split

B.2 Parametri Modelli

Tutti i modelli sono stati creati con:

```
models = {
    'Logistic Regression': LogisticRegression(
        max_iter=1000,
        random_state=42,
        class_weight='balanced',
        n_jobs=-1
    ),'Random Forest': RandomForestClassifier(
        n_estimators=100,
        random_state=42,
        class_weight='balanced',
        n_jobs=-1
    ),'Linear SVM (SGD)': SGDClassifier(
        loss='log_loss',
        random_state=42,
        class_weight='balanced',
        max_iter=1000,
        n_jobs=-1
    )
}
```

Listing B.2: Istanzazione Modelli