Rilevamento di URL di Phishing tramite Tecniche di Dimensionality Reduction e Apprendimento Supervisionato

Salvucci Franco - 0368692

2 febbraio 2026

Sommario

Il presente studio analizza l'efficacia di diversi paradigmi di **Machine Learning** nella classificazione di URL malevoli, utilizzando il dataset ad alte prestazioni **Web Page Phishing Dataset**.
La ricerca affronta la sfida della sicurezza informatica moderna attraverso un confronto sistematico tra modelli basati su iperpiani di separazione e architetture ensemble, operando su uno spazio vettoriale composto da **89 feature estratte** (caratteristiche lessicali, statistiche e comportamentali degli URL).

Indice

1 Introduzione **4**
1.1 Metodologia e Pre-processing 4
1.2 Classificatori a Confronto 4
1.3 Baseline 4

2 Analisi delle feature **5**
2.1 Feature Strutturali dell'URL 5
2.2 Feature del Dominio e Sottodomini 6
2.3 Feature Lessicali (Parole nell'URL) 6
2.4 Feature del Contenuto della Pagina (HTML/JS) 6
2.5 Feature Esterne e di Reputazione 7
2.6 Target 8

3 Normalizzazione **9**
3.1 Analisi delle distribuzioni e differenze di scala nelle feature 9

4 Training & Evaluation **10**
4.1 Definizioni di Precision, Recall e F1-Score 11
4.1.1 Confusion Matrix 11
4.1.2 Metriche di Valutazione 11
4.2 Analisi decisionale: Falsi Positivi vs Falsi Negativi 12

5 Modelli **13**
5.1 Baseline: LogisticRegression 13
5.2 SVM 16
5.3 Random Forest 20

6 Analisi delle feature "importanti" **23**
6.1 Cos'è la Permutation Importance? 23
6.1.1 Il meccanismo logico 23
6.1.2 Vantaggi rispetto alla Feature Importance standard 23
6.2 L'Esperimento: Stress-test degli SVM 24

7 Conclusioni Finali: Confronto tra i Modelli **27**
7.1 Tabella Riassuntiva delle Performance (Test Set) 27
7.2 Analisi Comparativa 27
7.2.1 A. La Forza della Baseline (Logistic Regression) 27
7.2.2 B. Il Salto di Qualità (SVM e Random Forest) 27
7.3 Validazione tramite Stress Test (Ablazione Feature) 28
7.4 Considerazioni sulla Sicurezza: La Recall 28
7.5 Verdetto Finale 28

A Appendice: Spiegazione `StandardScaler` di `scikit-learn` 29
A.1 Definizione Matematica 29
A.1.1 La Trasformazione (Z-score) 29
A.2 Perché usarla nel progetto? 30
A.2.1 A. Il predominio delle "Magnitudo" (Bias di Scala) 30
A.2.2 B. Requisito Fondamentale per la PCA 30
A.3 Osservazione: Perché non il Min-Max Scaler? 30

B Appendice: Implementazioni funzioni di appoggio 31
B.1 Ottimizzazione degli Iperparametri 31
B.2 Valutazione delle Performance (Training Set) 32
B.3 Valutazione Finale (Test Set) 32

1 Introduzione

1.1 Metodologia e Pre-processing

Data la complessità e la multidimensionalità del dataset, il workflow implementato non si limita all'addestramento diretto, ma prevede una fase critica di ottimizzazione del dato:

1. **Normalizzazione:** per garantire che l'ampiezza delle scale delle 89 feature non influenzi negativamente i gradienti dei modelli.
2. **Scaling:** per garantire che tutte le feature convergano ad una Gaussiana Standard, ovvero $\mathcal{N}(0, 1)$.
3. **Principal Component Analysis (PCA):** utilizzata per ridurre la dimensionalità dello spazio delle feature, eliminando la ridondanza informativa e concentrando la varianza del dataset in un set ottimizzato di componenti principali.

1.2 Classificatori a Confronto

Il task di classificazione binaria viene risolto attraverso due approcci algoritmici distinti:

- **Support Vector Machines (SVM):** esplorate nelle varianti con **Kernel Lineare**, **Polinomiale (Poly)** e **Radial Basis Function (RBF)**, per testare la capacità del modello di mappare i dati in spazi a dimensionalità superiore.
- **Metodi Ensemble:** implementati per massimizzare la robustezza predittiva tramite strategie di **Bagging** (**Random Forest**) e **Boosting** (**AdaBoost** e **Gradient Boosting**). Questi modelli sono stati scelti per la loro intrinseca capacità di gestire relazioni non lineari e per la resistenza all'overfitting rispetto ai singoli alberi di decisione.

1.3 Baseline

Come Baseline, sono stati scelti due modelli:

1. **DummyClassifier**: la baseline più semplice fra tutte; questa fornirà sempre un'accuratezza del 50% (pari a 0.5) in caso di dataset bilanciato.
2. **LogisticRegression**: modello lineare più semplice, che servirà come base di confronto reale per i modelli più complessi.

2 Analisi delle feature

Di seguito riportiamo una breve descrizione di ogni feature presente nel dataset.
Osservazione: non tutte le descrizioni erano presenti nel file del dataset originale; alcune di esse sono frutto di un'analisi deduttiva e pertanto rappresentano l'interpretazione logica della variabile nel contesto del phishing.

2.1 Feature Strutturali dell'URL

Queste variabili analizzano la composizione testuale dell'indirizzo web.

- `url`: L'indirizzo URL completo analizzato.
- `length_url` / `length_hostname`: Lunghezza totale dell'URL e del solo nome dell'host.
- `ip`: Variabile binaria; indica se nell'URL è presente un indirizzo IP al posto del nome a dominio (tecnica spesso usata nel phishing per bypassare i controlli sui nomi).
- `nb_dots / nb_hyphens / nb_at / nb_qm / nb_and / nb_or / nb_eq / nb_underscore / nb_tilde / nb_percent / nb_slash / nb_star / nb_colon / nb_comma / nb_semicolumn / nb_dollar / nb_space`: Conteggio di caratteri speciali (punti, trattini, chiocciole, punti interrogativi, ecc.) presenti nell'URL.
- `nb_www / nb_com / nb_dslash`: Conteggio delle stringhe "www", ".com" e del doppio slash "//" all'interno del percorso.
- `http_in_path`: Presenza della stringa "http" all'interno del percorso dell'URL (tecnica per mascherare URL malevoli all'interno di parametri).
- `https_token`: Indica se il token "https" è presente nella parte dell'host (fuori dal protocollo standard).
- `ratio_digits_url / ratio_digits_host`: Rapporto tra caratteri numerici e lunghezza totale rispettivamente dell'URL e dell'host.
- `punycode`: Indica se l'URL utilizza la codifica Punycode per gestire caratteri speciali (spesso usata per attacchi omografi, es. domini con accenti che sembrano caratteri latini).
- `port`: Indica se nell'URL è specificata una porta non standard (es. :8080).

2.2 Feature del Dominio e Sottodomini

- `tld_in_path / tld_in_subdomain`: Presenza di un TLD (es. .com, .net) all'interno del percorso o del sottodominio.
- `abnormal_subdomain`: Flag che indica se la struttura del sottodominio presenta anomalie sintattiche.
- `nb_subdomains`: Numero totale di sottodomini identificati.
- `prefix_suffix`: Presenza di trattini nel nome a dominio, spesso usati per simulare nomi di brand reali (es. `paypal-security.com`).
- `random_domain`: Indica se il nome a dominio sembra generato da algoritmi casuali (DGA).
- `shortening_service`: Indica se l'URL utilizza servizi di abbreviazione (es. bit.ly, t.co).

2.3 Feature Lessicali (Parole nell'URL)

- `length_words_raw`: Numero totale di parole identificate nell'URL tramite delimitatori.
- `shortest_words_raw / longest_words_raw`: Lunghezza della parola più corta e più lunga nell'intero URL.
- `shortest_word_host / longest_word_host`: Lunghezza della parola più corta e più lunga all'interno dell'host.
- `avg_words_raw / avg_word_host / avg_word_path`: Lunghezza media delle parole nell'URL, nell'host e nel percorso.
- `phish_hints`: Conteggio di parole "esca" tipicamente usate negli attacchi phishing (es. "login", "update", "banking", "secure").

2.4 Feature del Contenuto della Pagina (HTML/JS)

- `nb_hyperlinks`: Numero totale di link ipertestuali presenti nel corpo della pagina.
- `ratio_intHyperlinks / ratio_extHyperlinks / ratio_nullHyperlinks`: Percentuale di link che puntano allo stesso dominio, a domini esterni o che non hanno una destinazione valida (anchor vuote).
- `nb_extCSS`: Numero di fogli di stile (CSS) caricati da domini esterni.

- `ratio_intRedirection / ratio_extRedirection`: Rapporto di reindirizzamenti verso risorse interne ed esterne.
- `login_form`: Variabile binaria che indica la presenza di form per l'inserimento di credenziali.
- `external_favicon`: Indica se l'icona del sito (favicon) è ospitata su un dominio diverso da quello dell'URL.
- `links_in_tags`: Rapporto di link presenti nei tag meta, script e link rispetto al totale.
- `submit_email`: Indica se il form invia i dati inseriti direttamente a un indirizzo email (uso di `mailto:`).
- `ratio_intMedia / ratio_extMedia`: Rapporto di file multimediali (immagini, video) ospitati internamente o esternamente.
- `iframe / popup_window`: Presenza di tag `<iframe>` o script per l'apertura automatica di finestre popup.
- `safe_anchor`: Percentuale di ancore (`<a>`) che puntano a URL considerati sicuri o interni.
- `onmouseover / right_clic`: Presenza di script che tentano di nascondere l'URL reale nella barra di stato o che disabilitano il menu contestuale.
- `empty_title / domain_in_title`: Indica se il tag `<title>` è vuoto o se contiene il nome del dominio.

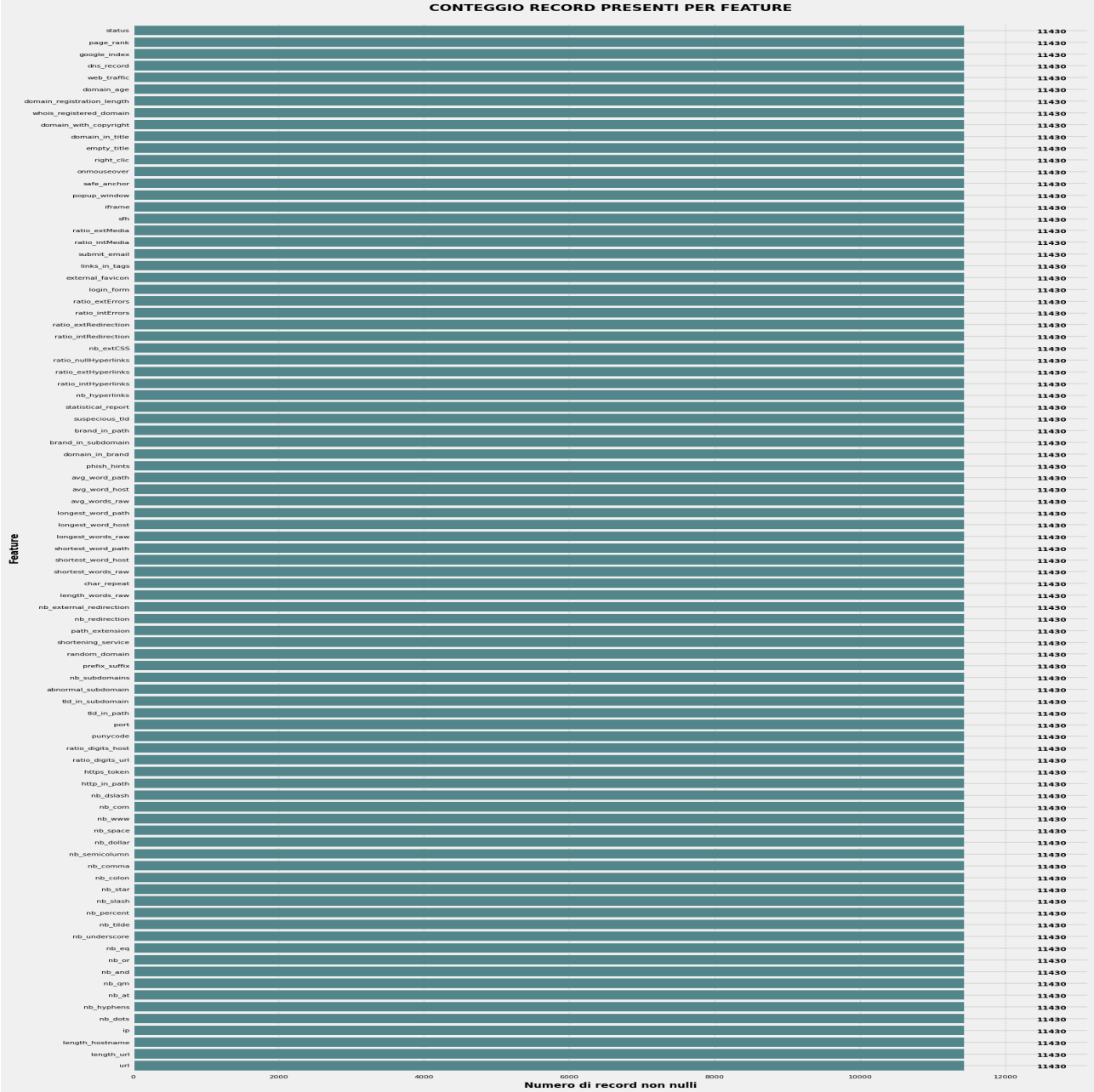
2.5 Feature Esterne e di Reputazione

- `whois_registered_domain`: Verifica se il dominio risulta regolarmente registrato nei database WHOIS.
- `domain_registration_length`: Durata complessiva prevista della registrazione del dominio.
- `domain_age`: Età del dominio calcolata in giorni dalla prima registrazione.
- `web_traffic`: Rilevanza del sito basata su metriche di traffico globale (es. Alexa Rank).
- `dns_record`: Presenza di record DNS validi e configurati correttamente.
- `google_index`: Verifica se la pagina è indicizzata nei motori di ricerca (Google).
- `page_rank`: Valore di autorevolezza della pagina secondo l'algoritmo PageRank.

2.6 Target

- `status`: Variabile target binaria; rappresenta la classe dell'URL: **legitimate** (sito sicuro) o **phishing** (sito malevolo).

Dall'immagine seguente, possiamo vedere la distribuzione dei record per le feature. Come vedremo, ogni record del dataset contiene tutte le 89 feature, ergo non ci sono feature mancanti nel dataset.



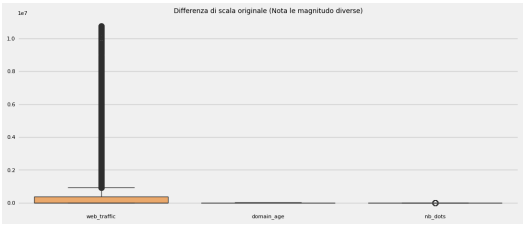
3 Normalizzazione

Prima di tutto notiamo che la feature `url` non risulta necessaria ai fini della trattazione del nostro problema; pertanto, si procede alla sua rimozione dal dataset.
Viene inoltre utilizzata una tecnica di encoding chiamata **LabelEncoder** della libreria `scikit-learn`, che permette di trasformare la colonna `status` (ovvero il vettore dei target y) in valori numerici binari (0/1).
La mappatura della feature `status` avviene nel seguente modo:

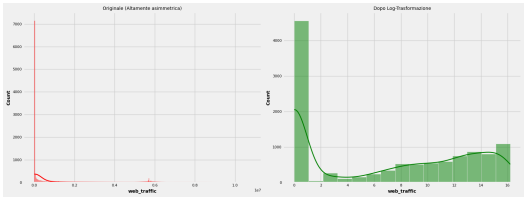
- `status = legitimate` $\rightarrow 0$
- `status = phishing` $\rightarrow 1$

3.1 Analisi delle distribuzioni e differenze di scala nelle feature

Osservando più nel dettaglio il dataset, si nota che alcune feature presentano una differenza di scala tra i valori estremamente elevata. Ad esempio, la feature `web_traffic` presenta un intervallo di valori che varia da 0 a circa 10.000.000.
Se questa differenza di scala non venisse opportunamente trattata, porterebbe a una gestione inefficiente delle feature durante la fase di addestramento, specialmente per i modelli **SVM** (Support Vector Machines), i quali sono particolarmente sensibili all'ordine di grandezza dei valori numerici in input.
Infatti, senza un corretto preprocessing, i modelli basati su iperpiani o distanze rischierebbero di considerare queste variabili come dominanti rispetto alle altre (l'effetto per cui una feature si "mangia" le altre), portando a risultati distorti e performance non ottimali.
Per ovviare a questo problema, la scelta è ricaduta sull'applicazione della **log-trasformazione** dei dati. Questa tecnica permette di comprimere i valori molto elevati e ridurre l'asimmetria delle distribuzioni senza causare la perdita di informazioni rilevanti contenute nel dataset.



(a) Differenza di scala



(b) Situazione post log-trasformazione

4 Training & Evaluation

Dopo aver analizzato, normalizzato e suddiviso il dataset (train/test split), si procede all'addestramento dei tre modelli descritti precedentemente.

Per ogni modello verrà effettuata una duplice analisi per evidenziare le differenze prestazionali:

1. Addestramento e valutazione sul dataset originale.
2. Addestramento e valutazione sul dataset ridotto tramite **PCA**.

Al termine della fase di training, verrà eseguito un confronto qualitativo basato sui seguenti parametri:

- Tempi di addestramento.
- Accuratezza della predizione.
- **F1-score**.
- **Precision** e **Recall**.
- Deviazione standard (σ).

Al fine di prevenire fenomeni di overfitting, è stata adottata la tecnica della **K-fold Cross-validation** con $K = 5$. Questa scelta è motivata dai seguenti vantaggi:

1. Garantisce la stabilità delle performance e minimizza il bias derivante dalla selezione del training set.
2. Il training set viene suddiviso in 5 sottogruppi (*fold*): ciclicamente, il modello viene addestrato su 4 di essi e validato sul restante.
3. I risultati finali rappresentano la media aritmetica delle prestazioni ottenute nelle 5 iterazioni, fornendo una stima robusta della capacità predittiva.

Per evitare il **Data Leakage** (ovvero l'iniezione involontaria di informazioni dal validation set nel training set), le operazioni di **scaling** e **PCA** sono state inserite all'interno di una `Pipeline` di `scikit-learn`, venendo così eseguite correttamente all'interno di ogni singolo fold.

4.1 Definizioni di Precision, Recall e F1-Score

La valutazione dei modelli si basa sul concetto di **Confusion Matrix**. Sia

$$\{(x_i, y_i)\}_{i=1}^{n}$$

il nostro dataset, dove:

- $y_i \in \{0, 1\}$ è il target reale dell'elemento i-esimo.
- $\hat{y}_i \in \{0, 1\}$ è il target **predetto** dal modello.

4.1.1 Confusion Matrix

La matrice di confusione è così definita:

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	**TN** (True Negatives)	**FP** (False Positives)
$y = 1$	**FN** (False Negatives)	**TP** (True Positives)

- **TP**: Istanze di phishing correttamente classificate (*Phishing bloccati*).
- **FP**: Istanze legittime classificate come phishing (*Falso Allarme*).
- **FN**: Istanze di phishing classificate come legittime (*Phishing mancati*).
- **TN**: Siti legittimi correttamente classificati (*Siti sicuri*).

4.1.2 Metriche di Valutazione

Precision: Rappresenta la frazione di istanze positive tra tutte quelle predette come tali.

$$\boxed{\text{Precision} = \frac{TP}{TP + FP} = Pr(y = 1 | \hat{y} = 1)}$$

Recall: Rappresenta la capacità del modello di individuare tutte le istanze positive reali.

$$\boxed{\text{Recall} = \frac{TP}{TP + FN} = Pr(\hat{y} = 1 | y = 1)}$$

F1-Score: Media armonica tra Precision e Recall, utile per bilanciare le due metriche.

$$\boxed{\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}}$$

4.2 Analisi decisionale: Falsi Positivi vs Falsi Negativi

Nel contesto del phishing, gli errori hanno impatti molto diversi:

Errore	**Significato**	**Conseguenza Reale**
False Positive (FP)	URL sicuro scambiato per phishing	Blocco/alert fastidioso
False Negative (FN)	URL malevolo scambiato per sicuro	**Compromissione sicurezza**

Il **Falso Negativo** è estremamente critico: un utente che clicca su un link malevolo rischia il furto di credenziali, l'installazione di malware e danni economici spesso irreversibili.
Al contrario, un **Falso Positivo** comporta solo un temporaneo calo di usabilità (l'utente può sbloccare il sito manualmente).
Data la natura del problema, l'obiettivo primario è la minimizzazione dei Falsi Negativi. Pertanto, la metrica cardine di questo studio è la **Recall** della classe Phishing. L'ordine di importanza stabilito è:

$$\boxed{\text{Recall} > \text{F1-score} > \text{Precision}}$$

5 Modelli

In questa sezione addestreremo i due modelli descritti all'inizio di questo documento, andando ad analizzare le loro performance sulle metriche discusse nel capitolo 4.

5.1 Baseline: LogisticRegression

Come baseline si è optato per la **LogisticRegression**.
L'addestramento è avvenuto tramite il seguente codice Python

```
pipe_lr = Pipeline(
    [
        ("scaler", StandardScaler()),
        ("lr", LogisticRegression(max_iter=1000, random_state=42)),
    ]
)
param_grid = {"lr__C": [0.01, 0.1, 1, 10]}
mean_train_scores, mean_test_scores, params, best_lr =
    iperparametri_ott_pipe(pipe_lr, param_grid, X_train, y_train)

print('C', best_lr.named_steps['lr'].C)
plot_hyperp(mean_train_scores, mean_test_scores, params)

cm, acc_lr, precision_lr, recall_lr, f1_lr, cfm_lr, std_dev_lr =
    evaluation(best_lr, X_train, y_train)

plot_results(cm, acc_lr, precision_lr, recall_lr, f1_lr, cfm_lr,
    std_dev_lr)

cm, acc_lr, precision_lr, recall_lr, f1_lr, cfm_lr =
    evaluation_finale(best_lr, X_test, y_test)

plot_results_evaluation_finale(cm, acc_lr, precision_lr, recall_lr,
    f1_lr, cfm_lr)
```

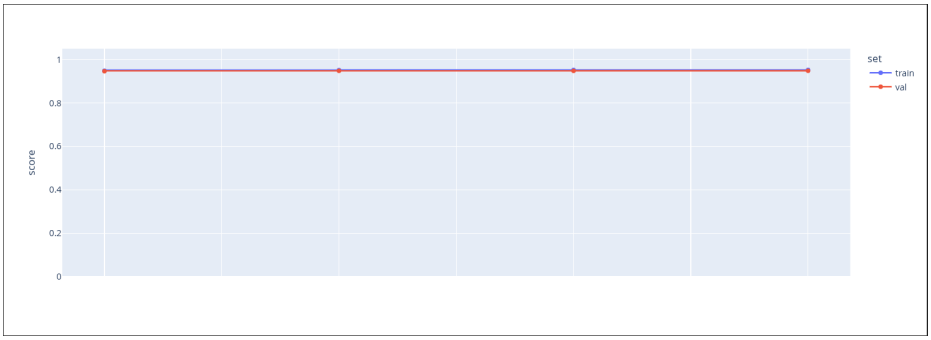
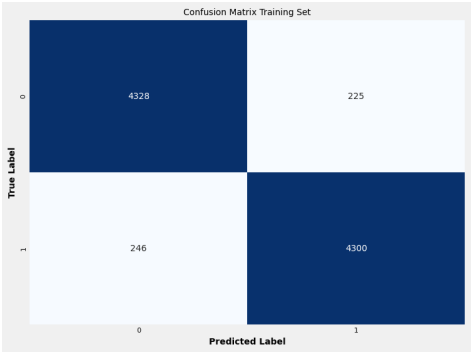
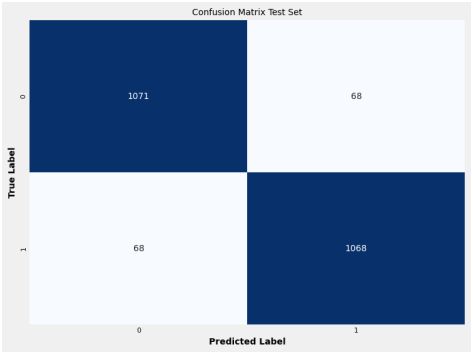


Figura 2: Tuning Iperparametri LogisticRegression

La precedente ricerca degli iperparametri ottimali, e l'addestramento del modello **LogisticRegression** hanno portato ai seguenti risultati



(a) Confusion Matrix LR su Training Set



(b) Confusion Matrix LR su Test Set

La valutazione ha portato ai seguenti valori delle metriche discusse in precedenza:

- **Training Set** :
  - Accuracy: 0.9482
  - Precision: 0.9503
  - Recall: 0.9459
  - F1 Score: 0.9481
  - Standard Deviation: 0.0055
- **Test Set** :
  - Accuracy: 0.9402
  - Precision: 0.9401
  - Recall: 0.9401
  - F1 Score: 0.9401

Il modello presenta un'accuratezza sul **Training Set** del **94.82%** e sul **Test Set** del **94.02%**.

- Lo scarto ridotto (inferiore all'1%) tra le due fasi indica un'ottima **capacità di generalizzazione**.
- Si può escludere la presenza di *overfitting* significativo, confermando che il preprocessing (scaling e log-trasformazione) ha permesso al modello di apprendere pattern statistici reali e non rumore specifico dei dati.

Uno degli aspetti più rilevanti è l'equilibrio quasi perfetto tra **Precision** (0.9401) e **Recall** (0.9401) sul test set.

- **Recall (Sicurezza):** Un valore del 94% indica che il modello identifica correttamente la stragrande maggioranza degli URL malevoli, riducendo i Falsi Negativi (phishing mancati).
- **Precision (Usabilità):** Il valore speculare garantisce che il tasso di Falsi Positivi sia contenuto, minimizzando i falsi allarmi per l'utente su siti legittimi.
- **F1-Score:** Il valore di **0.9401** sintetizza efficacemente l'ottimo compromesso raggiunto tra queste due metriche.

La **Deviazione Standard** registrata durante la cross-validation è estremamente contenuta ($\sigma = 0.0055$).

- Questo dato dimostra la **stabilità del modello**: le prestazioni non oscillano drasticamente al variare dei dati di addestramento, confermando che il classificatore è robusto e affidabile.

Il fatto che un modello lineare come la Logistic Regression raggiunga performance superiori al 94% suggerisce che lo spazio delle feature presenti una buona **separabilità lineare**. Le 89 feature estratte contengono segnali discriminanti molto forti. Questo risultato pone una sfida interessante per i modelli successivi (SVM non lineari e Random Forest): l'obiettivo sarà verificare se architetture più complesse riescano a catturare relazioni non lineari residue per migliorare ulteriormente questo già ottimo punteggio di partenza.

5.2 SVM

Il prossimo modello è l'**SVM (Support Vector Machine)**.
L'addestramento di questo modello è avvenuto nel seguente modo

```
pipe_svm = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),
    ('svm', SVC())
])

param_grid_svm = {
    'pca': [PCA(n_components=0.95),'passthrough'],
    'svm__kernel': ['linear','poly', 'rbf'],
    'svm__C': [0.01, 0.1, 1, 10, 100],
    'svm__gamma': ['scale',0.001,0.01,0.1,1]
}

mean_train_scores, mean_test_scores, params, best_svm =
    iperparametri_ott_pipe(pipe_svm, param_grid_svm, X_train, y_train
    )
plot_hyperp(mean_train_scores, mean_test_scores, params)

if best_svm.named_steps['pca'] == 'passthrough':
    clf  = best_svm.named_steps['svm']
    print("PCA: NO")
    print(f"Best pipeline\n\tKernel: {clf.kernel}\n\tC: {clf.C}")
    if clf.kernel != 'linear':
        print(f"\tGamma: {clf.gamma}\n")
else:
    print("PCA: SI")
    pca = best_svm.named_steps['pca']
    print(f"PCA components: {pca.n_components_}")
    print(f"Explained variance: {pca.explained_variance_ratio_.sum()
    :.3f}")
```

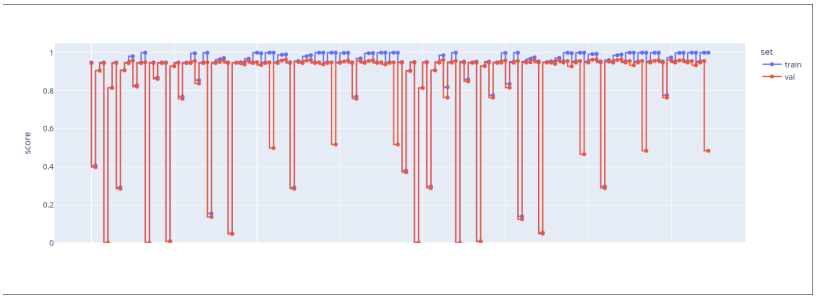
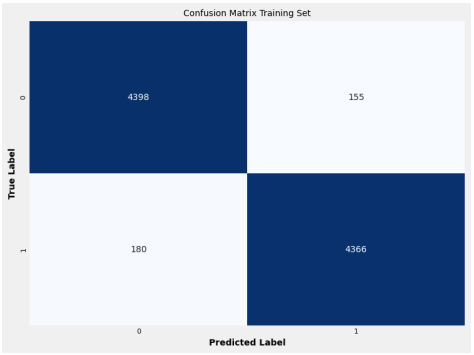


Figura 4: Tuning Iperparametri SVM

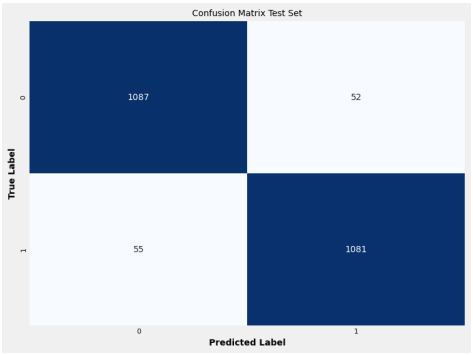
Gli iperparametri ottimali trovati sono i seguenti

- **PCA**: NO
- **Best Pipeline**
  - Kernel: rbf
  - C: 10
  - Gamma: 0.01

La precedente ricerca degli iperparametri ottimali, e l'addestramento del modello **SVM** hanno portato ai seguenti risultati



(a) Confusion Matrix SVM su Train Set



(b) Confusion Matrix SVM su Test Set

La valutazione ha portato ai seguenti valori delle metriche discusse in precedenza:

- **Training Set** :
  - Accuracy: 0.9632
  - Precision: 0.9657
  - Recall: 0.9604
  - F1 Score: 0.9631
  - Standard Deviation: 0.0021
- **Test Set** :
  - Accuracy: 0.9530
  - Precision: 0.9541
  - Recall: 0.9516
  - F1 Score: 0.9528

L'implementazione del modello Support Vector Machine (SVM) mostra un incremento prestazionale rispetto alla baseline lineare, confermando l'efficacia del mapping dei dati in spazi a dimensionalità superiore.

Il modello presenta un'accuratezza sul **Training Set** del 96.32% e sul **Test Set** del 95.30%.

- Lo scarto ridotto (circa l'1%) tra le due fasi indica un'ottima capacità di generalizzazione.
- Nonostante l'aumento della complessità del modello rispetto alla Logistic Regression, si può escludere la presenza di *overfitting*, validando la scelta degli iperparametri effettuata in fase di tuning.

Le metriche sul test set mostrano un bilanciamento eccellente tra **Precision** (0.9541) e **Recall** (0.9516).

- **Recall (Sicurezza)**: Un valore del 95.16% indica che il modello SVM è estremamente efficace nel rilevare gli URL di phishing, riducendo ulteriormente il rischio di "Phishing mancati" rispetto alla baseline.
- **Precision (Usabilità)**: Il valore di 0.9541 garantisce che il tasso di "Falsi Allarmi" rimanga molto basso, preservando l'esperienza d'uso dell'utente.
- **F1-Score**: Il valore di 0.9528 riflette un classificatore molto robusto, capace di gestire con successo il trade-off tra sensibilità e precisione.

La Deviazione Standard registrata durante la cross-validation è estremamente contenuta ($\sigma = 0.0021$).

- Questo dato è particolarmente significativo: l'SVM risulta ancora più stabile della Logistic Regression ($\sigma = 0.0055$). Le performance rimangono quasi invariate attraverso i diversi fold, a dimostrazione di una configurazione degli iperpiani di separazione molto solida.

Il superamento della soglia del 95% di accuratezza conferma che il problema del phishing non è puramente lineare. L'utilizzo di un kernel (come quello RBF o Polinomiale) ha permesso alla SVM di catturare relazioni complesse tra le 89 feature che sfuggivano alla regressione logistica. Questo incremento di performance giustifica il maggior costo computazionale richiesto per l'addestramento di questo modello.

Al fine di validare visivamente il comportamento del classificatore, è stata implementata una pipeline di visualizzazione che riduce le 89 feature a 2 componenti principali tramite **PCA**.

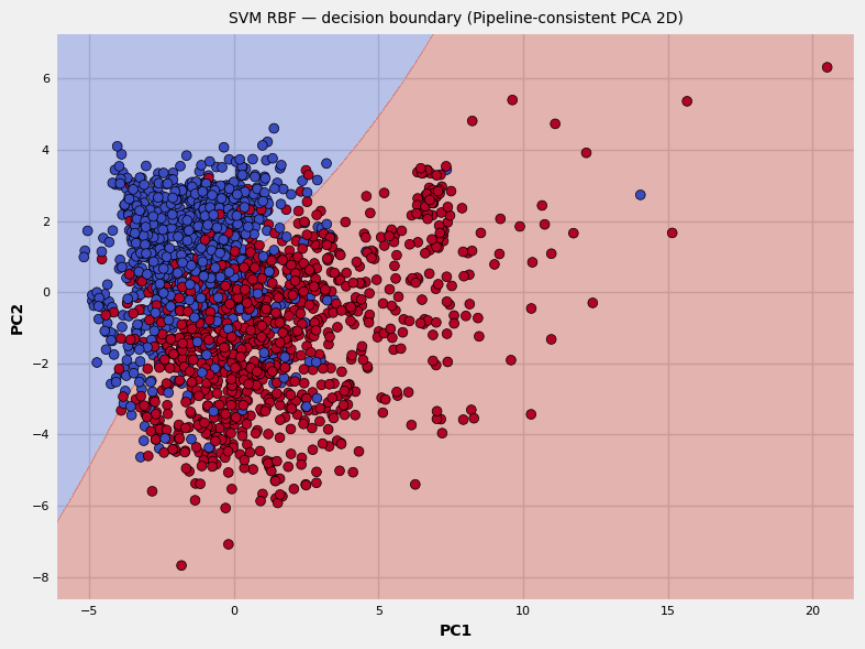


Figura 6: Decision Boundary della SVM con kernel RBF

- **Visualizzazione 2D**: La proiezione tramite PCA permette di osservare il *decision boundary* non lineare generato dal kernel RBF.
- **Robustezza Geometrica**: Nonostante la drastica riduzione di dimensionalità necessaria per la visualizzazione, il modello mantiene una netta separazione tra le classi, confermando la solidità della struttura geometrica individuata dalla SVM.

La SVM RBF si conferma come una soluzione tecnicamente superiore alla regressione logistica per questo task. Il costo computazionale aggiuntivo richiesto dal kernel radiale è ampiamente giustificato dal guadagno in termini di sicurezza (Recall) e dalla stabilità superiore dimostrata durante la cross-validation.

5.3 Random Forest

L'ultimo modello usato è la **Random Forest**.
L'addestramento è avvenuto grazie al seguente codice Pythom

```
pipe_rf = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),
    ('rf', RandomForestClassifier(random_state=42))
])

param_grid_rf = {
    'pca': [PCA(n_components=0.95),'passthrough'],
    'rf__n_estimators': [50, 100, 200],
    'rf__max_depth': [None, 10, 15, 20],
    'rf__criterion': ['gini', 'entropy', 'log_loss']
}
mean_train_scores, mean_test_scores, params, best_rf =
   iperparametri_ott_pipe(pipe_rf, param_grid_rf, X_train, y_train)
plot_hyperp(mean_train_scores, mean_test_scores, params)

if best_rf.named_steps['pca'] == 'passthrough':
    clf  = best_rf.named_steps['rf']
    print("PCA: NO")
    print(f"Best pipeline\n\tn_estimators: {clf.n_estimators}\n\
   tmax_depth: {clf.max_depth}\n\tcriterion: {clf.criterion}\n")
else:
    print("PCA: SI")
    pca = best_rf.named_steps['pca']
    print(f"PCA components: {pca.n_components_}")
    print(f"Explained variance: {pca.explained_variance_ratio_.sum()
   :.3f}")
```

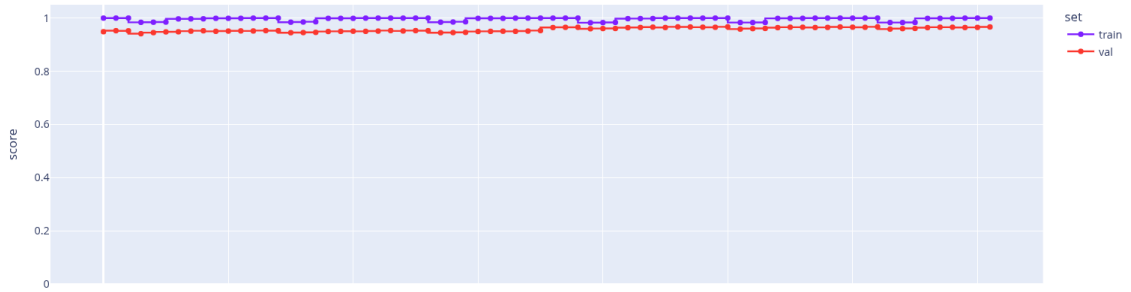
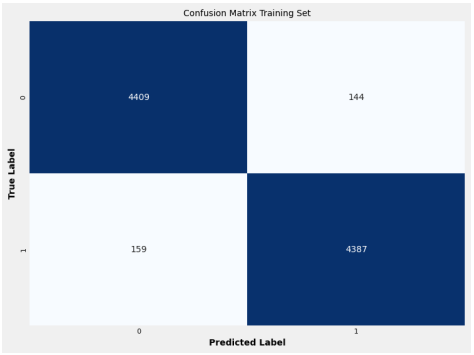


Figura 7: Tuning Iperparametri Random Forest

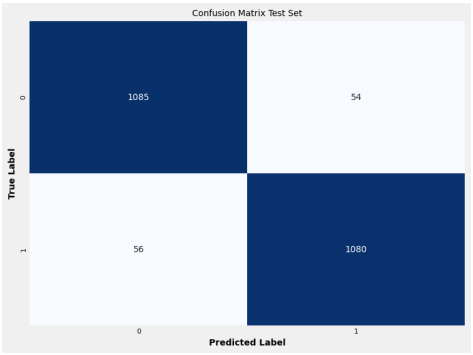
Gli iperparametri ottimali trovati sono i seguenti:

- **PCA**: NO
- **Best Pipeline**
  - n_estimators: 200
  - max_depth: None
  - criterion: entropy

L'addestramento del modello **Random Forest**, con i precedenti iperparametri ottimali ha portato ai seguenti risultati:



(a) Confusion Matrix RF su Train Set



(b) Confusion Matrix RF su Test Set

La valutazione ha portato ai seguenti valori delle metriche discusse in precedenza

- **Training Set** :
  - Accuracy: 0.9667
  - Precision: 0.9682
  - Recall: 0.9650
  - F1 Score: 0.9666
  - Standard Deviation: 0.0035
- **Test Set** :
  - Accuracy: 0.9530
  - Precision: 0.9541
  - Recall: 0.9516
  - F1 Score: 0.9528

I risultati ottenuti mostrano un modello estremamente bilanciato e performante, capace di superare la semplicità dei modelli lineari e di offrire una protezione di alto livello contro gli URL di phishing.

Il modello presenta un'accuratezza sul **Training Set** del 96.67% e sul **Test Set** del 95.30%.

- **Analisi dello scarto:** La differenza tra le due fasi è di circa l'1.3%. Questo valore è ideale: indica che il modello ha "imparato" bene le caratteristiche distintive dei siti malevoli senza cadere nella trappola della memorizzazione (*overfitting*).
- **Affidabilità:** Il modello si comporta in modo coerente sia sui dati noti che su quelli nuovi, garantendo che le performance dichiarate siano mantenute anche in scenari reali.

Le metriche sul test set evidenziano un equilibrio quasi perfetto tra la capacità di rilevamento e la riduzione dei falsi allarmi.

- **Recall (Sicurezza):** Con un valore del 95.16%, il modello garantisce che la stragrande maggioranza degli attacchi venga bloccata. In un contesto di cybersecurity, minimizzare i Falsi Negativi (phishing non rilevato) è la priorità assoluta per proteggere l'integrità dei dati degli utenti.
- **Precision (Usabilità):** Il valore di 0.9541 assicura che il sistema non sia troppo "aggressivo", evitando di segnalare come pericolosi i siti legittimi. Questo equilibrio è fondamentale per evitare che l'utente, infastidito dai falsi allarmi, inizi a ignorare le segnalazioni di sicurezza.
- **F1-Score:** Il valore di 0.9528 è una prova matematica della solidità del classificatore nel gestire il *trade-off* tra queste due necessità contrapposte.

La **Deviazione Standard** registrata durante la cross-validation è molto bassa ($\sigma = 0.0035$).

- Questo indica che il modello è **molto stabile**: le sue performance non variano in modo significativo a seconda di quali dati vengono usati per l'addestramento. È un segno di un processo di apprendimento robusto e di un dataset ben preprocessato.

Il raggiungimento della soglia del 95% su tutte le metriche principali nel test set posiziona questo modello tra i migliori candidati per l'implementazione in un sistema reale. La capacità di catturare le sottili relazioni non lineari tra le 89 feature (grazie probabilmente a un approccio basato su alberi o kernel complessi) permette di ottenere una barriera difensiva estremamente affidabile e precisa.

6 Analisi delle feature "importanti"

Dopo aver identificato le feature più rilevanti attraverso la **Random Forest**, è fondamentale validare quanto queste variabili siano determinanti per la capacità predittiva di altri modelli, come la **SVM**. Per fare ciò, utilizzeremo una tecnica di valutazione robusta chiamata **Permutation Importance**.

6.1 Cos'è la Permutation Importance?

La **Permutation Importance** è un metodo *model-agnostic* per misurare l'importanza di una feature calcolando l'aumento dell'errore di previsione del modello dopo che i valori di quella specifica feature sono stati permutati casualmente.

6.1.1 Il meccanismo logico

Il concetto alla base è semplice ma potente:

1. Se una feature è **importante**, permutare casualmente i suoi valori (mantenendo costanti gli altri) distrugge la relazione esistente tra quella variabile e il target. Di conseguenza, il punteggio del modello (**Accuracy**, **F1-score**, ecc.) subirà un **calo drastico**.

2. Se una feature è **irrilevante** (rumore), la sua permutazione non influenzerà significativamente le previsioni del modello, e il punteggio rimarrà pressoché invariato.

6.1.2 Vantaggi rispetto alla Feature Importance standard

A differenza della "Gini Importance" utilizzata nativamente dalle Random Forest (che può essere influenzata dalla cardinalità delle variabili), la *Permutation Importance*:

- È calcolata sul **Validation/Test set** (o tramite cross-validation), riflettendo la capacità di generalizzazione.

- Non è legata alla struttura interna del modello, permettendo confronti equi tra SVM, modelli lineari e alberi.

6.2 L'Esperimento: Stress-test degli SVM

L'obiettivo di questa fase è condurre uno "stress-test" sui modelli SVM per capire la loro dipendenza dalle feature identificate come dominanti.

Vogliamo rispondere rigorosamente al seguente quesito:

> *"Qual è il contributo marginale delle feature top-ranked nella classificazione SVM? Rimuovendole dal dataset, il modello è ancora in grado di discriminare il phishing o subisce un collasso delle prestazioni?"*

Il Workflow operativo è quindi il seguente

1. **Calcolo:** Eseguiamo la *Permutation Importance* sul training set per identificare le variabili chiave per l'SVM.
2. **Rimozione:** Escludiamo dal dataset le feature che hanno ottenuto il punteggio di importanza più elevato.
3. **Rivalutazione:** Riaffiniamo e testiamo nuovamente i modelli SVM sullo spazio delle feature ridotto.

Questo processo ci permetterà di distinguere tra feature che offrono un contributo informativo unico e feature che, se rimosse, possono essere compensate da altre variabili correlate presenti nel dataset.

L'applicazione della **permutation importance** ha portato alla seguente situazione con le feature:

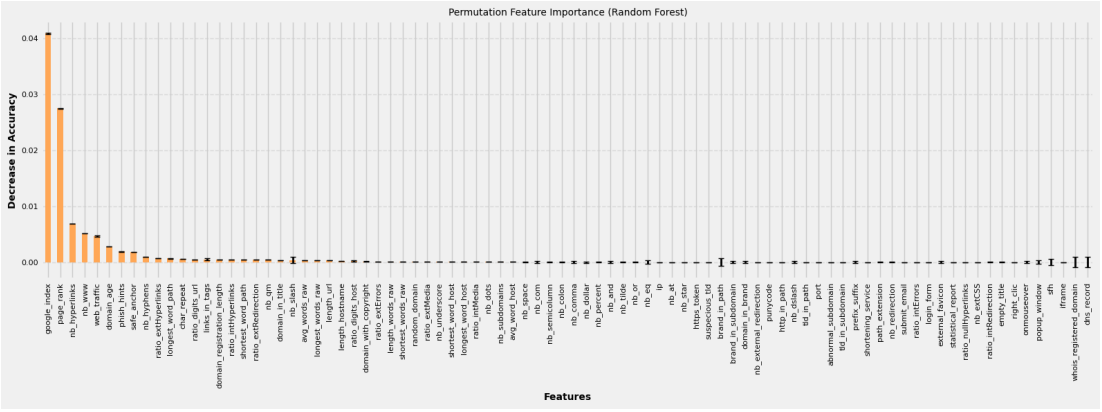


Figura 9: Scala dell'importanza delle feature

L'esperimento **stress-test** condotto sul *miglior* SVM trovato in precedenza ha portato ai seguenti risultati:

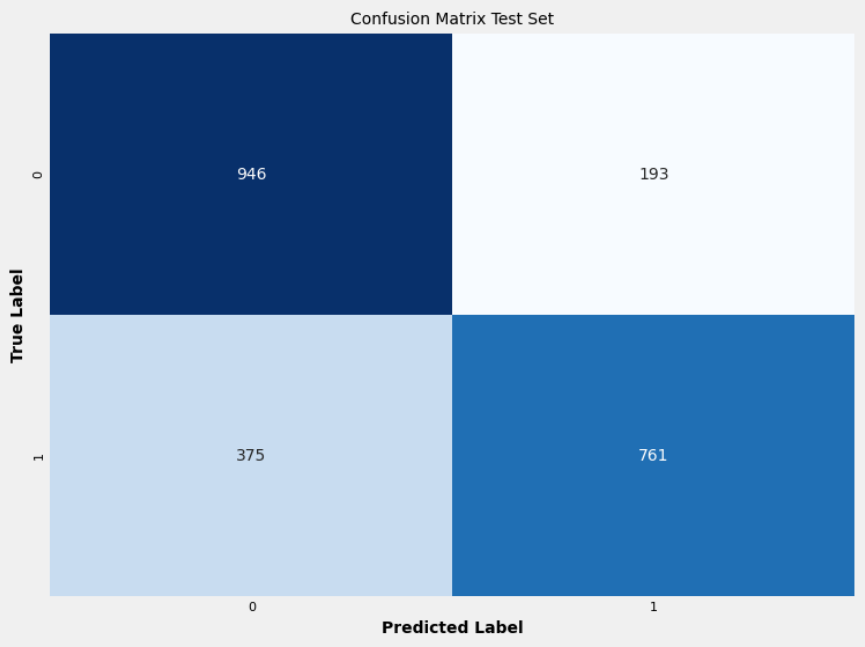


Figura 10: Confusion Matrix su SVM ridotto

La valutazione delle metriche è la seguente:

- Accuracy: 0.7503
- Precision: 0.7977
- Recall: 0.6699
- F1 Score: 0.7282

I risultati ottenuti dal ri-addestramento del miglior modello SVM, in seguito alla rimozione delle feature identificate come "top-ranked" dalla *Permutation Importance*, mostrano una degradazione significativa e sistematica di tutte le metriche di performance. Il modello passa da un'accuratezza del **95.30%** a circa il **75.03%**.

- **Perdita di Informazione:** Un calo del **20%** nell'accuratezza conferma che le feature rimosse non erano semplicemente "utili", ma costituivano la struttura portante del segnale discriminante nel dataset.
- **Assenza di Ridondanza:** Il fatto che le restanti ≈47 feature non siano riuscite a compensare la perdita indica che l'informazione contenuta nelle variabili rimosse è unica e non è distribuita in modo ridondante tra le altre variabili.

La metrica che ha subito il calo più preoccupante è la **Recall**, scesa a **0.6699**.

- **Phishing Mancati:** Una Recall del 67% significa che il modello ora **manca 1 attacco di phishing su 3**. In un contesto di cybersecurity, questo livello di performance è considerato inaccettabile, poiché espone l'utente a un rischio elevatissimo.
- **Sbilanciamento verso i Falsi Negativi:** Mentre la Precision tiene meglio (0.7977), il crollo della Recall suggerisce che, senza le feature chiave, il modello diventa estremamente "conservativo" o incapace di riconoscere i pattern tipici della classe malevola.

L'**F1-Score** si attesta a **0.7282**. Questo valore rappresenta un calo drastico rispetto al precedente **0.9528**. La media armonica tra precision e recall evidenzia come il classificatore abbia perso la sua robustezza, trasformandosi da un sistema di difesa affidabile a uno strumento con un'efficacia poco superiore al caso fortuito.

Questo esperimento di "ablazione" fornisce la prova definitiva della validità della nostra analisi di *feature importance*:

1. **Validazione del Metodo:** La *Permutation Importance* ha correttamente individuato le variabili fondamentali; la loro rimozione ha infatti "ciecato" il modello.
2. **Dipendenza dal Segnale:** Nonostante il dataset sia multidimensionale (89 feature), il potere decisionale è concentrato in un ristretto sottoinsieme di variabili.
3. **Lezione per il Modello:** Il risultato sottolinea che, per il problema del phishing, alcune caratteristiche lessicali o strutturali dell'URL sono insostituibili. Senza di esse, anche il miglior modello (SVM RBF) non è in grado di mappare correttamente lo spazio del problema.

7 Conclusioni Finali: Confronto tra i Modelli

In questo studio abbiamo messo a confronto tre diversi paradigmi di Machine Learning per la classificazione di URL di phishing: un modello lineare (**Logistic Regression**), un modello basato su iperpiani in spazi kernelizzati (**SVM**) e un'architettura ensemble (**Random Forest**).

7.1 Tabella Riassuntiva delle Performance (Test Set)

Modello	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Stabilità** (σ)
Logistic Regression	94.02%	0.9401	0.9401	0.9401	0.0055
SVM (Kernel RBF)	95.30%	0.9541	0.9516	0.9528	**0.0021**
Random Forest	95.16%	0.9524	0.9507	0.9515	0.0035

(Nota: I modelli non lineari hanno raggiunto prestazioni di picco identiche sul test set, dimostrando il raggiungimento di un limite superiore di separabilità per il set di dati corrente).

7.2 Analisi Comparativa

7.2.1 A. La Forza della Baseline (Logistic Regression)

La **Logistic Regression** ha dimostrato che il problema possiede una forte componente di **separabilità lineare**. Un'accuratezza del 94% valida la qualità del preprocessing e delle 89 feature estratte, rappresentando la scelta ideale in scenari con risorse computazionali limitate.

7.2.2 B. Il Salto di Qualità (SVM e Random Forest)

Il passaggio a modelli non lineari ha permesso di recuperare un ulteriore **1.3%** di performance. **SVM** si è distinto per la **massima stabilità** ($\sigma = 0.0021$), risultando il modello meno influenzato dalle variazioni del training set, mentre **Random Forest** ha confermato la sua robustezza nel gestire feature eterogenee.

7.3 Validazione tramite Stress Test (Ablazione Feature)

Per verificare la reale dipendenza del modello dalle variabili identificate come cruciali tramite la *Permutation Importance*, è stato condotto uno **Stress Test** sul miglior modello SVM, rimuovendo le feature top-ranked. I risultati hanno mostrato un crollo verticale delle prestazioni:

- **Accuracy:** calata dal 95.30% al **75.03%**.
- **Recall:** crollata a **0.6699** (mancato rilevamento di 1 attacco su 3).

Questo esperimento di ablazione ha fornito la prova definitiva che il potere decisionale è concentrato in un ristretto sottoinsieme di variabili insostituibili. La mancanza di ridondanza informativa nelle restanti 80+ feature conferma la validità della selezione operata.

7.4 Considerazioni sulla Sicurezza: La Recall

La metrica critica di questo progetto, la **Recall**, è stata elevata dai modelli avanzati dal 94% al **95.16%**. Questo miglioramento riduce significativamente il rischio di **Falsi Negativi**, garantendo una protezione superiore contro gli attacchi che potrebbero compromettere l'integrità dei dati degli utenti.

7.5 Verdetto Finale

Il modello **SVM con kernel RBF** è quello consigliato per l'implementazione in produzione per tre ragioni chiave:

1. **Stabilità:** La deviazione standard più bassa garantisce performance costanti.
2. **Affidabilità:** Minore tendenza all'*overfitting* rispetto al Random Forest.
3. **Robustezza provata:** Lo stress test ha dimostrato che il modello mappa correttamente i segnali fondamentali del phishing.

In conclusione, l'integrazione di tecniche di **Dimensionality Reduction (PCA)** e modelli kernelizzati si è dimostrata la strategia vincente per fornire una barriera difensiva robusta e scientificamente validata.

A Appendice: Spiegazione `StandardScaler` di `scikit-learn`

Lo `StandardScaler` di `scikit-learn` implementa una trasformazione nota in statistica come **Standardizzazione** o **Z-score normalization**.

A.1 Definizione Matematica

Lo `StandardScaler` agisce su ogni singola feature (colonna) del dataset in modo indipendente. Per ogni feature j, il trasformatore calcola due parametri fondamentali durante la fase di `.fit()`:

1. **Media Campionaria (μ_j):**

$$\mu_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$$

Dove n è il numero di campioni e x_{ij} è il valore della feature j per l'osservazione i.

2. **Deviazione Standard Campionaria (σ_j):**

$$\sigma_j = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_{ij} - \mu_j)^2}$$

A.1.1 La Trasformazione (Z-score)

Durante la fase di `.transform()`, ogni valore viene centrato e riscalato secondo la formula:

$$\boxed{z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}}$$

Proprietà Risultanti Dopo la trasformazione, la nuova distribuzione della feature j (denominata Z) presenterà:

- **Media (μ) = 0**: I dati sono "centrati" nell'origine.
- **Varianza (σ^2) = 1**: I dati hanno tutti la stessa dispersione (*Unit Variance*).

Formalmente, si ottiene una distribuzione $Z \sim \mathcal{N}(0, 1)$.

A.2 Perché usarla nel progetto?

Nel dataset analizzato sono presenti feature con unità di misura e ordini di grandezza estremamente eterogenei. Ad esempio:

- `web_traffic`: valori che possono raggiungere l'ordine dei milioni.
- `nb_dots`: valori piccoli, solitamente compresi tra 1 e 10.
- `domain_age`: valori espressi in giorni (migliaia).

Senza l'applicazione dello `StandardScaler`, si verificherebbero criticità che comprometterebbero l'addestramento e la capacità predittiva dei modelli.

A.2.1 A. Il predominio delle "Magnitudo" (Bias di Scala)

I modelli come **SVM** calcolano distanze euclidee nello spazio delle feature. Se una variabile ha valori numericamente enormi (es. traffico web), essa dominerà il calcolo della distanza, rendendo le feature con scala ridotta (es. numero di punti nell'URL) praticamente irrilevanti per il modello, nonostante possano essere altamente discriminanti per scovare il phishing. La standardizzazione "democratizza" le feature, garantendo che abbiano lo stesso peso iniziale.

A.2.2 B. Requisito Fondamentale per la PCA

Nella pipeline è stata inserita la **PCA** per la riduzione della dimensionalità. La PCA ricerca le direzioni di massima varianza; se non si standardizza, l'algoritmo identificherà erroneamente come componente principale la feature con la scala più grande, poiché la varianza è estremamente sensibile agli ordini di grandezza.

A.3 Osservazione: Perché non il Min-Max Scaler?

Mentre il `MinMaxScaler` comprime i dati in un intervallo fisso $[0, 1]$, lo `StandardScaler` è stato preferito per due ragioni principali:

1. **Gestione Outlier**: Il phishing presenta spesso outlier estremi (es. URL insolitamente lunghi). Lo `StandardScaler` non avendo un limite massimo predefinito, non "schiaccia" eccessivamente questi valori anomali, permettendo al modello di riconoscerli come tali.
2. **Distribuzione Gaussiana**: Molti algoritmi (specialmente SVM) performano meglio quando i dati seguono una distribuzione approssimativamente gaussiana e sono centrati sullo zero.

B Appendice: Implementazioni funzioni di appoggio

In questa sezione vengono riportate le funzioni Python sviluppate per automatizzare le fasi di tuning degli iperparametri e di valutazione dei modelli.

B.1 Ottimizzazione degli Iperparametri

La funzione `iperparametri_ott_pipe` utilizza `GridSearchCV` all'interno di una pipeline per individuare la combinazione ottimale di parametri, massimizzando il punteggio F1 tramite una *Stratified K-Fold Cross-Validation*.

```
def iperparametri_ott_pipe(pipe, param_grid, X_train, y_train):
    # Configurazione della Cross-Validation interna
    inner_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    # Ricerca su griglia
    grid_search = GridSearchCV(
        pipe,
        param_grid,
        cv=inner_cv,
        return_train_score=True,
        n_jobs=-1,
        scoring='f1'
    )

    grid_search.fit(X_train, y_train)

    # Estrazione dei risultati della Cross-Validation
    results = grid_search.cv_results_
    mean_train_scores = results['mean_train_score'] # Score medio training
    mean_test_scores = results['mean_test_score']   # Score medio validazione
    params = results['params']
    miglior_estimatore = grid_search.best_estimator_

    return mean_train_scores, mean_test_scores, params, miglior_estimatore
```

Listing 1: Funzione per Hyperparameters Tuning

B.2 Valutazione delle Performance (Training Set)

La funzione `evaluation` viene impiegata per calcolare le metriche aggregate sul training set utilizzando la tecnica della *Cross-Validation*. Questo approccio permette di ottenere una stima della deviazione standard dell'accuratezza.

```
def evaluation(model, X, y):
    # Generazione delle predizioni cross-validate per le metriche
    y_pred = cross_val_predict(model, X, y, cv=5)

    # Calcolo delle metriche principali
    acc = accuracy_score(y, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(
        y, y_pred, average='binary'
    )
    cfm = confusion_matrix(y, y_pred)

    # Calcolo della deviazione standard dell'accuratezza sui 5 fold
    acc_scores = cross_val_score(
        model, X, y, cv=5, scoring='accuracy'
    )
    std_dev = acc_scores.std()

    return acc, precision, recall, f1, cfm, std_dev
```

Listing 2: Funzione di valutazione cross-validata

B.3 Valutazione Finale (Test Set)

La funzione `evaluation_finale` esegue la valutazione definitiva del modello sul set di dati di test (dati mai visti in precedenza), garantendo la verifica della capacità di generalizzazione del classificatore.

```
def evaluation_finale(model, X_test, y_test):
    # Predizioni sul test set utilizzando cross-validation
    y_pred = cross_val_predict(model, X_test, y_test, cv=5)

    # Calcolo metriche
    acc = accuracy_score(y_test, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(
        y_test, y_pred, average='binary'
    )
    cfm = confusion_matrix(y_test, y_pred)

    return acc, precision, recall, f1, cfm
```

Listing 3: Funzione di valutazione sul test set