# Progetto ML

# PROGETTO MACHINE LEARNING: PHISING URL RECOGNITION

## Abstract: Rilevamento di URL di Phishing tramite Tecniche di Dimensionality Reduction e Apprendimento Supervisionato

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).

## 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. **Analidi della skewness e magnitudo delle feature:** 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. **PCA** per ridurre la dimensionalità del dataset

## 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.

## Baseline

Come Baseline, sono stati scelti due modelli:

1. **DummyClassifier** : la baseline più semplice fra tutte
2. **LogisticRegression** : modello più semplice, ci servirà da base reale (farà anche da **strong baseline**)

Leggiamo il dataset con

```
data = pd.read_csv('Dataset/web-page-phishing/dataset_phishing.csv')
```

---

## 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, alcune di esse sono frutto di un mio ragionamento e pertanto potrebbero non essere corrette

### 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 (spesso usato nel phishing).
- **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).

- **https_token**: Indica se il token "https" è presente nella parte dell'host (non nel protocollo).
- **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 caratteri speciali (es. domini con accenti).
- **port**: Indica se nell'URL è specificata una porta non standard.

## 2. Feature del Dominio e Sottodomini

- **tld_in_path / tld_in_subdomain**: Presenza di un TLD (es. .com, .net) nel percorso o nel sottodominio.
- **abnormal_subdomain**: Indica se la struttura del sottodominio è anomala.
- **nb_subdomains**: Numero di sottodomini presenti.
- **prefix_suffix**: Presenza di trattini nel nome a dominio per separare prefissi o suffissi.
- **random_domain**: Indica se il dominio sembra generato casualmente.
- **shortening_service**: Indica se viene utilizzato un servizio di abbreviazione URL (es. bit.ly).

## 3. Feature Lessicali (Parole nell'URL)

- **length_words_raw**: Numero totale di parole identificate nell'URL.
- **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 nell'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 tipicamente usate negli attacchi phishing (es. "login", "update", "secure").

## 4. Feature del Contenuto della Pagina (HTML/JS)

- **nb_hyperlinks**: Numero totale di link presenti nella pagina web.
- **ratio_intHyperlinks / ratio_extHyperlinks / ratio_nullHyperlinks**: Percentuale di link che puntano allo stesso dominio, a domini esterni o che sono vuoti/nulli.
- **nb_extCSS**: Numero di file CSS caricati da domini esterni.
- **ratio_intRedirection / ratio_extRedirection**: Rapporto di reindirizzamenti interni ed esterni.
- **login_form**: Presenza di form di inserimento credenziali (input di tipo password).

- **external_favicon**: Indica se la favicon (l'icona del sito) è caricata da un dominio esterno.
- **links_in_tags**: Percentuale di link presenti nei tag (meta, script, link) rispetto al totale.
- **submit_email**: Indica se il form invia i dati direttamente a una mail (tramite `mailto:`).
- **ratio_intMedia / ratio_extMedia**: Rapporto di file multimediali (immagini, video) interni ed esterni.
- **iframe / popup_window**: Presenza di tag iframe o script che generano finestre popup.
- **safe_anchor**: Percentuale di ancore ( `<a>` ) che puntano a URL sicuri o allo stesso dominio.
- **onmouseover / right_clic**: Presenza di script che intercettano il movimento del mouse o disabilitano il tasto destro.
- **empty_title / domain_in_title**: Indica se il titolo della pagina è vuoto o se contiene il nome a dominio.

### 5. Feature Esterne e di Reputazione

- **whois_registered_domain**: Indica se il dominio è regolarmente registrato nei database WHOIS.
- **domain_registration_length**: Durata (in giorni) della registrazione del dominio.
- **domain_age**: Età del dominio in giorni (i domini recenti sono più sospetti).
- **web_traffic**: Rilevanza del sito in base al traffico web (es. ranking Alexa).
- **dns_record**: Presenza di record DNS validi per il dominio.
- **google_index**: Indica se la pagina è indicizzata su Google.
- **page_rank**: Valore del PageRank del sito (misura dell'autorevolezza).

### Target

- **status**: La variabile da predire; indica se l'URL è **legitimate** (sicuro) o **phishing** (malevolo).

# Analisi delle distribuzioni delle feature

## Target Label Encoding

Prima di tutto notiamo che la feature `url` non serve nella trattazione del nostro problema, pertanto possiamo rimuoverla

Utilizziamo inoltre una tecnica di Labeling chiamata **LabelEnconder** di `scikit.learn`, che ci permette di trasformare la colonna **status** (che corrisponde alla nostra colonna dei target `y`) in tutti valori numerici `0/1`.

La mappatura della feature `status` avviene nel seguente modo:

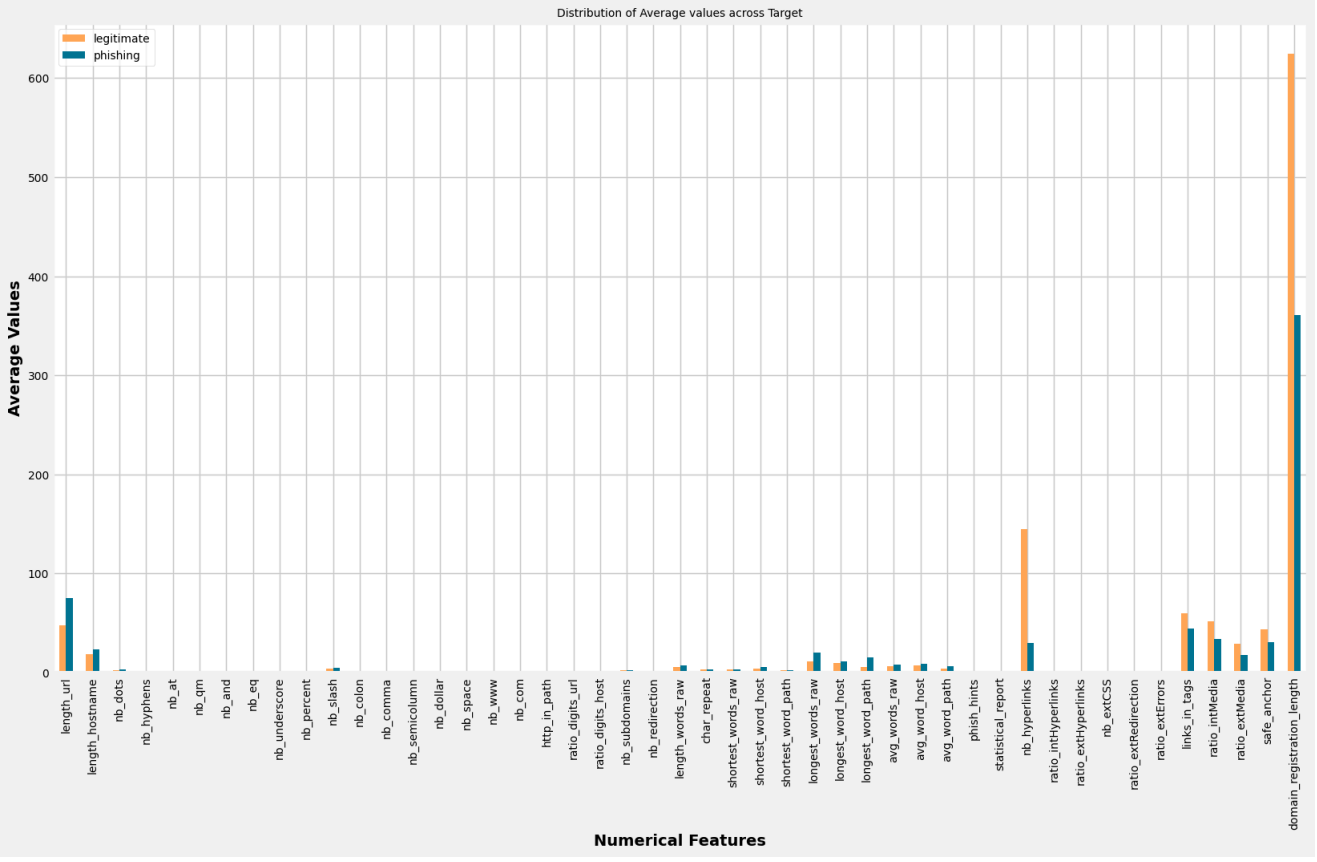
- Lo status = `legitimate` viene mappato nel numero $0$
- Lo status = `phishing` viene mappato nel numero $1$

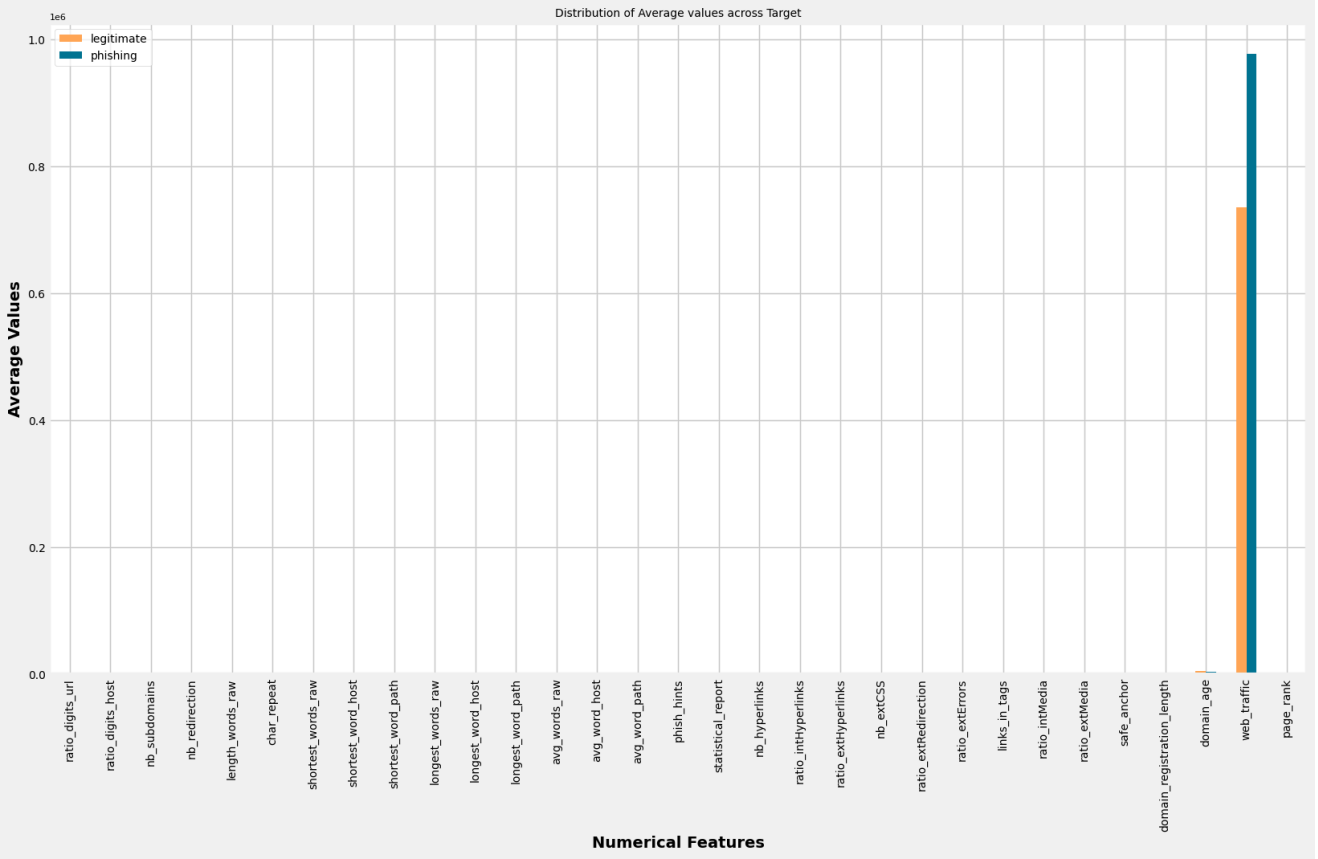
# Separazione delle feature

Andiamo ora ad analizzare le nostre feature, dividendole in due gruppi

- feature **numeriche**: feature che indicano misurazioni/conteggi; possono essere *continue* o *discrete*
- feature **categoriali**: feature che descrivono caratteristiche qualitative, etichette o gruppi non numerici; possono essere identificate con valori numerici, anche se non hanno un significato numerico intrinseco; possono essere dei flag booleani e indicati con $0/1$ nel dataset.

Visualizziamo la distribuzione quindi delle feature numeriche in relazione allo status



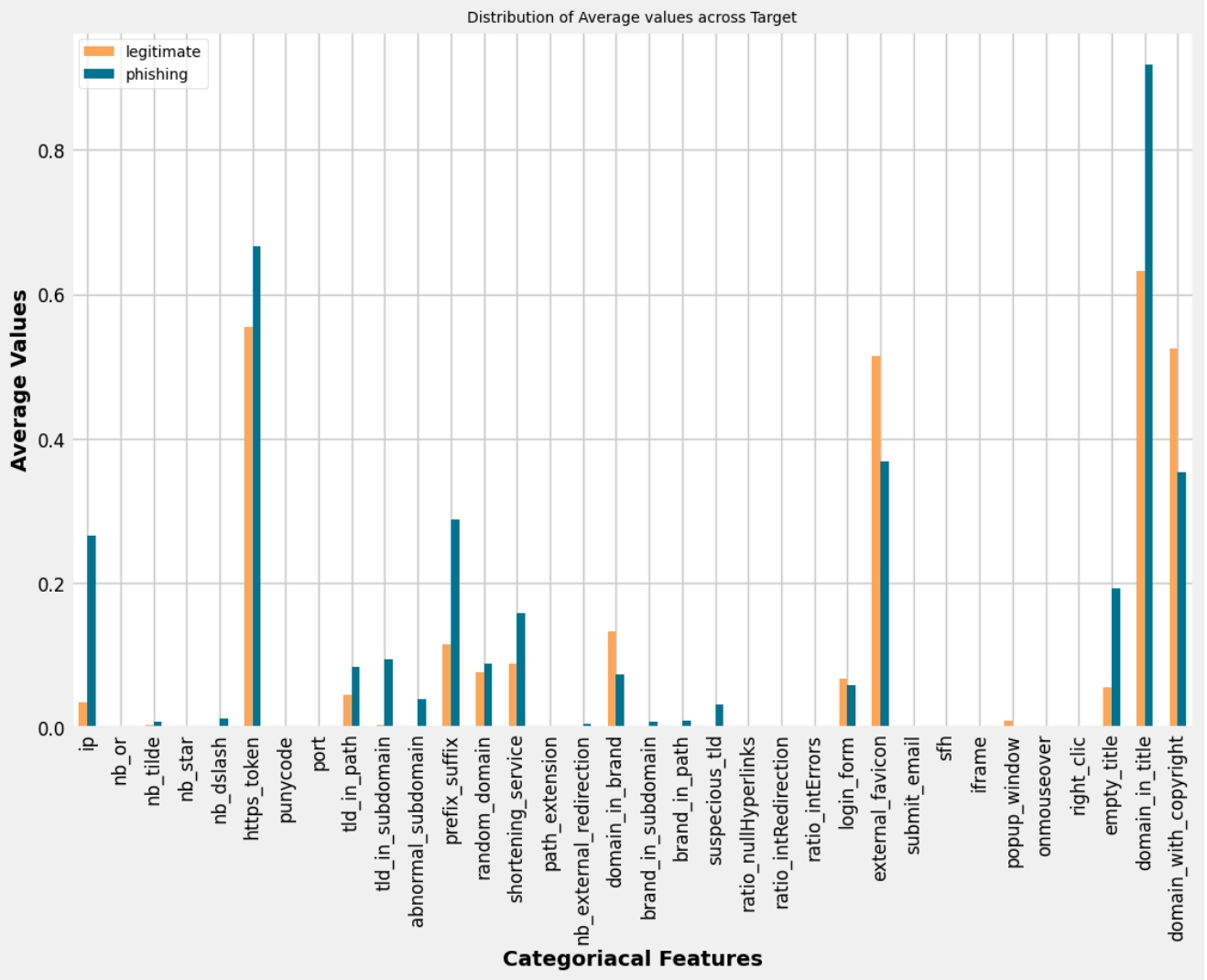



Osservazioni

- Più grande è la feature `lenght_url`, più è probabile che la url in questione sia `Phishing`
- C'è una differenza molto elevata per la feature `nb_hyperlinks`. Più grande significa più probabile che sia `Legitimate`
- L'alto volume delle feature `links_in_tags` , `safe_anchor` segnala che la URL sia più propensa vero lo status `Legitmate`
- Similmente alla feature `lenght_url`, più la feature `domain_registration_lenght` è elevata e più è probabile che il sito sia `Legitimate`

Inoltre, da come si può vedere nel secondo grafico, la feature `web_traffic` domina il dataset con la sua magnitudo elevatissima.
Si può anche notare come più è alto quel valore, e più il sito è probabile che sia di `Phishing`

Visualizziamo la distribuzione quindi delle feature categoriali in relazione allo status




Osservazioni

- Più grande è la feature `domain_in_title`, più è probabile che la url in questione sia `Phishing`
- C'è una differenza molto elevata per la feature `prefix_suffix`. Più grande significa più probabile che sia `Phishing`
- Similmente alla feature `prefix_suffix`, più la feature `ip` è elevata e più è probabile che il sito sia `Phishing`

# Skewness & Magnitudo delle feature

Analizziamo ora le differenze fra le varie feature.

Ci concentreremo per lo più su due fattori fondamentali, che sono:

- **skewness** delle feature
- **magnitudo** delle feature

## Skenwess : Definizione e Soluzione

La skewness di una feature ci indica quanto la distribuzione di quella feature ***si discosta*** da una distribuzione normale (la classica "campana" Gaussiana)

Possiamo identificare $3$ tipi di skewness nelle feature, che sono

| Tipo | Descrizione | Relazione tra Media e Mediana |
|---|---|---|
| Positiva (Right-skewed) | La "coda" della distribuzione è più lunga verso destra. La maggior parte dei dati è concentrata a sinistra. | Media > Mediana |
| Zero (Symmetrical) | La distribuzione è perfettamente simmetrica (come una Gaussiana). | Media ≈ Mediana |
| Negativa (Left-skewed) | La "coda" è più lunga verso sinistra. La maggior parte dei dati è concentrata a destra. | Media < Mediana |

Perchè questo fattore è fondamentale nella trattazione del nostro problema?

Perchè molti modelli di Machine Learning, tra cui il modello **SVM** (scelto per la risoluzione del task), sono sensibili all'asimmetria delle feature; questo potrebbe portare a situazioni spiacevoli quali:

- **Distorsione del/dei modello/i**
- **Performance ridotte**
- **Instabilità**

Per tali ragioni, verrà eseguito un controllo/calcolo dell'asimmetria delle feature presenti nel dataset. A quelle feature che hanno skewness elevata verrà applicata la tecnica della **log-trasformazione** ($log1p$ di scikit.learn)

Questa tecnica permette di applicare il logaritmo naturale $\ln(x)$ ai valori di una feature; questo permette di:

- Ridurre la skewness positiva
- Stabilizza la varianza

Inoltre, dato che ci sono valori vicini allo zero, e altri che sono esattamente zero, si è optato per l'applicazione di $\log(1+x)$ piuttosto che $\log(x)$

## Magnitudo: Definizione e Soluzione

Il magnitudo di una feature indica la grandezza dei valori di quella feature (ovvero la scala di valori).

Perchè questo fattore è di fondamentale trattazione?

Perchè i modelli basati su **distanza** (come il kNN e/o SVM) o su **discesa del gradiente** (come Regressione Lineare e/o Reti Neurali), se addestrati su un dataset avente feature con magnitudo estremamente diversi, potrebbero portare a risultati molto spiacevoli.

Prendiamo come esempio il modello SVM: lui cerca di trovare l'iperpiano che **massimizza il margine** tra le classi.

Se prendiamo due feature $x_1$ (es. reddito) e $x_2$ (es. età) tali per cui $mag(x_1) >> mag(x_2)$, ($mag(x_i) =$ magnitudo della feature $x_i$) allora la prima dominerà sulla seconda. L'algoritmo quindi "ignorerà" la seconda feature perchè, matematicamente, le sue variazioni sembrano irrilevanti rispetto a quelle del reddito

Per risolvere questo problema si è fatto uso della tecnica di **Standardizzazione (Z-Score Scaling)** di scikit.learn, che trasforma i dati in modo che abbiano **media=0** e **deviazione standard = 1**; cioè comprime i dati in modo che seguano una **distribuzione Gaussiana Standard** $\mathcal{N}(0, 1)$

Vedere appendice qui

# Train/Test Split

Diviamo il dataset originale in:

- Training Set: insieme di valori del dataset originale con cui verranno addestrati i nostri modelli
- Test Set: insieme di valori del dataset originale, ***separato*** dal training set, con cui verranno valutate le prestazioni e le performance dei nostri modelli

Il test set rappresenta quindi "nuovi" dati per i nostri modello, dati che loro $NON\ devono\ mai\ vedere$ prima della fine dell'addestramento.

La divisione avverrà secondo il classico rapporto $80/20$

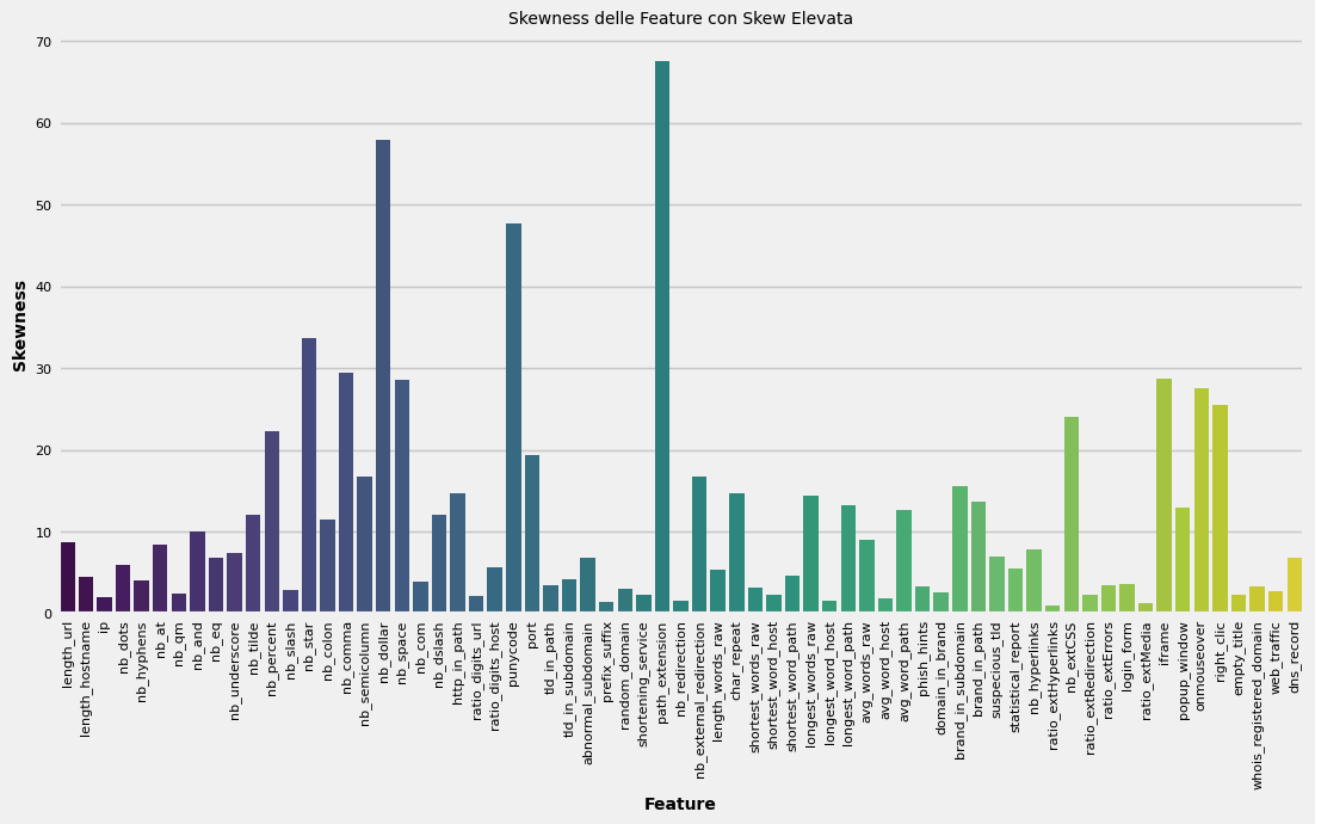
# Calcolo Skewness e Magnitudo

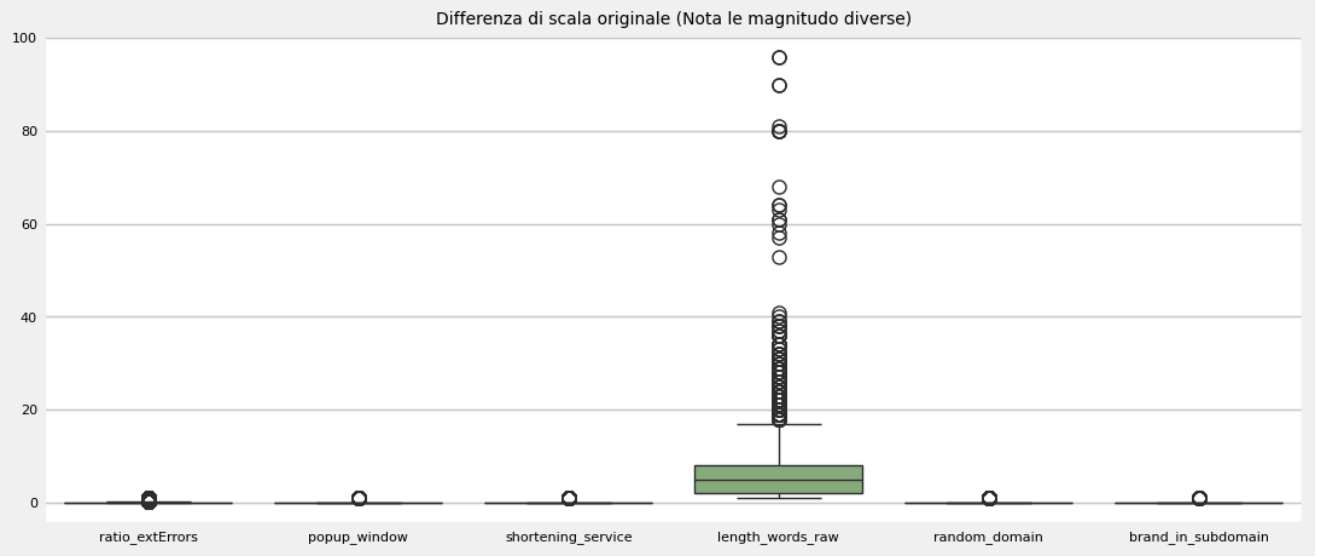
Calcoliamo quindi la skenwess e la magintudo delle feature.

Per quanto detto prima, prenderemo in considerazione le feature con skewness maggiore e ne fearemo la log trasformazione.

Poi applicheremo lo StandardScaler a tutte le feature.

Il tutto verrà eseguito tramite il processo `Pipeline` di scikit.learn, in modo da evitare il `Data Leakage` quando faremo la divisione con cross-validation.




# Definizione preprocessor

Definiamo quindi il nostro preprocessor, da inserire in fase di Training all'interno della Pipeline (in modo da garantire che ogni operazione venga fatta correttamente all'interno dei fold, così evitiamo **Data Leakage**)

Il preprocessor avrà quindi la seguente forma:

- alle feature estreme verrà applicato log-trasformazione
- a tutte le feature verrà applicato lo StandardScaler (comprese le feature estreme)

Per implementarlo, applicheremo la funzione **ColumnTransformer**

# Training & Evaluation

Dopo aver analizzato e diviso (train/test split) il nostro dataset originale, siamo pronti per addestrare i $2$ modelli descritti all'inizio di questo notebook.

Per ogni modello, verrà applicata una Pipeline contenente:

1. Per SVM - Preprocessor + Scaler + PCA
2. Per RandomForest due versioni, raw e solo con log-trasformazione

Alla fine, dopo aver addestratto tutti i modelli scelti, faremo proprio un confronto qualitativo fra essi, mettendo in luce caratteristiche come:

1. Tempi di addestramento
2. Precisione della predizione
3. F1-score
4. Precision e Recall
5. Deviazione standard ($\sigma$)

Al fine di evitare il più possibile situazioni critiche di overfitting, useremo la tecnica $K$-**fold Cross-validation**, con parametro $K = 5$

Perchè questa tecnica?

1. Applicare questa tecnica garantisce la stabilità delle performance e la minimizzazione del bias dovuto alla selezione del training set.
2. Questo approccio divide il training set in $5$ sottogruppi (**fold**): ciclicamente, i vari modelli verranno addestrati su $4$ di essi e validati sul restante.
3. I risultati riportati rappresentano la media aritmetica delle prestazioni ottenute nelle $5$ iterazioni, fornendo una stima più affidabile delle capacità predittiva rispetto al singolo split statico.

Avendo quindi optato per la seguente strategia, eseguiremo **scaling** e **PCA** dentro ogni fold (usando la `Pipeline` di scikit.learn), in modo da evitare il più possibile "Data Leakage", ovvero "iniettare" dati nel validation set

# Definizioni di Precision, Recall e F1-Score

Abbiamo detto che valuteremo, oltre all'accuracy dei modelli, le metriche:

- Precision
- Recall

- F1-Score

Prima di definire rigorosamente le 3 metriche introdotte, definiamo il concetto di `Confusion Matrix` e come questo viene applicato nel calcolo di Precision, Recall e F1-Score

# Confusion Matrix

Sia

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

il nostro dataset, con:

- $y_i \in \{0, 1\}$ il target per l'elemento $i$-esimo del dataset
- $\hat{y}_i \in \{0, 1\}$ il target **predetto** per l'elemento $i$-esimo del dataset

Definiamo la **Confusion Matrix** come la matrice

| | $\hat{y} = 0$ | $\hat{y} = 1$ |
|---|---|---|
| $y = 0$ | TN | FP |
| $y = 1$ | FN | TP |

Dove:

- **TP** (True Positives): istanze di phishing correttamente classificate come tali (**Phishing bloccati**)
- **FP** (False Positives): istanze legittime classificate come di phishing (**Falso Allarme**)
- **FN** (False Negatives): istanze di phishing classificate come legittime (**Phishing mancati**)
- **TN** (True Negatives): istanze di siti legittimi correttamente classificate come tali (**Siti sicuri**)

# Precision

Definiamo la `Precision` come la frazione:

$$\boxed{\text{Precision} = \frac{TP}{TP + FP}}$$

Formalmente, la precision di una classe è la **probabilità** che un'istanza sia effettivamente positiva, sapendo che il classificatore l'ha predetta come positiva,

ovvero:

$$\text{Precision} = Pr(y = 1|\hat{y} = 1)$$

# Recall

Definiamo la `Recall` come la frazione:

$$\boxed{\text{Recall} = \frac{TP}{TP + FN}}$$

Formalmente, la recall di una classe è la **probabilità** che il classificatore predica positivo, sapendo che l'istanza è effettivamente positiva, ovvero:

$$\text{Recall} = Pr(\hat{y} = 1|y = 1)$$

# F1-Score

Definiamo la `F1-Score` come la frazione:

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

Formalmente, la `F1-score` è la **media armonica** tra precision e recall, ed è massimo solo quando entrambe sono alte.

# Analisi decisionale del problema : Falsi Positivi o Falsi Negativi?

Classificazione binaria:

- Classe (1): URL di phishing
- Classe (0): URL legittimo

Errori possibili:

| Errore | Significato | Conseguenza reale |
|---|---|---|
| **False Positive (FP)** | URL legittimo classificato come phishing | Blocco/alert inutile |
| **False Negative (FN)** | URL phishing classificato come legittimo | Compromissione sicurezza |

**False Negative** (molto grave)

- L’utente clicca su un link malevolo

- Possibile:
  - furto credenziali
  - malware
  - compromissione account
  - danni economici
- Costo elevato e potenzialmente irreversibile

**False Positive** (fastidioso ma accettabile)

- L’utente vede un alert
- L’URL viene bloccato temporaneamente
- Può essere:
  - sbloccato manualmente
  - whitelistato
- Costo basso e reversibile

Data la trattazione del problema, quello che noi vogliamo che sia minimizzato è il numero di **Falsi Negativi**, di conseguenza, la metrica più importante fra tutte sarà la **Recall** della classe di Phishing

Di conseguenza, l'ordine che i nostri modelli dovranno rispettare sarà:

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

**Recall**: sicurezza

**F1**: compromesso globale

**Precision**: usabilità

---

# Implementazioni funzioni di appoggio

In questa sezione, implementiamo alcune funzioni di appoggio che ci serviranno più avanti

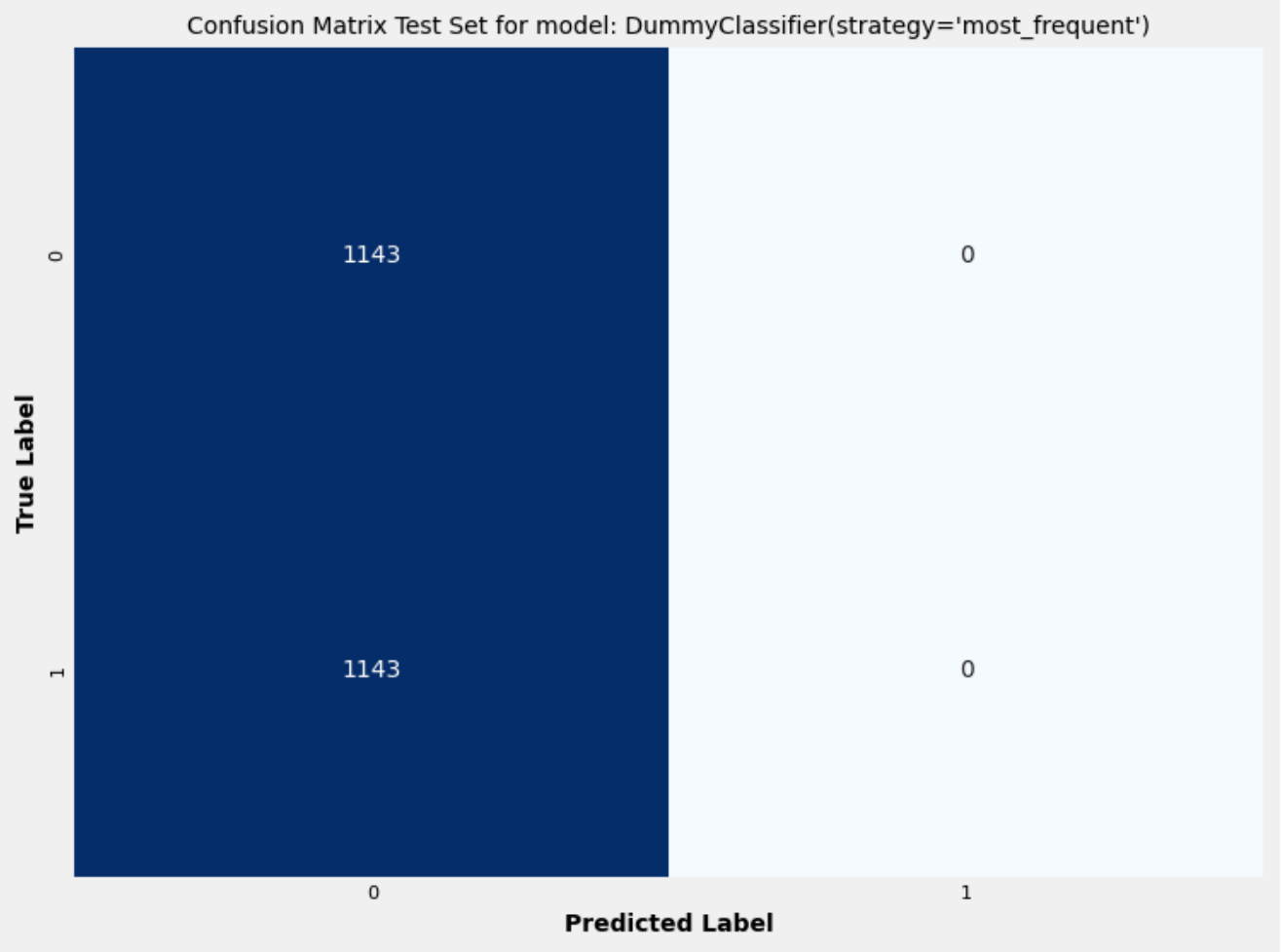
Ad esempio, fra queste funzioni abbiamo quella per calcolare gli iperparametri ottimali del modello, in modo da ottenere la predizione migliore fra tutte

---

# Baseline & Strong Baseline - DummyClassifier & LinearRegression

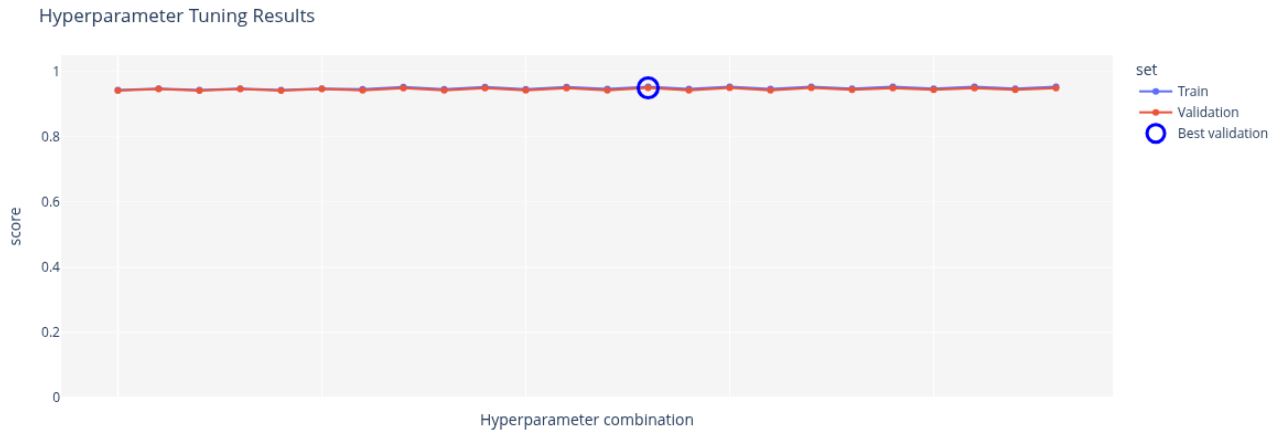
Come `baseline` sono stati scelti due modelli:

1. `DummyClassifier` di scikit.learn: è un classificatore che qualunque dato vede lo classifica in una singola classe
2. `LogisticRegression` : regressione logistica, più solida (fungerà da **strong** baseline)

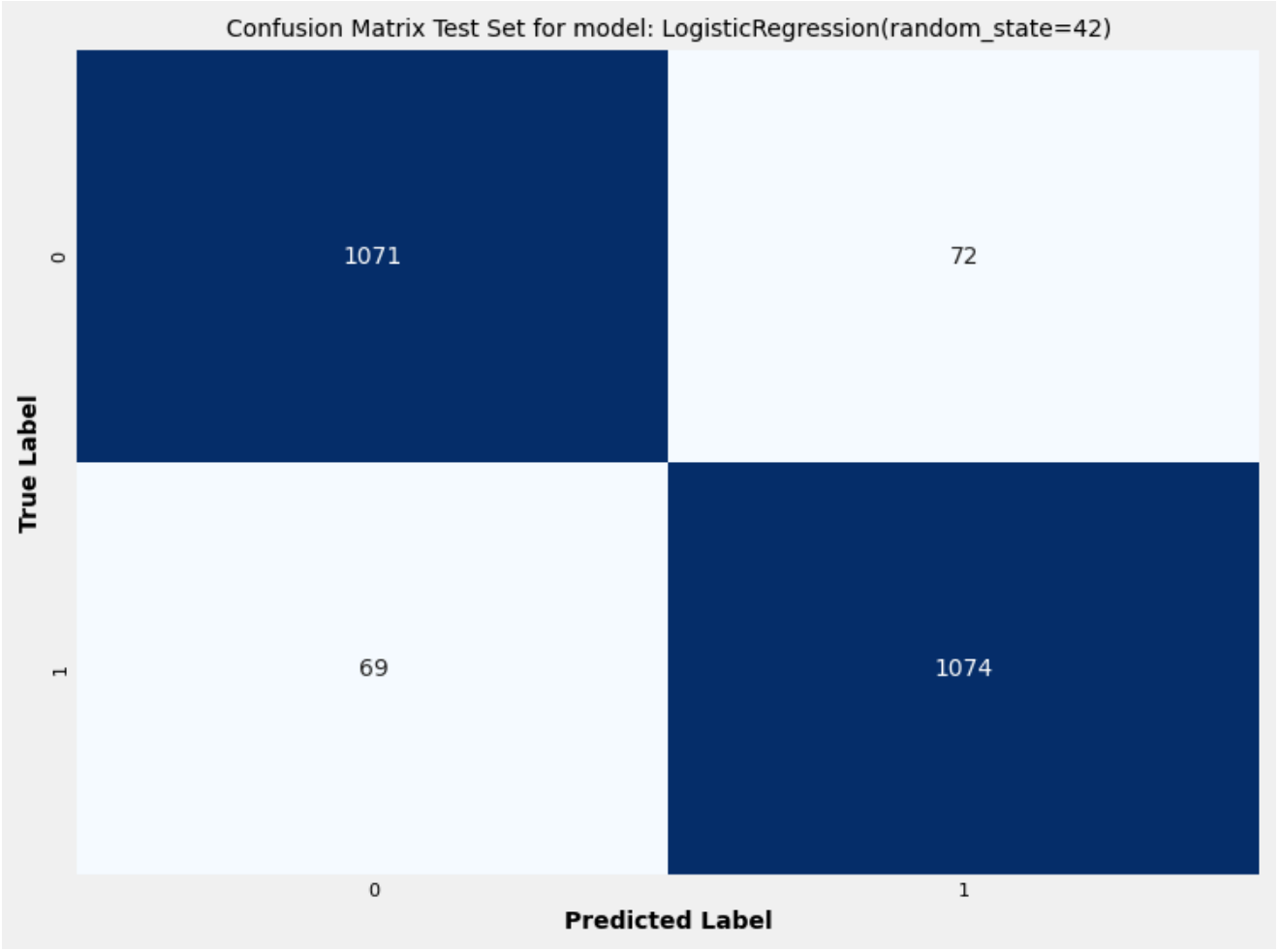



--- RISULTATI FINALI SUL TEST SET ---
Accuracy: 0.5000 Precision: 0.0000 Recall: 0.0000 F1 Score: 0.0000

Per quanto riguarda la LogisticRegression invece abbiamo che:

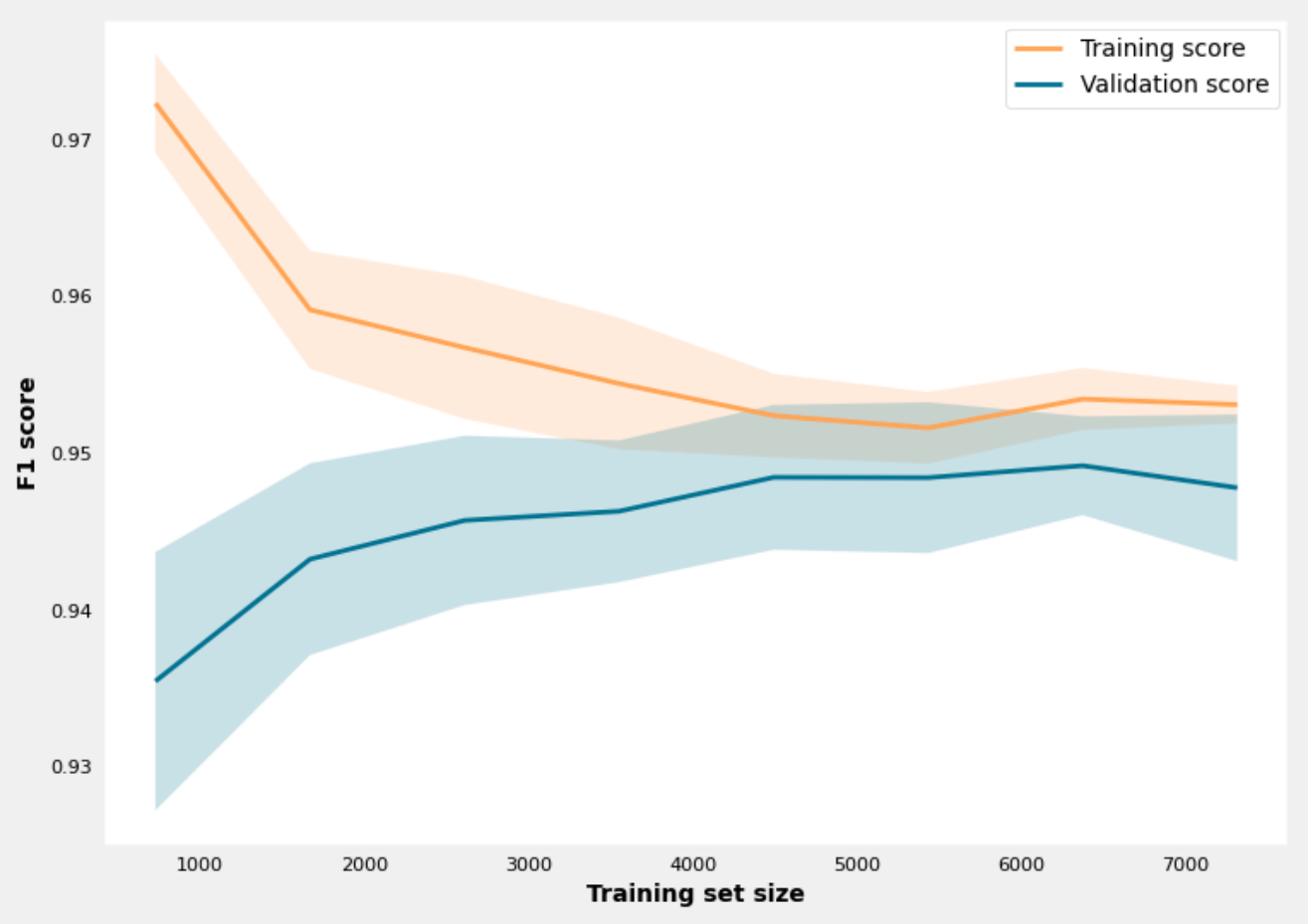



PCA: passthrough Best pipeline C: 1.0 max_iter: 100




--- RISULTATI FINALI SUL TEST SET ---

Accuracy: 0.9383 Precision: 0.9372 Recall: 0.9396 F1 Score: 0.9384




# Commmenti sui risultati della LogisticRegression

I risultati ottenuti dal classificatore di **Logistic Regression** forniscono un quadro solido e coerente delle prestazioni del modello, agendo come un'ottima (**strong**) baseline per il rilevamento di URL di phishing.

## 1. Capacità di Generalizzazione

Il modello presenta un'accuratezza sul **Training Set** del **94.79%** e sul **Test Set** del **93.83%**.

- Lo scarto ridotto (circa l'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.

## 2. Bilanciamento tra Precision e Recall

Uno degli aspetti più rilevanti è l'equilibrio quasi perfetto tra **Precision** (0.9372) e **Recall** (0.9396) sul test set.

- **Recall (Sicurezza):** Un valore del $\sim$ 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 **0.9384** sintetizza efficacemente l'ottimo compromesso raggiunto tra queste due metriche.

## 3. Stabilità e Affidabilità

La **Deviazione Standard** registrata durante la cross-validation è estremamente contenuta ($\sigma = 0.0049$).

- 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.

## 4. Considerazioni sulla Linearità del Problema

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.

---

# SVM - Kernel Lineare, Poly, RBF

Il classificatore **Support Vector Machine** è un modello di apprendimento supervisionato basato sulla ricerca dell'iperpiano di separazione ottimo in uno spazio vettoriale ad alta dimensionalità.

## 1. Formulazione Matematica (Caso Lineare)

Sia dato un dataset di addestramento $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ , dove $\mathbf{x}_i \in \mathbb{R}^d$ rappresenta il vettore delle feature e $y_i \in \{-1, 1\}$ è l'etichetta di classe. L'obiettivo della SVM è individuare un iperpiano definito dall'equazione:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

che separi le classi massimizzando il **margine** geometrico, ovvero la distanza tra l'iperpiano e i punti più vicini di ogni classe (i vettori di supporto).

Il problema di ottimizzazione per una SVM a margine "soffice" (*Soft-Margin SVM*) è formulato come segue:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$

soggetto ai vincoli:

$$y_i(\mathbf{w}^T\mathbf{x} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Dove:

- $\frac{2}{||\mathbf{w}||}$ rappresenta l'ampiezza del margine.
- $\xi_i$ sono le **variabili di slack** che permettono la classificazione errata di alcuni punti per gestire dati non perfettamente separabili (rumore).
- $C > 0$ è il parametro di regolarizzazione che controlla il trade-off tra la massimizzazione del margine e la minimizzazione dell'errore di addestramento.

# 2. Rappresentazione Duale e Kernel Trick

Attraverso l'uso dei moltiplicatori di Lagrange $\alpha_i$, il problema può essere espresso nella sua **forma duale**, che dipende esclusivamente dal prodotto scalare tra i vettori di input:

$$\max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j (\mathbf{x}_i^T\mathbf{x}_j)$$

Questa formulazione permette l'applicazione del **Kernel Trick**. Se i dati non sono linearmente separabili nello spazio originale, vengono mappati in uno spazio ad alta dimensionalità $\mathcal{H}$ tramite una funzione non lineare $\phi(\mathbf{x})$. Il prodotto scalare $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ viene sostituito da una funzione **Kernel** $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$:

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$$

# 3. Tipologie di Kernel

La scelta della funzione Kernel determina la geometria del confine di decisione (*decision boundary*). Le principali funzioni utilizzate sono:

## A. Kernel Lineare

Utilizzato quando i dati sono già linearmente separabili nello spazio delle feature originale.

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$$

## B. Kernel Polinomiale

Permette di modellare interazioni tra feature fino al grado $d$.

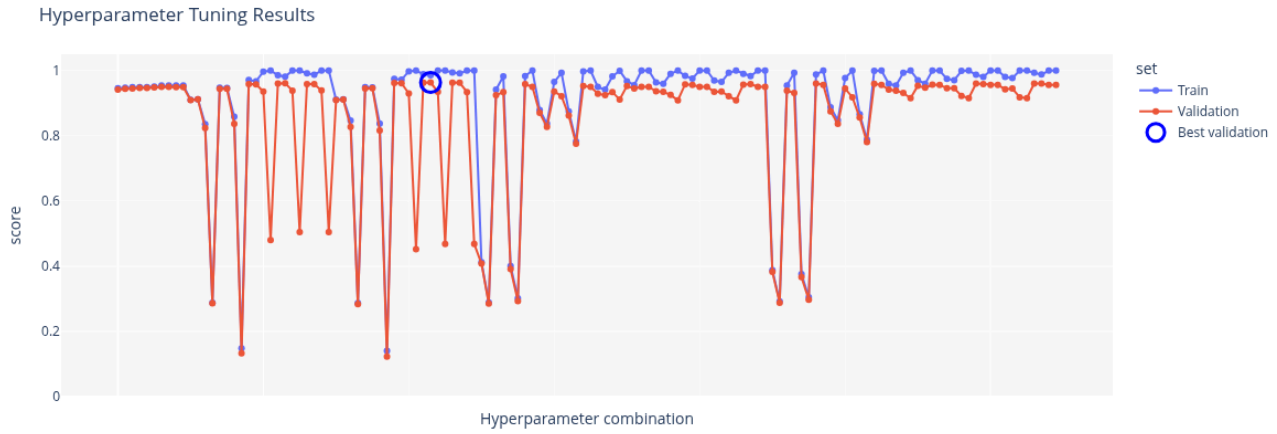
$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = (\gamma\mathbf{x}^T\mathbf{z} + r)^d$$

## C. Radial Basis Function (RBF / Gaussiano)

È il kernel più diffuso e potente per gestire relazioni non lineari complesse. Mappa i dati in uno spazio a dimensionalità infinita.

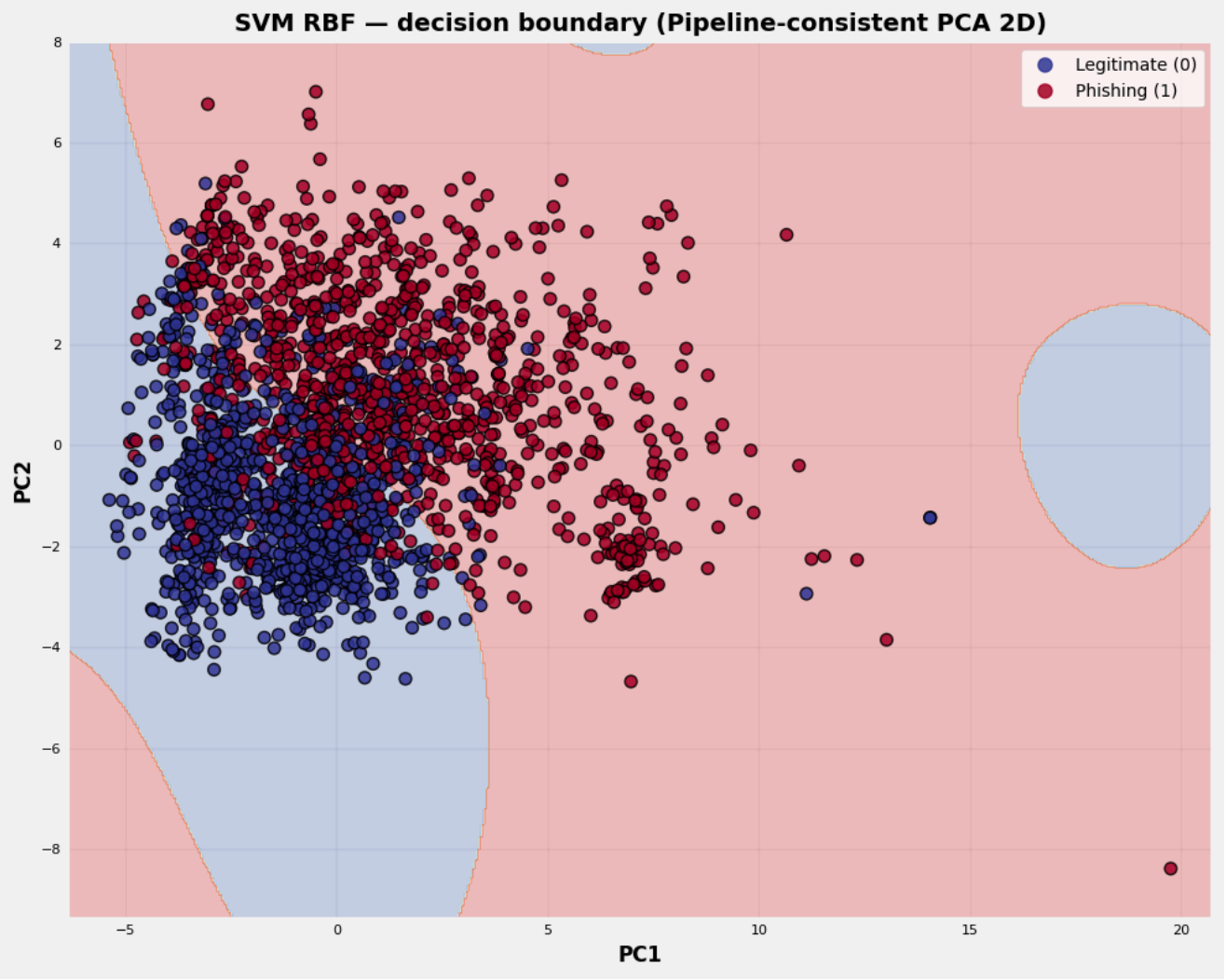
$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \exp(-\gamma||\mathbf{x} - \mathbf{z}||^2)$$

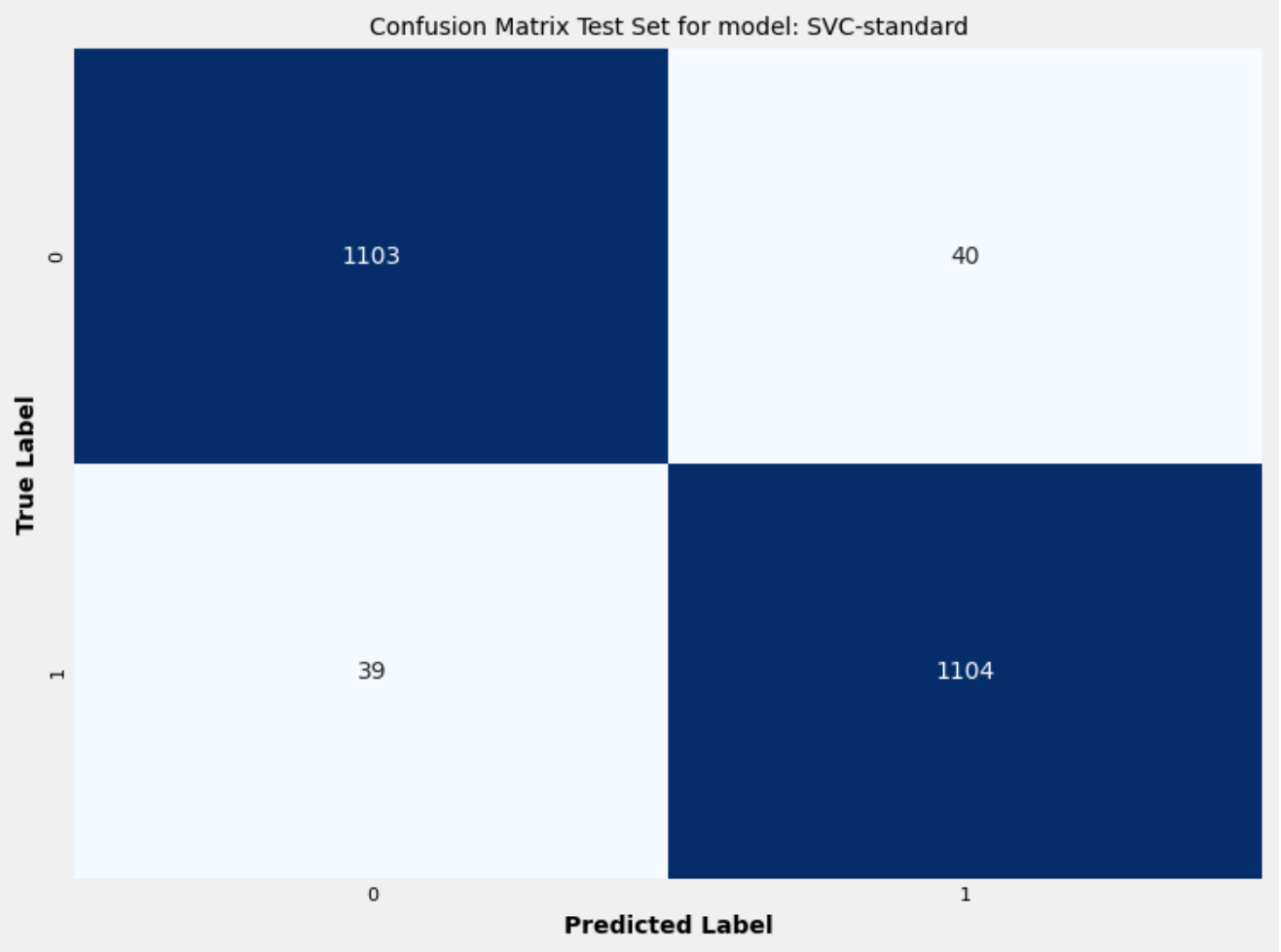
Il parametro $\gamma$ controlla l'ampiezza della campana gaussiana: valori elevati portano a un adattamento molto stretto ai dati (rischio overfitting).



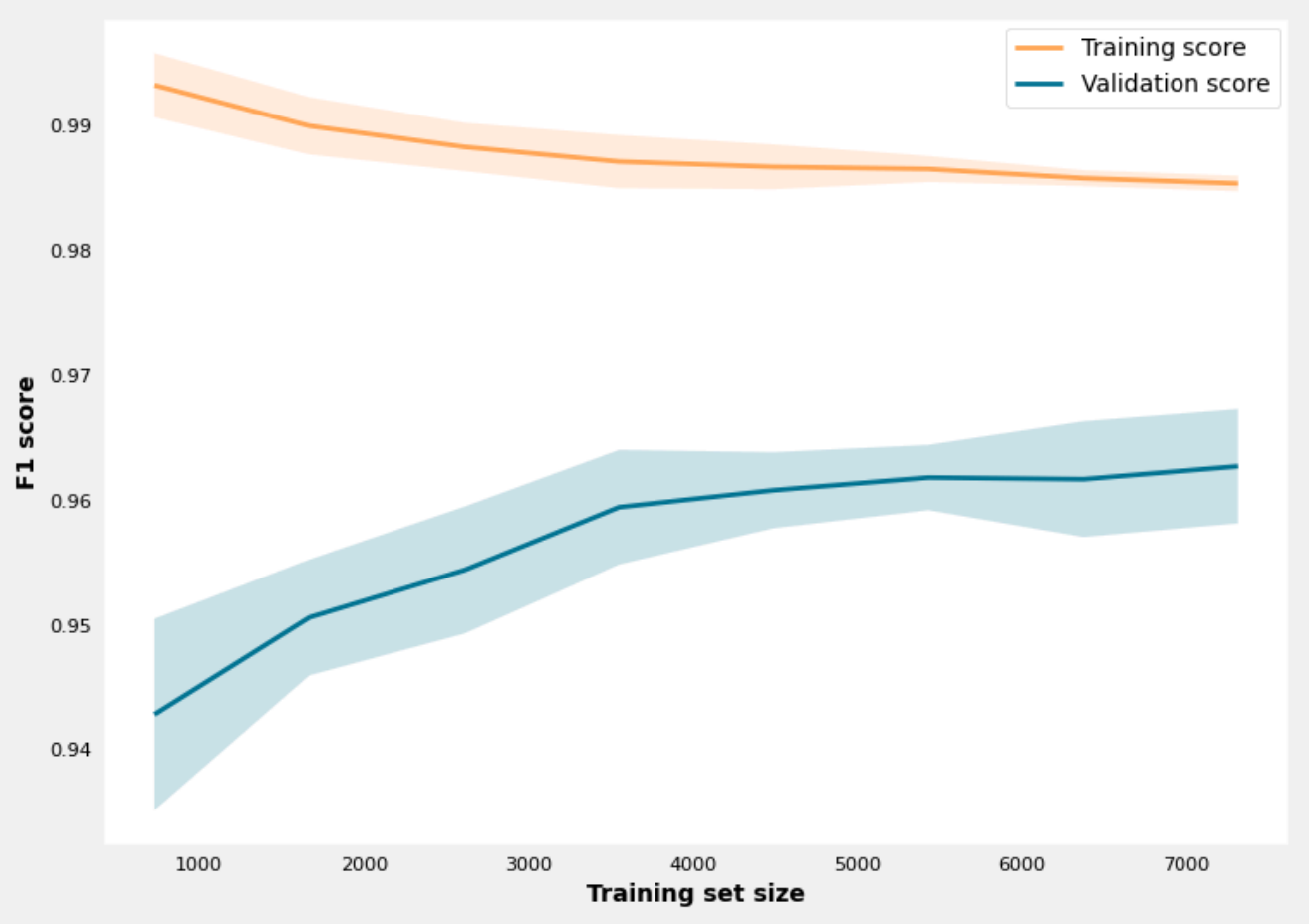

PCA: passthrough Best pipeline Kernel: rbf C: 5.0 Number of support vectors: 1477 Gamma: 0.01

Visualizzazione in 2D dei decision boundary dell'SVM


SVM RBF — decision boundary (Pipeline-consistent PCA 2D)
Legitimate (0)
Phishing (1)
PC2
PC1
8
6
4
2
0
−2
−4
−6
−8
−5
0
5
10
15
20


Confusion Matrix Test Set for model: SVC-standard
True Label
Predicted Label
1103
40
39
1104
0
1
0
1

--- RISULTATI FINALI SUL TEST SET ---
Accuracy: 0.9654 Precision: 0.9650 Recall: 0.9659 F1 Score: 0.9655




# Commmenti sui risultati della SVM

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.

## 1. Capacità di Generalizzazione

Il modello presenta un'accuratezza sul **Training Set** del 96.28% e sul **Test Set** del 96.54%.

- Lo scarto praticamente nullo 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.

## 2. Bilanciamento tra Precision e Recall

Le metriche sul test set mostrano uno sbilanciamento quasi insignificante tra **Precision** (0.9650) e **Recall** (0.9659).

- **Recall (Sicurezza)**: Un valore del 96.59% 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 0.9650 garantisce che il tasso di "Falsi Allarmi" rimanga molto basso, preservando l'esperienza d'uso dell'utente.
- **F1-Score**: Il valore di 0.9655 riflette un classificatore molto robusto, capace di gestire con successo il trade-off tra sensibilità e precisione.

## 3. Stabilità e Affidabilità

La Deviazione Standard registrata durante la cross-validation è estremamente contenuta ($\sigma = 0.0046$).

- Questo dato è particolarmente significativo: l'SVM risulta lievemente più stabile della Logistic Regression ($\sigma = 0.0049$). Le performance rimangono quasi invariate attraverso i diversi fold, a dimostrazione di una configurazione degli iperpiani di separazione molto solida.

## 4. Considerazioni sull'apporto della Non-Linearità

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) 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.

---

# Ensemble Methods - RandomForest

Il **Random Forest** è un algoritmo di apprendimento supervisionato basato su un paradigma di **Ensemble Learning** chiamato **Bagging** (Bootstrap Aggregating). Il modello consiste in una vasta collezione (foresta) di alberi di decisione non correlati, i cui risultati vengono aggregati per fornire una predizione robusta.

## 1. Architettura Ensemble: Bagging e Feature Randomness

L'obiettivo del Random Forest è ridurre la varianza del modello senza aumentarne il bias. Questo viene ottenuto attraverso due meccanismi principali:

## A. Bootstrap Aggregating (Bagging)

Dato un dataset di addestramento $\mathcal{T}$ di dimensione $n$, il modello genera $\mathcal{B}$ nuovi set di dati $\mathcal{T}_b$ (campioni bootstrap) selezionando $n$ osservazioni da $\mathcal{T}$ con reinserimento (*replacement*).
Per ogni campione $b = 1, \ldots, \mathcal{B}$, viene addestrato un albero di decisione $T_b$.

## B. Feature Randomness (Metodo dei Sottospazi Casuali)

A differenza dei normali alberi di decisione, durante la costruzione di ogni nodo di ogni albero, l'algoritmo non cerca la migliore feature tra tutte le $d$ variabili disponibili. Al contrario, seleziona un sottoinsieme casuale di feature di dimensione $m$ (tipicamente $m \approx \sqrt{d}$ per la classificazione).

Il criterio di split viene quindi calcolato solo su questo sottoinsieme:

$$\text{Best Split} = arg \max_{j \in \{1,\ldots,m\}} \Delta I(j, s)$$

# 2. Criteri di Suddivisione (Split)

Per ogni nodo, l'albero cerca di massimizzare la "purezza" dei nodi figli. I due criteri più comuni per misurare l'impurità in un task di classificazione binaria ($y \in \{0, 1\}$) sono:

1. **Indice di Gini (Gini Impurity):**

$$G = 1 - \sum_{i \in \{0,1\}} p_i^2$$

2. **Entropia (Information Gain):**

$$H = -\sum_{i \in \{0,1\}} p_i \log_2(p_i)$$

Dove rappresenta la frazione di campioni appartenenti alla classe nel nodo corrente.
Lo split ottimale è quello che massimizza la riduzione dell'impurità (Information Gain).

# 3. Aggregazione Finale

Una volta addestrati tutti i alberi, la predizione finale per un nuovo input viene ottenuta tramite **voto di maggioranza** (Majority Voting):

$$\hat{y} = \text{vote}\{T_1(\mathbf{x}), T_2(\mathbf{x}), \ldots, T_{\mathcal{B}}(\mathbf{x})\}$$

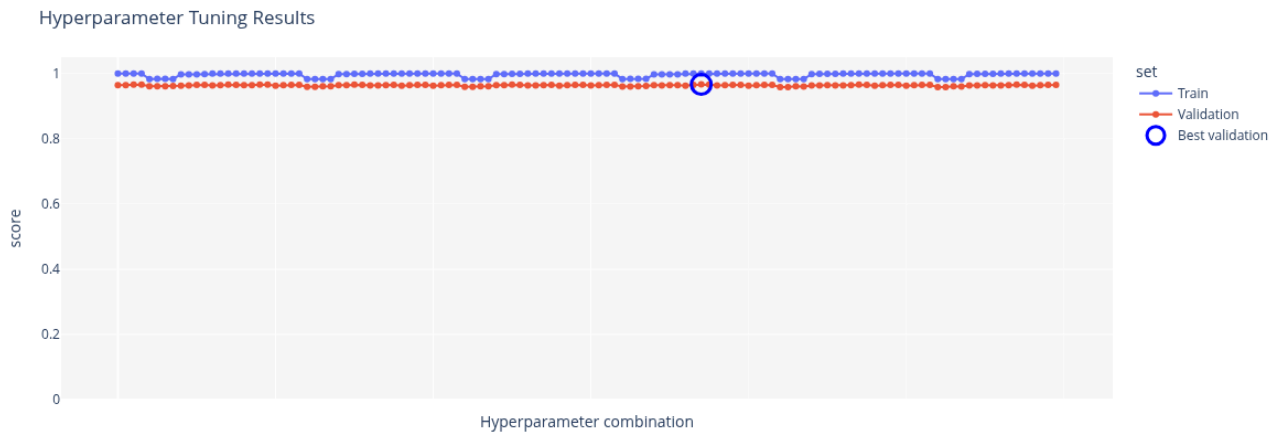
Useremo RF per classificare, ma anche per selezionare le feature che hanno impattato più del dovuto sulla classificazione

Proveremo due versioni della RandomForest, una che classifica usando dati `raw` (quindi nessun lavoro di feature engineering come fatto per la SVM), e l'altra che classifica usando log-trasformazione

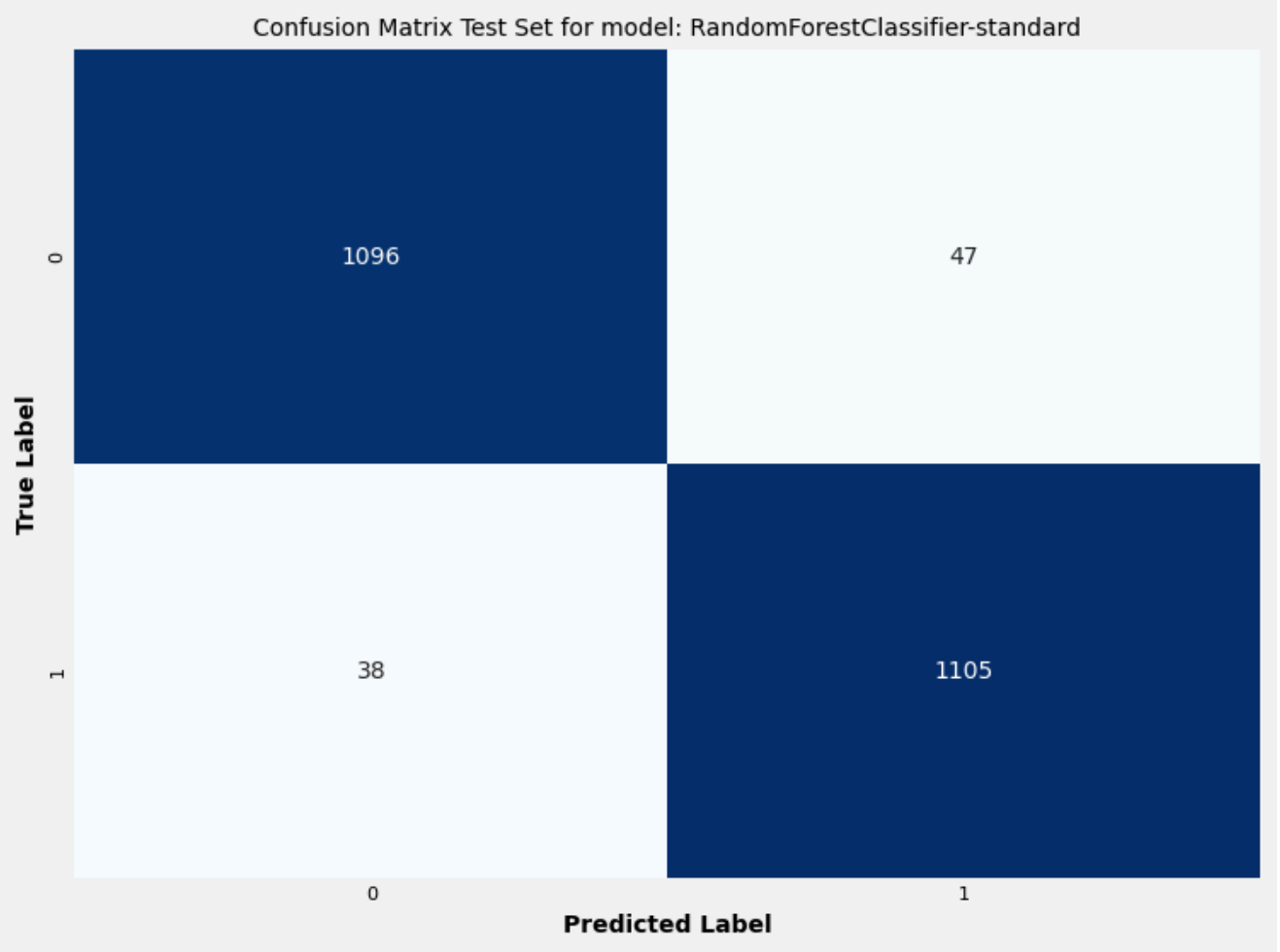
# Addestramento Random Forest standard

In questa sezione addestriamo quindi la RF con dati `raw`, ovvero dati in cui non è stata fatta nessuna operazione di **feature-engineering**

Come al solito, verrà effettuato un ***tuning*** degli iperparametri ottimali

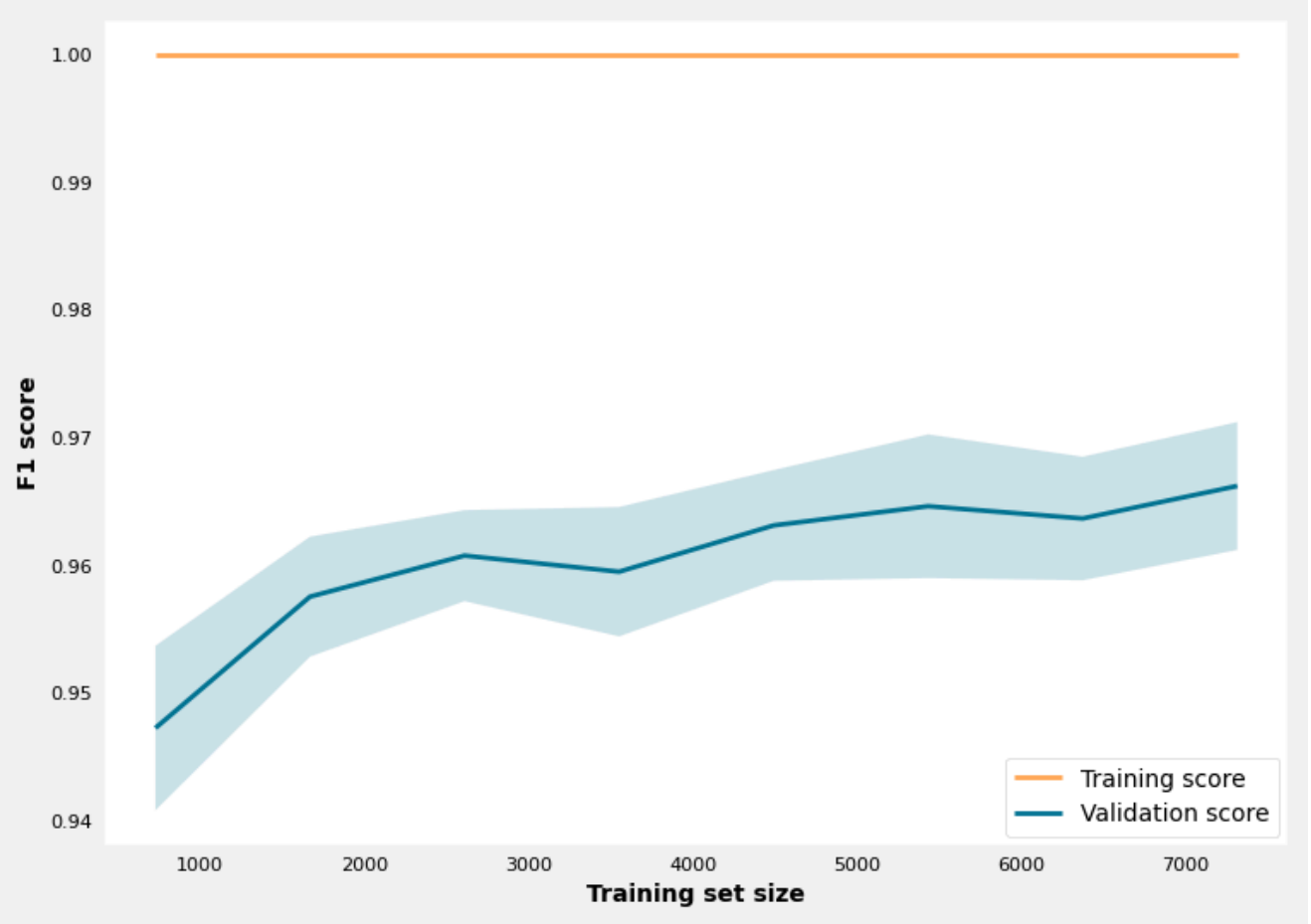



Best pipeline n_estimators: 200 max_depth: 20 criterion: gini class_weight: balanced



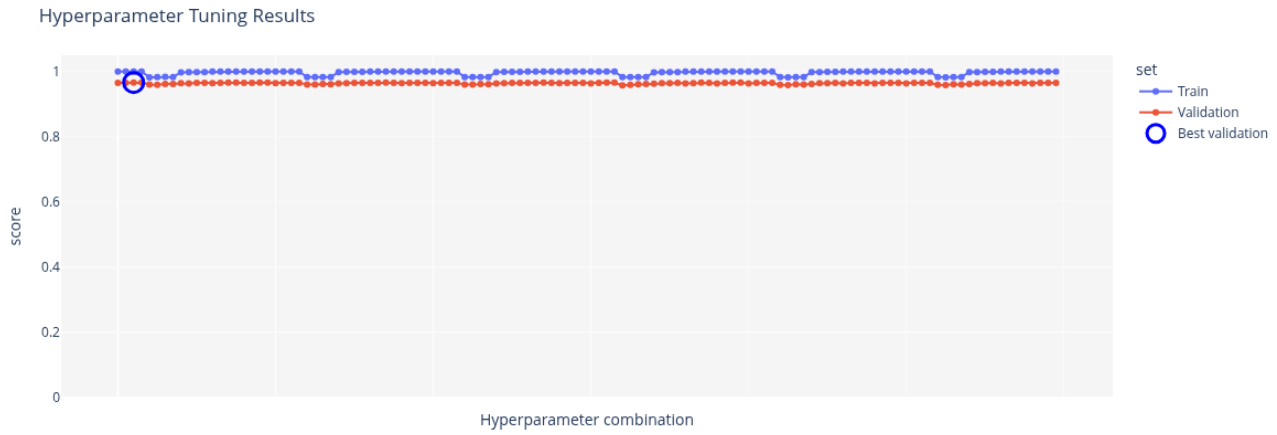

--- RISULTATI FINALI SUL TEST SET ---

Accuracy: 0.9628 Precision: 0.9592 Recall: 0.9668 F1 Score: 0.9630

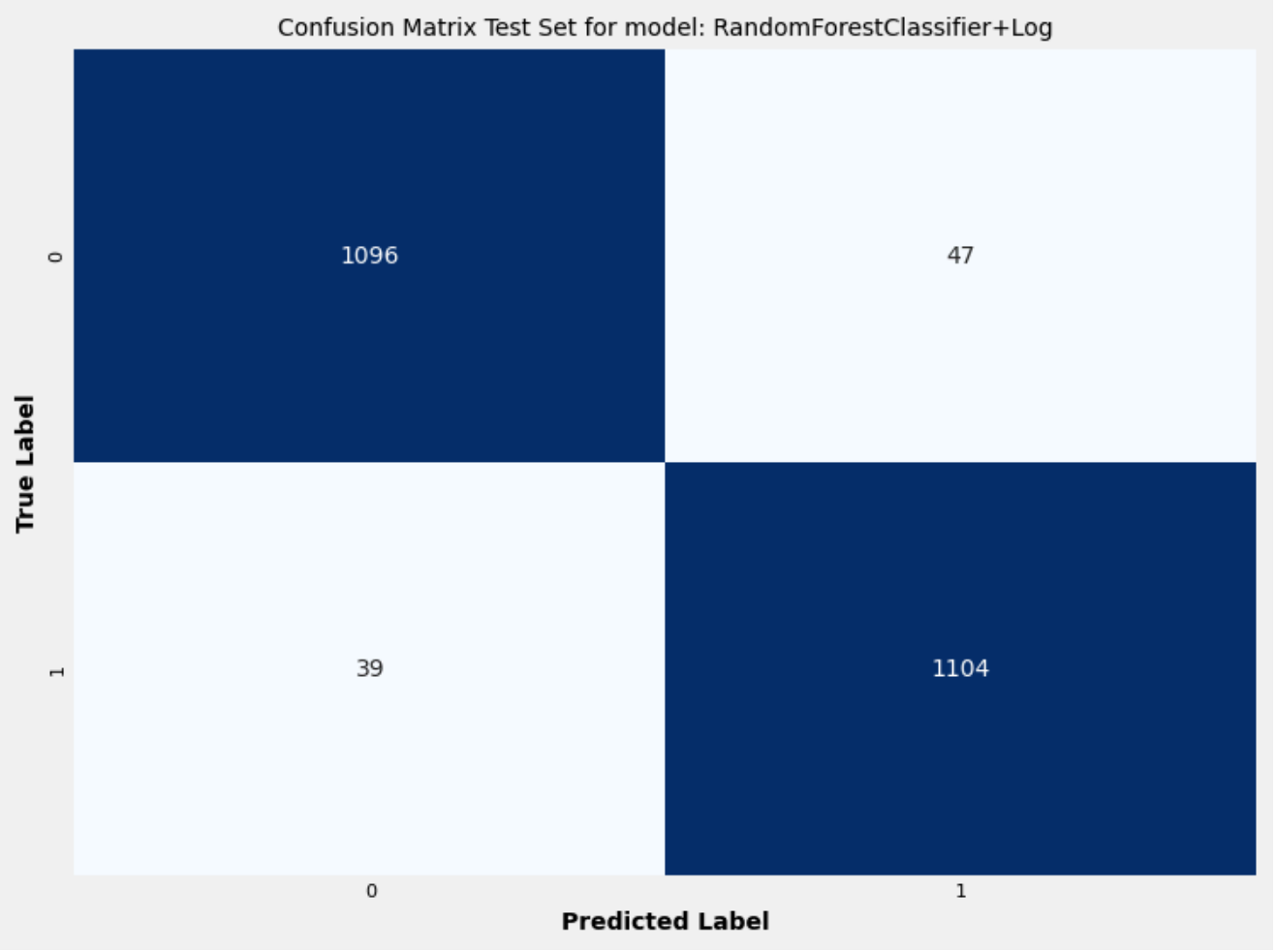



# Addestramento Random Forest con dati log-trasformati

Qui invece addestriamo la RF usando dati log-trasformati, usando il preprocessor inizializzato prima

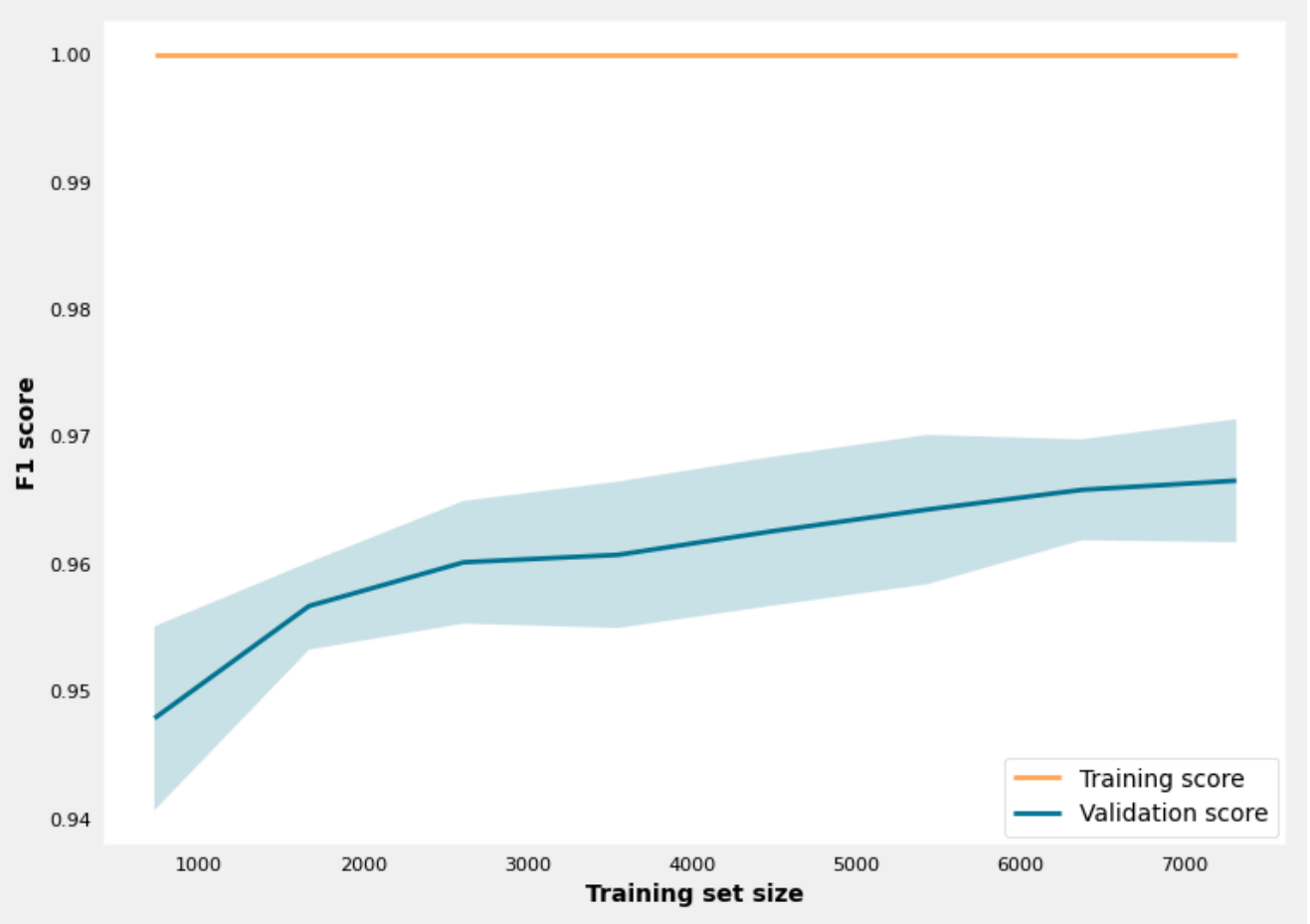



Best pipeline n_estimators: 200 max_depth: None criterion: gini class_weight: None




--- RISULTATI FINALI SUL TEST SET ---

Accuracy: 0.9624 Precision: 0.9592 Recall: 0.9659 F1 Score: 0.9625




Confrontiamo ora i due modelli di Random Forest addestrati in precedenza.

Prenderemo come "migliore" il modello che risulterà essere **più stabile**

# Commmenti sui risultati dei modelli RandomForest

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.

## 1. Eccellente Capacità di Generalizzazione

I modello presentano un'accuratezza sul **Training Set** del **96.63%** e sul **Test Set** del **96.28%** nel caso dei dati raw, e nell'altro (dati log-trasformati) un'accuratezza sul sul **Training Set** del **96.58%** e sul **Test Set** del **96.24%**.

- **Analisi dello scarto:** La differenza tra le due fasi è di circa l'**0.2/0.3%** in entrambi i modelli. Questo valore è ottimo: 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.

## 2. Analisi della Sicurezza e dell'Usabilità (Precision vs Recall)

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 $\approx$ **96.5%**, entrambi i modelli garantiscono 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 **0.9592** in entrambi i modelli 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 **0.9630** è una prova matematica della solidità del classificatore nel gestire il trade-off tra queste due necessità contrapposte.

## 3. Stabilità nelle Prestazioni

La **Deviazione Standard** registrata durante la cross-validation è molto bassa per entrambi i modelli ($\sigma = 0.0046$ per il primo e $\sigma = 0.0061$ nell'altro).

- Questo indica che entrambi i modelli risultano essere **molto stabili**, con il primo che risulta essere leggermente più stabile del secondo. Le loro 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.

## 4. Conclusioni sull'efficacia del modello

Il raggiungimento della soglia del $\approx$**96%** su tutte le metriche principali nel test set posiziona questi modelli tra i migliori candidati per l'implementazione in un sistema reale. La capacità di catturare le sottili relazioni non lineari tra le 89 feature permette di ottenere una barriera difensiva estremamente affidabile e precisa.

---

# Feature Selection: 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**.

# 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.

## 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.

## 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.

# 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?"*

## Workflow operativo

1. **Calcolo:** Eseguiamo la Permutation Importance sul training set per identificare le variabili chiave per l'SVM. Il calcolo avverrà sia sfruttando la miglior Random Forest sia sfruttando il miglior modello SVM

2. **Rimozione:** Escludiamo dal dataset le feature che hanno ottenuto il punteggio di importanza più elevato.
3. **Rivalutazione:** Riaffiniamo e testiamo nuovamente il modello SVM sullo spazio delle feature ridotto.

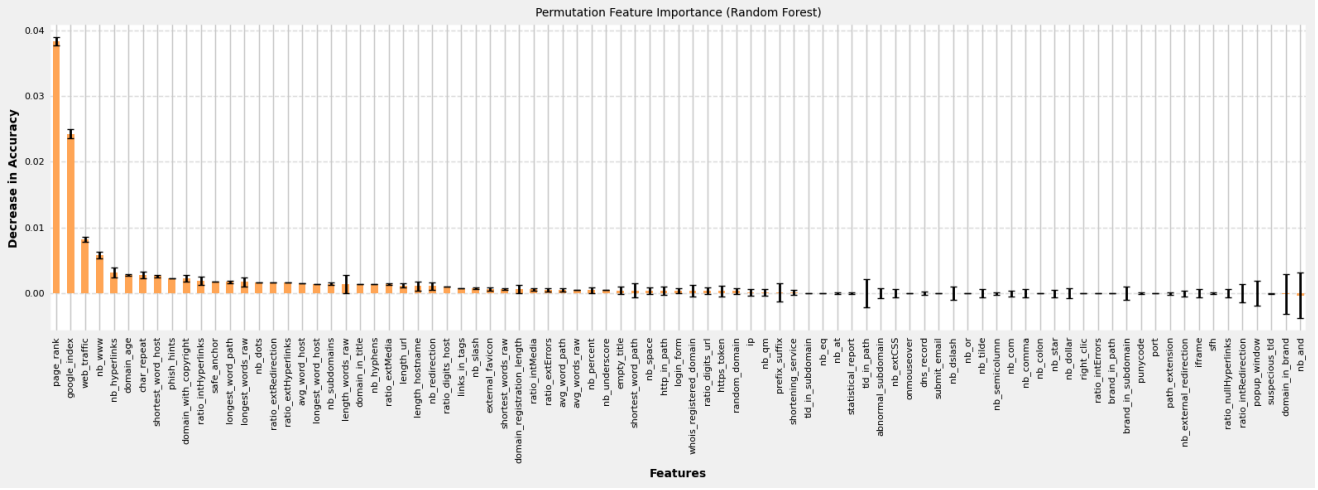
Questo processo ci permetterà di distinguere tra feature che offrono un contributo informativo unico e feature che, se rimosse, possono essere compensate da altre variabili correlate presenti nel dataset.

# Permutation Importance calcolata con Random Forest

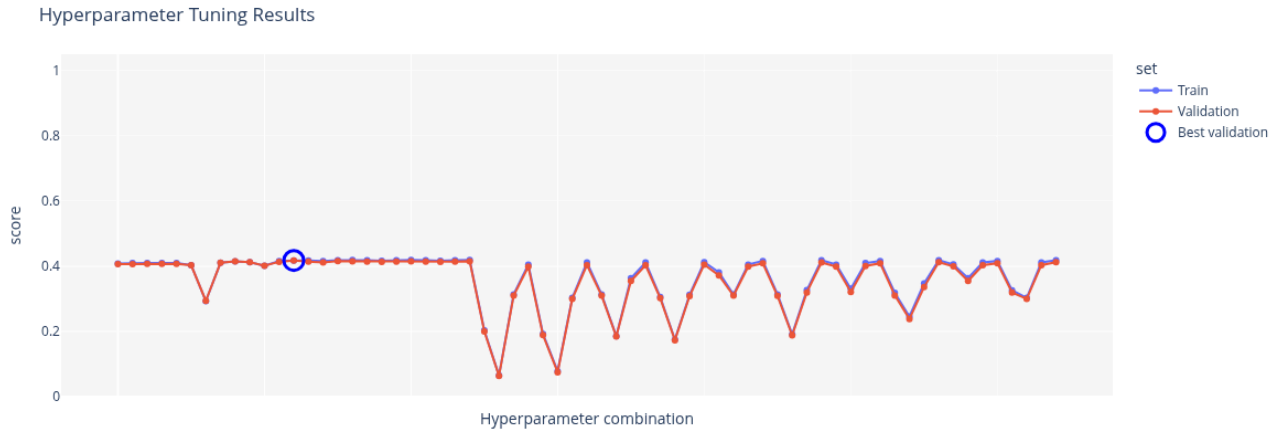
Calcoliamo la permutation importance sfruttando come modello la miglior Random Forest, in questo modo possiamo individuare i pilastri informativi del dataset.

A questo punto l'esperimento sarà:

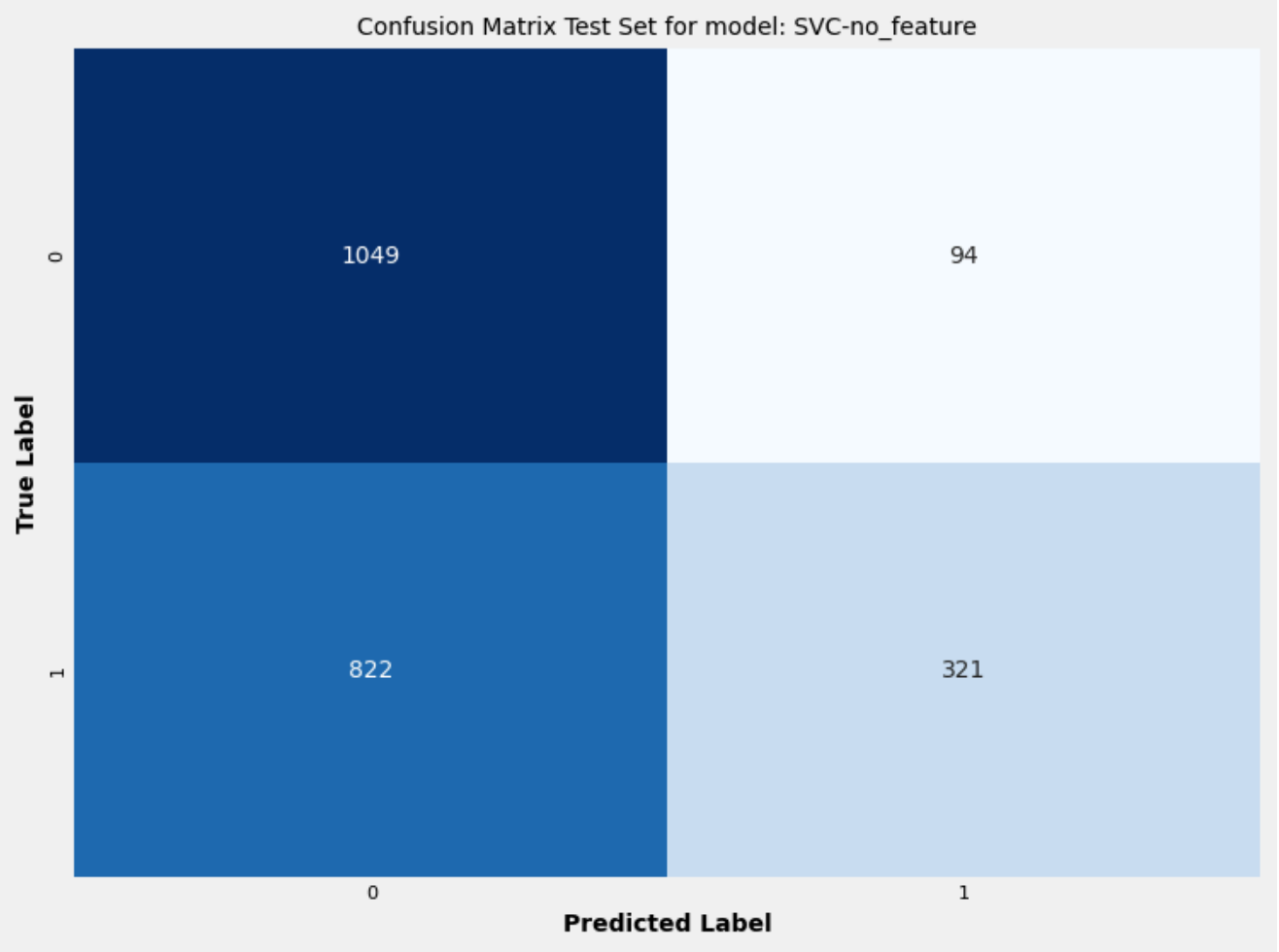
> Togliendo dal dataset ciò che il modello Random Forest individua come "pilastri", la SVM riesce ancora a cavarsela?.




Ri-addestramento della SVM con dataset ridotto tramite permutation importance, calcolata con miglior Random Forest



Best pipeline Kernel: rbf C: 0.1 Gamma: 1.0 Class weight: None Support vectors: $[3698, 3732]$ Number of support vectors: 7430




--- RISULTATI FINALI SUL TEST SET ---
Accuracy: 0.5993 Precision: 0.7735 Recall: 0.2808 F1 Score: 0.4121

Come vediamo dal grafico precedente, eliminando dal dataset quelle feature che la Random Forest ha ritenuto più "importanti", il miglior modello SVM ha avuto un calo estremo nella Recall (aumento esponenziale di Falsi Negativi), mantenendo però una Precision abbastanza elevata.

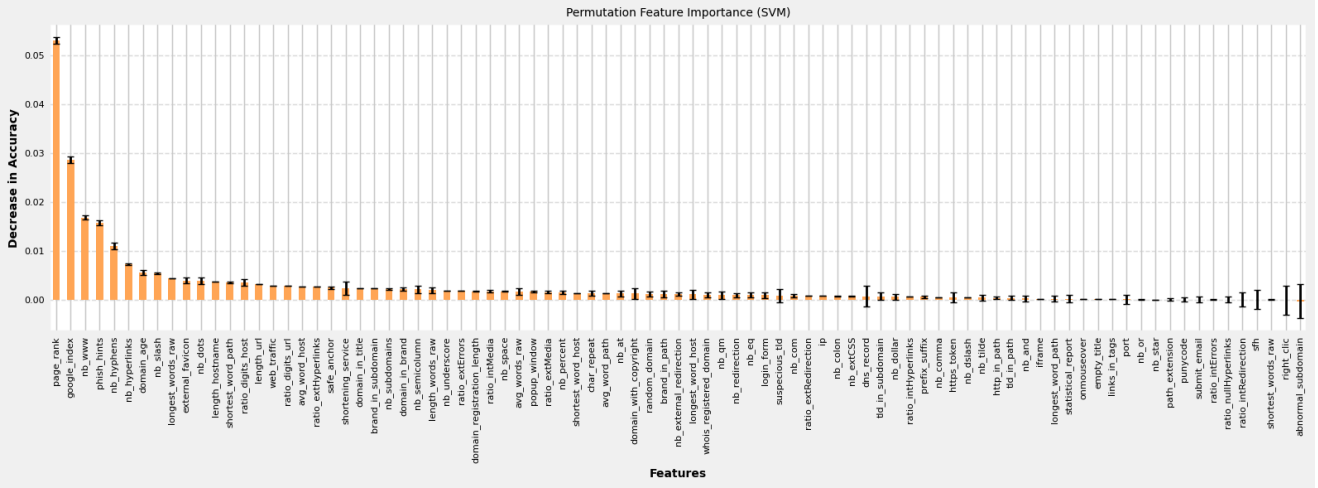
Questo è comunque un risultato $molto$ basso, perchè significa che la sicurezza degli utenti è messa troppo a rischio

## Permutation Importance calcolata con SVM

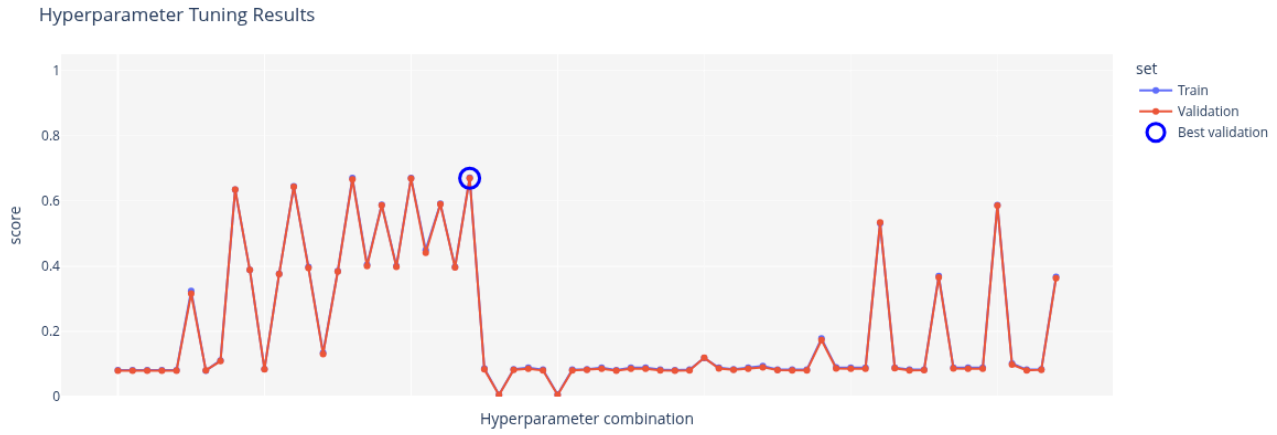
Calcoliamo la permutation importance sfruttando come modello la miglior SVM, in questo modo possiamo individuare le "leve" che la miglior SVM ha usato per classificare

A questo punto l'esperimento sarà:

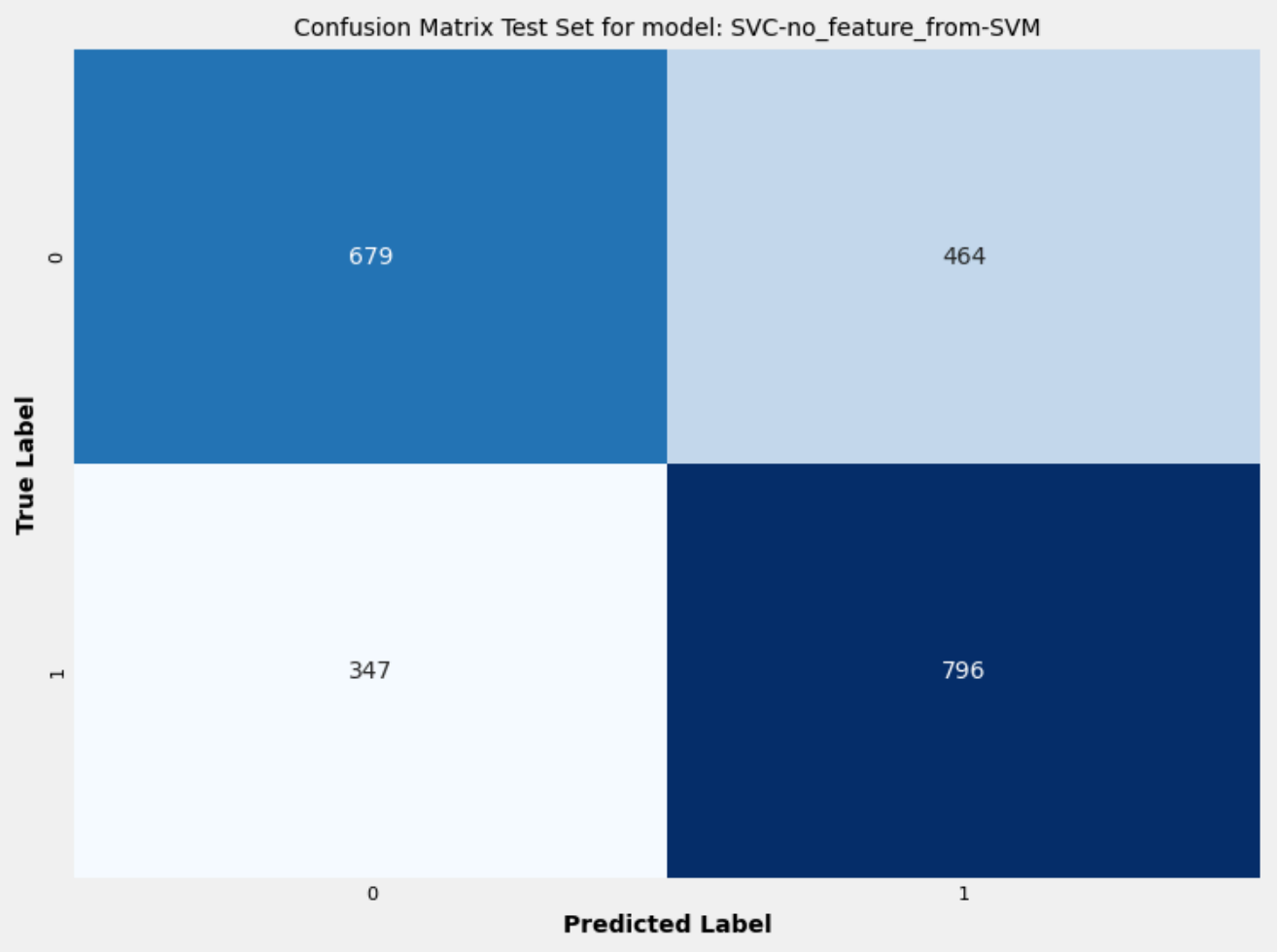
> Quali sono le feature a cui la SVM è più sensibile? Se vengono eliminate, la SVM è capace di ricostruire una logica diversa con quello che resta?




Ri-addestramento della SVM con il dataset ridotto




Best pipeline Kernel: rbf C: 10.0 Gamma: 1.0 Class weight: None Support vectors: $[3181, 3197]$ Number of support vectors: 6378




--- RISULTATI FINALI SUL TEST SET ---
Accuracy: 0.6452 Precision: 0.6317 Recall: 0.6964 F1 Score: 0.6625

---

# Analisi dello Stress-Test: Impatto della Rimozione delle Feature Dominanti

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.

## 1. Crollo della Capacità Predittiva

Il modello passa da:

- un'accuratezza del **96.50%** a circa il **60.00%** se prendiamo la Permutation Importance calcolata tramite la RandomForest
- un'accuratezza del **96.50%** a circa il **64.50%** se prendiamo la Permutation Importance calcolata tramite il miglior SVM

**Perdita di Informazione:** Un calo del ≈ **35%** 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 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.

# 2. Il Collasso della Recall (Criticità di Sicurezza)

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

- **0.2808** (Permutation Importance da RandomForest).
- **0.6964** (Permutation Importance da SVM).

**Phishing Mancati:** Una Recall del 69% significa che il modello ora **manca circa 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. Risulta però comunque più accettabile rispetto ad una Recall del 28%, che significa che il modello **manca 8 attacchi di phishing su 10**, il che significa che l'80% delle volte sbaglia a classificare una possibile minaccia informatica. Questo risultato è ovviamente puramente sperimentale, tale modello non potrebbe mai essere accettato come classificatore per tale contesto

**Sbilanciamento verso i Falsi Negativi:** Mentre la Precision tiene meglio (0.7735 da un lato e 0.6317 dall'altro), 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.

# 3. Analisi della F1-Score

L'**F1-Score** si attesta a **0.6625** in un caso e **0.4121** nell'altro.

- Questo valore rappresenta un calo drastico rispetto al precedente **0.9655**. 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 (solo se consideriamo il primo caso, altrimenti nel secondo caso il classificatore si trova addirittura al di sotto del caso fortuito).

## 4. Conclusioni sull'Importanza delle Feature

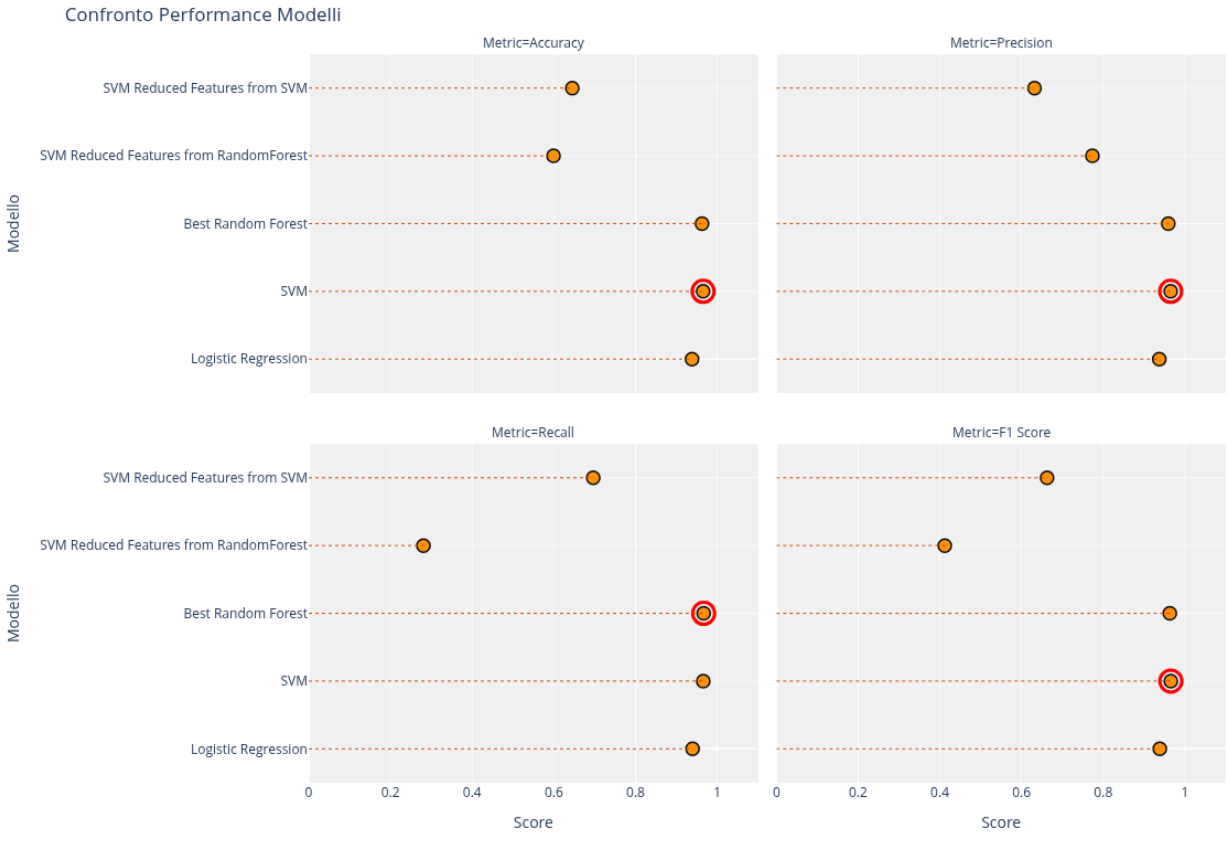
Questo esperimento di "ablazione" fornisce la prova definitiva della validità della nostra analisi di feature importance:

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.

---

# Plot dei risultati

Di seguito, il plot dei risultati sul TestSet di ogni modello provato




---

# 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**).

# 1. Tabella Riassuntiva delle Performance (Test Set)

| Modello | Accuracy | Precision | Recall | F1-Score | Stabilità ($\sigma$) |
|---|---|---|---|---|---|
| **Logistic Regression** | 93.83% | 0.9372 | 0.9396 | 0.9384 | 0.0049 |
| **SVM (Kernel RBF)** | **96.54%** | 0.9650 | 0.9659 | 0.9655 | **0.0046** |
| **Random Forest** | 96.28% | 0.9592 | ***0.9668*** | 0.9630 | 0.0046 |

*(Nota: I modelli non lineari hanno raggiunto prestazioni di picco identiche sul test set, evidenziando il raggiungimento di un limite superiore di separabilità per questo specifico dataset).*

# 2. Analisi Comparativa

## A. La Forza della Baseline (Logistic Regression)

La **Logistic Regression** ha dimostrato che il problema del phishing possiede una forte componente di **separabilità lineare**. Ottenere un'accuratezza del $\approx 94\%$ con un modello così semplice valida la qualità del preprocessing e delle 89 feature estratte. Rappresenta la scelta ottimale in scenari dove la velocità di esecuzione è più critica della precisione estrema.

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

Il passaggio a modelli non lineari ha permesso di recuperare un ulteriore **3%** di accuratezza. In ambito cybersecurity, questo incremento si traduce nel blocco di migliaia di potenziali minacce aggiuntive.

- **SVM** si è distinto per la **massima stabilità** ($\sigma = 0.0046$), dimostrando di essere il modello meno influenzato dal rumore nei dati.
- **Random Forest** ha mostrato un'ottima capacità di adattamento, sebbene con una tendenza leggermente superiore alla memorizzazione del training set rispetto alla SVM.

# 3. Validazione tramite Stress Test (Ablazione Feature)

Per verificare quanto il modello dipenda realmente dalle feature individuate tramite la **Permutation Importance**, abbiamo condotto uno stress test sul miglior modello SVM, rimuovendo le variabili più importanti. I risultati hanno confermato l'importanza critica di tali feature:

- Permutation Importance ottenuta tramite Random Forest:

- **Accuracy:** crollata dal 96.54% al **59.93%**.
- **Recall:** collassata a **0.2808** (il modello manca 8 attacchi su 10).

- Permutation Importance ottenuta tramite Random Forest:

- **Accuracy:** crollata dal 96.54% al **64.52%**.
- **Recall:** collassata a **0.6964** (il modello manca 1 attacco su 3).
  Questo esperimento di ablazione fornisce la prova definitiva che il potere decisionale è concentrato in un ristretto sottoinsieme di variabili insostituibili. La totale incapacità delle restanti feature di compensare la perdita dimostra che il segnale del phishing è netto e localizzato.

# 4. Considerazioni sulla Sicurezza: La Recall

La metrica critica del progetto, la **Recall**, è stata elevata dai modelli avanzati dal 94% al **96.68%**. Questo miglioramento riduce drasticamente il rischio di **Falsi Negativi**, garantendo che l'utente finale sia protetto con un'efficacia superiore rispetto a un approccio lineare standard.

# 5. Verdetto Finale

Il modello **SVM con kernel RBF** è quello consigliato per la messa in produzione per tre ragioni chiave:

1. **Stabilità Massima:** La deviazione standard più bassa garantisce che il modello si comporti in modo prevedibile su diversi set di dati.
2. **Affidabilità:** Presenta il miglior bilanciamento tra performance e generalizzazione (minor rischio di overfitting rispetto a Random Forest).
3. **Robustezza Provata:** Lo stress test ha dimostrato che il modello ha imparato correttamente i segnali fondamentali della minaccia.

---

---

# Appendice: Spiegazione StandardScaler di scikit.learn

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

## 1. Definizione Matematica

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

1. **Media Campionaria ($\mu_j$):**

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

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

2. **Deviazione Standard Campionaria ($\sigma_j$):**

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

## La Trasformazione (Z-score)

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

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

**Proprietà Risultanti**

Dopo la trasformazione, la nuova distribuzione della feature $j$ (chiamiamola $Z$) avrà:

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

Ovvero

$$Z \sim \mathcal{N}(0, 1)$$

# 2. Perché usarla nel progetto?

Nel dataset abbiamo feature con unità di misura e ordini di grandezza totalmente diversi.

Ad esempio:

- `web_traffic`: valori che possono arrivare a milioni.
- `nb_dots`: valori piccoli, solitamente tra 1 e 10.
- `domain_age`: valori in giorni (migliaia).

Senza `StandardScaler`, ci ritroveremo in situazioni poco piacevoli, che potrebbero portare ad un'addestramento sbagliato e di conseguenza una predizione sbagliata

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

La **SVM** calcola distanze euclidee.

Se una feature ha valori enormi (es. traffico web), essa dominerà il calcolo della distanza, rendendo le feature piccole (es. numero di punti nell'URL) praticamente invisibili al modello, anche se queste ultime sono più importanti per scovare il phishing.

La standardizzazione "democratizza" le feature: tutte pesano allo stesso modo all'inizio del training.

### B. Requisito Fondamentale per la PCA

Nella pipeline abbiamo inserito l'opzione di **PCA** per ridurre la dimensionalità.

La PCA cerca le direzioni di massima varianza.

Se non si standardizza, la PCA identificherà come "componente principale" semplicemente la feature con i numeri più grandi, poiché la varianza è sensibile alla scala.

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

Mentre il `MinMaxScaler` schiaccia i dati tra 0 e 1, lo `StandardScaler` è preferibile in questo caso perché:

1. **Gestione Outlier:** Il phishing ha spesso outlier estremi (es. URL lunghissimi). Lo `StandardScaler` non ha un limite massimo predefinito, quindi non "comprime" troppo gli outlier, permettendo al modello di riconoscerli come anomalie.
2. **Distribuzione Gaussiana:** Molti algoritmi (specialmente SVM) assumono implicitamente che i dati siano distribuiti in modo approssimativamente gaussiano e centrati sullo zero.