Università
Ca'Foscari
Venezia

10-fold cross-validation

Confronto tra vari classificatori

Alessio Marco Rinaldo

December 19, 2024

Università Ca'Foscari Venezia

Contents

1 Introduzione **2**

2 Trasformazione TF-IDF **3**

3 SVM (Support Vector Machine) **4**
3.1 Kernels 5
3.2 Similarità del coseno 6

4 Random Forest **7**
4.1 Decision trees 7
4.2 Random forests 7

5 Naive Bayes **8**
5.1 Modelli discriminativi vs generativi 8
5.2 ESEMPIO: (classificatore di mail) 8

6 K-NN (K-Nearest Neighbors) **9**
6.1 Theorem 9
6.2 Principio di funzionamento 9

7 10 Fold Cross Validation **11**
7.1 Procedimento 11

8 Analisi risultati **12**
8.1 Accuratezze della cross-validation 12
8.2 Tempi di validazione 12
8.3 Addestramento e testing 13

1 Introduzione

2 Trasformazione TF-IDF

TF-IDF sta per Term Frequency-Inverse Document Frequency, è una tecnica utilizzata per valutare la rilevanza di una parola in un documento rispetto all'insieme dei documenti utilizzati come dataset. Si basa su due principali assunzioni logiche:

1. Term Frequency: se una parola compare spesso in un documento, la sua rilevanza sarà maggiore (aumenta la probabilità che quel documento sia incentrato proprio su quella parola). Volendo calcolare la TF di un termine i in un documento j:

$$TF(i,j) = f_{ij} \tag{1}$$

   con f_{ij} frequenza relativa del termine i nel totale delle parole presenti nel documento j

2. Inverse Document Frequency: le parole che appaiono spesso in tutti i documenti avranno una rilevanza minore (aumenta la probabilità che quelle parole non siano specifiche per un documento ma abbiano uno scopo sintattico, senza contenuto semantico). Volendo calcolare la IDF di un termine i in un set di N documenti:

$$IDF(i) = ln\left(\frac{N}{n_i}\right) \tag{2}$$

   con n_i numero di documenti del set in cui appare il termine i

La trasformazione TF-IDF dell'i-esima parola nel j-esimo documento è la moltiplicazione di questi due valori:

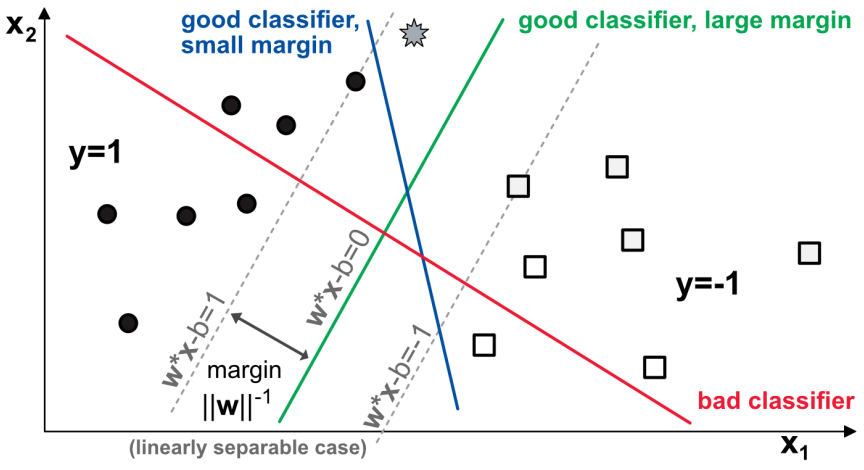
$$\boxed{TF - IDF(i,j) = TF(i,j) * IDF(i) = f_{ij} * ln\left(\frac{N}{n_i}\right)} \tag{3}$$

3 SVM (Support Vector Machine)

Il Support Vector Machine è un classificatore utilizzato soprattutto in caso di dati classificabili binariamente (come nel nostro caso SPAM e NON-SPAM) che cerca di trovare un iperpiano separatore tra le due classi tale che massimizza il margine.

- **x** = vettore che identifica un punto nello spazio delle feature;
- **w** = vettore dei pesi associati ad ogni punto x, normale all'iperpiano stesso ne stabilisce l'orientamento nello spazio e stabilisce il peso di ciascuna feature x_i;
- **b** = termine di bias, descrive lo spostamento dell'iperpiano rispetto all'origine del sistema di coordinate;
- **y** = vettore delle label, contiene tutti i valori desiderati per i punti dello spazio (la classe a cui un punto dovrebbe appartenere);
- **Iperpiano**: generalizzazione del concetto di piano quando le dimensioni in gioco sono superiori a 3. Nel caso 2D, un iperpiano separatore è una retta; in 3D un iperpiano separatore è un piano mentre per 4D o superiori si generalizza con il termine iperpiano. L'equazione dell'iperpiano nello spazio delle feature è:
$$w \cdot x + b = 0 \tag{4}$$
- **Margine**: Distanza tra l'iperpiano e i punti appartenenti alle varie classi. L'equazione del margine nello spazio delle feature è:
$$\gamma_x = (x \cdot w + b)y_x \tag{5}$$
dove y_x è la label associata al punto identificato dal vettore x. Se il termine tra parentesi e la label sono di segno concorde, il margine risulta positivo, negativo altrimenti. L'obiettivo è quello di arrivare ad avere solo margini positivi, il che vorrebbe dire aver classificato correttamente tutti i punti.

Date queste definizioni, è possibile trovare infiniti iperpiani che separano il set di dati. Per trovare il migliore tra essi, si introduce il concetto di massimizzazione del margine.



Data l'Equazione 5 e sapendo che i margini devono essere tutti positivi, si può normalizzare tale condizione per semplificare il problema in modo tale da ottenere una disequazione del tipo:

$$\gamma_x = (x \cdot w + b)y_x \geq 1$$

dove

$\longrightarrow$ $(x \cdot w + b)y_x = 1$ è soddisfatta da tutti i punti più vicini all'iperpiano (**Support Vector**);

$\longrightarrow$ $(x \cdot w + b)y_x > 1$ è soddisfatta da tutti gli altri punti

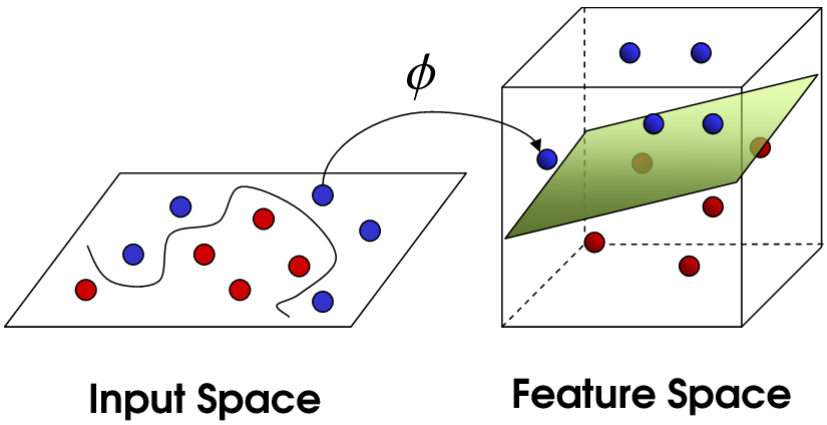
Si può quindi definire il **margine geometrico** come segue:

$$\gamma_{geom} = \frac{\gamma_x}{||w||} = \frac{1}{||w||} = ||w||^{-1}$$

nel caso x sia un Support Vector.
L'obiettivo diventa quindi quello di massimizzare il margine geometrico, o analogamente minimizzare $||w||$ (oppure minimizzare $\frac{1}{2}||w||^2$, forma concettualmente equivalente ma più semplice da trattare nei problemi di ottimizzazione in quanto la sua derivata è lineare).

3.1 Kernels

Tutto il discorso fatto è valido qualora i dati siano linearmente separabili. Per molti problemi reali però i dati NON sono linearmente separabili nello spazio delle feature. In questi casi si ricorre al cosiddetto "Metodo dei kernel", che consiste nell'aumentare il numero di dimensioni dello spazio delle feature (nel quale i dati non erano linearmente separabili) per fare in modo che nella nuova dimensione i dati siano linearmente separabili:



Si ricorre ad una funzione (in genere **NON LINEARE**) ϕ che mappa ogni punto x nel nuovo spazio a maggiori dimensioni. Tuttavia, aumentando il numero di dimensioni (ipoteticamente anche utilizzandone infinite) fa diventare proibitivo il costo computazionale dell'algoritmo. Al fine però di addestrare il modello attraverso il training set e prevedere la classificazione di nuovi dati dal test set tutto ciò che occorre sono prodotti scalari tra vettori. È possibile quindi definire una funzione di kernel:

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

Ciò permette di calcolare direttamente il prodotto scalare (che è quello che effettivamente occorre) tra i vettori nello spazio ad alta dimensione utilizzando i vettori originali x_i e x_j e non le loro trasformazioni $\phi(x_i)$ e $\phi(x_j)$.
Tra i kernel più usati:

1. **Linear**: $K(x_i, x_j) = x_i \cdot x_j$
2. **Polynomial**: $K(x_i, x_j) = (1 + x_i \cdot x_j)^n$
3. **Radial Basis Function**: $K(x_i, x_j) = \exp\left(-\frac{1}{2}\frac{\|x_i - x_j\|^2}{\sigma^2}\right)$

3.2 Similarità del coseno

Sapendo che per definizione il prodotto scalare tra due vettori viene calcolato come:

$$x_i \cdot x_j = ||x||\,||y||\,cos(\theta)$$

dove θ è l'angolo compreso tra i due vettori. Se questi due vettori vengono normalizzati (e hanno quindi norma unitaria), il loro prodotto scalare sarà funzione del solo angolo θ.

$$\hat{x} = \frac{x}{||x||} \qquad \hat{y} = \frac{y}{||y||}$$

Il kernel angolare diventa quindi:

$$K(\hat{x}, \hat{y}) = \hat{x} \cdot \hat{y} = \frac{x \cdot y}{||x||\,||y||} = cos(\theta) = K(\theta)$$

4 Random Forest

4.1 Decision trees

Un albero decisionale è un grafo composto da nodi (**features**), rami (possibili valori della feature), foglie (**classe** di appartenenza attesa).

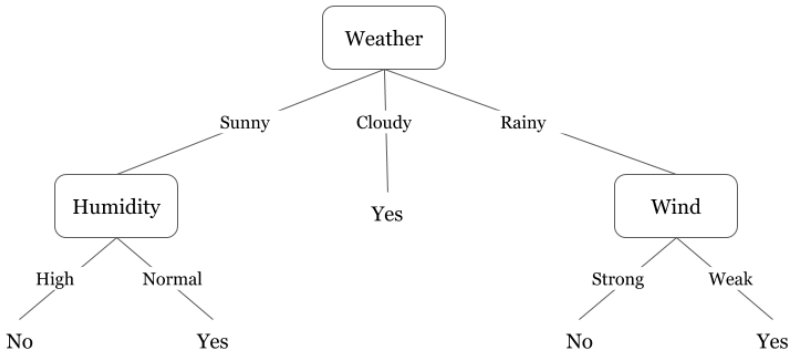


Figure 1: Esempio di un albero decisionale

Vi è una feature principale (**radice**) e delle feature secondarie che derivano dalla principale. L'algoritmo di scelta parte sempre dalla radice e si ferma quando trova una foglia.
L'obiettivo dell'algoritmo è massimizzare la separazione delle classi (nell'Albero 1 le classi sono Yes/No) con il minor numero di decisioni possibili. Un grosso vincolo di questo algoritmo decisionale è che ad ogni scelta della feature successiva si "oscura" tutto il resto dell'albero: ciò vuol dire che ad esempio non sarebbe possibile predire avendo l'Albero 1 la label di un nuovo dato avente come features:

1. **Weather** = Sunny
2. **Wind** = Weak/Strong

in quanto come feature successiva a "Weather" è presente solo "Humidity".

4.2 Random forests

Le foreste casuali sono insiemi di alberi decisionali costruiti in modo casuale.
Bootstrapping: ogni albero viene addestrato solo su un sottoinsieme dei dati di training. Il sottoinsieme viene estratto casualmente dal set di dati, è un'estrazione con sostituzione quindi dei dati possono comparire su più alberi diversi mentre altri possono non essere mai estratti.
Out-of-Bag (OOB): circa $\frac{1}{3}$ dei dati del campione di training non vengono estratti, ma utilizzati successivamente per validazione.
Feature bagging: ad ogni nodo, l'algoritmo dovrebbe selezionare la feature che meglio separa i dati di training. Così facendo però, molti alberi separati tenderanno a preferire le stesse feature dominanti, creando un'importante correlazione nella foresta. Per ovviare a questo problema, ad ogni nodo viene considerato solo un campione casuale di possibili feature tra tutte quelle disponibili e la feature migliore di quel campione sarà quella scelta.

5 Naive Bayes

5.1 Modelli discriminativi vs generativi

Il Naive Bayes è un modello di classificazione generativa che si basa sul Teorema di Bayes e sull'ipotesi di indipendenza condizionale tra le caratteristiche date le classi. In pratica, si assume che, dato l'appartenenza a una classe specifica, ogni caratteristica sia indipendente dalle altre. Questa assunzione semplifica enormemente il calcolo delle probabilità, rendendo il modello molto efficiente.

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{6}$$

dove

1. $P(y|x)$ = probabilità che l'elemento x (vettore delle feature) appartenga alla classe y
2. $P(x|y)$ = probabilità che l'elemento a cui corrisponde la classe y sia l'elemento x
3. $P(y)$ = probabilità che un elemento qualsiasi sia della classe y
4. $P(x)$ = probabilità che un elemento qualsiasi sia l'elemento x

Fondamentale è l'assunzione di indipendenza condizionale tra le varie feature x_i conoscendo la classe y, che permette di trattare le feature separatamente:

$$P(x_1, x_2, \ldots, x_n|y) = \prod_{i=1}^{n} P(x_i|y) \tag{7}$$

PRO	**CONTRO**
• Semplice ed efficiente da implementare • Veloce da addestrare e applicare su grandi dataset • Funziona bene con dati ad alta dimensionalità • Restituisce stime probabilistiche interpretabili	• Ipotesi di indipendenza spesso irrealistica • Non cattura le interazioni tra caratteristiche • Le performance possono degradare con dati complessi • Meno preciso rispetto a modelli più complessi come Random Forest o SVM

Table 1: Tabella dei pro e contro del modello Naive Bayes.

5.2 ESEMPIO: (classificatore di mail)

Nel caso si voglia classificare un set di mail come spam/non-spam, ognuna di essere può essere descritta come un vettore lungo quanto il numero totale di parole presenti nel dizionario, in corrispondenza dei termini del vocabolario presenti all'interno della mail vi è un 1, 0 altrimenti (rappresentazione **Bag of Words**).
Il modello stima le probabilità $P(word|spam)$ e $P(word|!spam)$ per ogni parola, assumendo indipendenza tra esse.
Così facendo, ad ogni nuova mail il modello calcola la probabilità $P(email|spam)\cdot P(spam)$ e $P(email|!spam)\cdot P(!spam)$ sfruttando l'indipendenza assunta tra le parole che compongono l'email. Di conseguenza:

$$P(email|spam) = \prod_{i=1}^{n} P(word_i|spam)$$

e analogamente per $P(email|!spam)$.

6 K-NN (K-Nearest Neighbors)

Il k-Nearest Neighbors (k-NN) è un classificatore non parametrico basato su un principio molto semplice: un punto viene classificato in base alla classe dei suoi k vicini più prossimi nello spazio delle feature. L'idea centrale è che i punti vicini tra loro tendono ad appartenere alla stessa classe.

6.1 Theorem

Se si lascia il numero di campioni n crescere all'infinito, e il numero di vicini per ogni campione k(n) crescere all'infinito anch'esso ma più lentamente di n:

$$\lim_{n\to\infty} \frac{k(n)}{n} = 0$$

allora per ogni punto il classificatore NN convergerà in probabilità alla reale densità.

6.2 Principio di funzionamento

Non vi è un'effettiva fase di training. Tutti i campioni di training vengono semplicemente salvati insieme alle rispettive etichette.

- Dato un nuovo punto da classificare, viene calcolata la distanza tra questo punto e ciascuno dei campioni nel dataset di training
- Vengono selezionati i k campioni più vicini (i "k-nearest neighbors")
- La classe del nuovo punto viene assegnata in base alla maggioranza delle classi dei vicini. Ad esempio, se $k = 5$ e 3 vicini (email) appartengono alla classe A (spam), mentre 2 appartengono alla classe B (!spam), il punto viene classificato come appartenente alla classe A.

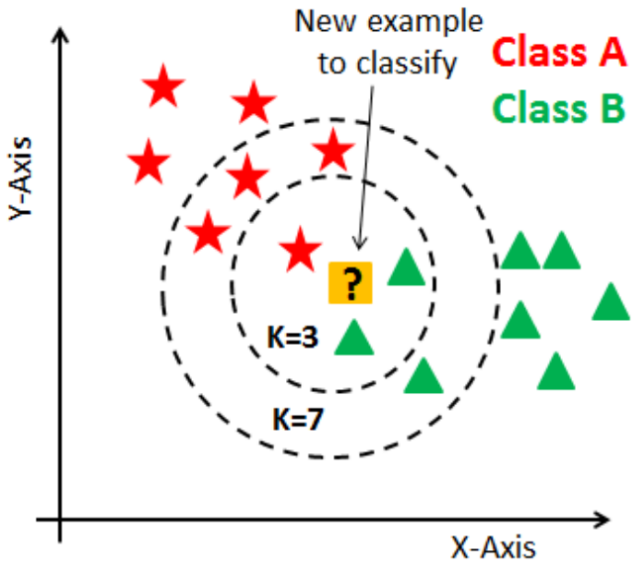


Figure 2: Esempio di un nuovo dato da classificare con k-NN

PRO	**CONTRO**
• Semplice e intuitivo da implementare • Non richiede addestramento esplicito • Funziona bene con dati non lineari e distribuzioni complesse • Adattativo alla distribuzione locale dei dati	• Elevato costo computazionale in fase di classificazione • Necessita di grande memoria per conservare tutti i dati • Sensibile al rumore e agli outlier per valori piccoli di k

Table 2: Tabella dei pro e contro del classificatore k-NN.

Volendolo applicare sempre all'esempio della classificazione di email spam e non-spam, dopo aver rappresentato ogni email tramite il modello **Bag-of-Words** nello spazio delle feature, appena ne arriva una nuova la posiziono in base alle sue feature (alle parole che contiene), e nello spazio n-dimensionale che si è creato vado a vedere a che classe (spam/!spam) appartendono le k mail più vicine.

7 10 Fold Cross Validation

La 10-Fold Cross Validation è una tecnica di validazione incrociata utilizzata per valutare la capacità di generalizzazione di un modello di apprendimento automatico. L'idea principale è quella di suddividere il training set in 10 sottoinsiemi (fold), per poi utilizzare questi sottoinsiemi sia per l'addestramento che per la validazione in maniera alternata.

7.1 Procedimento

Divido inizialmente il mio dataset in training set e test set (in media la divisione è di 80% per training set e il restante 20% per il test set)
I dati estrapolati per il training set saranno poi quelli sui quali verrà effettivamente applicata la cross validation.

- Il set viene diviso in 10 subset di dimensioni simili tra loro (folds)
- Vengono compiute 10 iterazioni totali. Per ogni iterazione 9 dei 10 folds vengono utilizzati per l'addestramento del modello, mentre il restante viene utilizzato per la validazione
- A ogni iterazione si cambia il fold utilizzato per la validazione, in modo che ciascun fold venga utilizzato esattamente una volta come test set.
- Viene calcolata una metrica (ad esempio accuratezza) ad ogni iterazione e alla fine viene restituita la media tra le 10

La cross-validation deve essere applicata ai soli dati di training in quanto se fosse eseguita anche sui dati di test si introdurrebbe un bias, per poi testare il modello finale su dei dati utilizzati per addestrarlo.

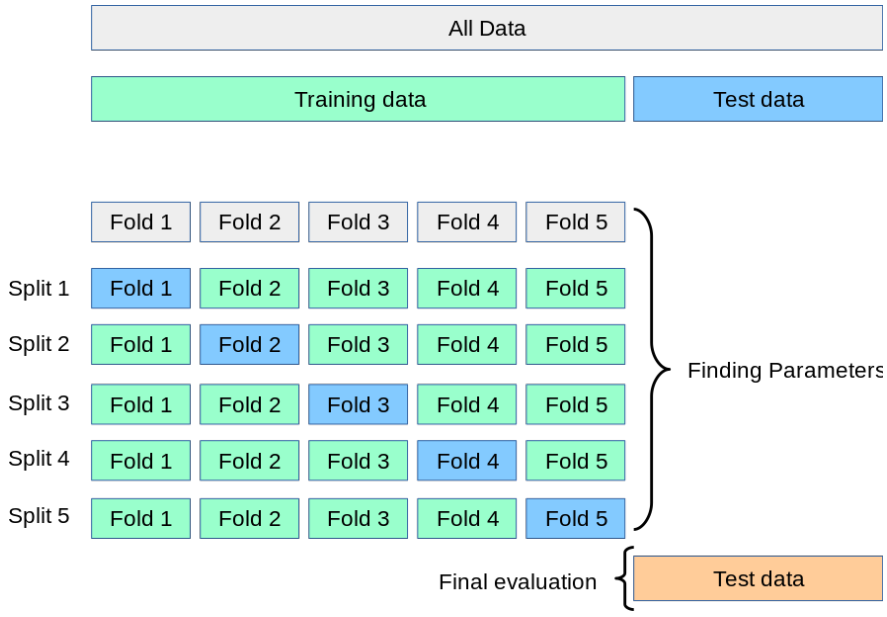


Figure 3: Esempio di 5-fold cross validation

8 Analisi risultati

8.1 Accuratezze della cross-validation

SVM (linear)	SVM (poly)	SVM (RBF)	Random Forest	Naive Bayes	k-NN
0.9329	0.9367	0.9351	0.9448	0.8272	0.9147

Table 3: Accuratezze per ciascun metodo

8.2 Tempi di validazione

SVM (linear)	SVM (poly)	SVM (RBF)	Random Forest	Naive Bayes	k-NN
4.5658	2.0921	2.6796	10.5673	0.0796	0.5222
5.4761	2.1901	2.5177	10.4417	0.1001	0.4407
5.3363	2.2566	2.6266	10.2351	0.0823	0.4640
4.6124	2.1552	2.4805	10.4297	0.0772	0.4637
5.5583	2.2182	2.5249	10.5536	0.0820	0.4348
4.5846	2.1982	2.4667	9.9879	0.0693	0.4090
4.5529	2.0735	2.3249	9.9275	0.0685	0.4443
4.4475	2.0653	2.3341	10.9524	0.1098	0.5724
5.3506	2.1278	2.4128	10.1818	0.0715	0.4198
4.6044	2.1961	2.4006	10.1331	0.0710	0.4395
4.6133	2.1478	2.3667	10.1186	0.0705	0.4252
4.6971	2.1392	2.4100	10.1688	0.0721	0.4147
5.1182	2.1950	2.4424	10.2647	0.0758	0.4307
4.7631	2.1492	2.3950	10.0874	0.0698	0.4248
4.6264	2.2438	2.4954	10.2515	0.0844	0.4268
5.7683	2.4492	2.6787	11.0586	0.0896	0.4980
4.8701	2.1831	2.4386	10.1000	0.0865	0.4750
4.7949	2.1208	2.4276	10.2418	0.0782	0.4131
4.8594	2.1693	2.4183	10.2946	0.0815	0.4649
4.6442	2.1005	2.3777	10.0802	0.0691	0.4127
4.8922	2.1735	2.4609	10.3038	0.0794	0.4498
0.3848	0.0816	0.0997	0.2869	0.0106	0.0406

MEDIA DEVIAZIONE STANDARD

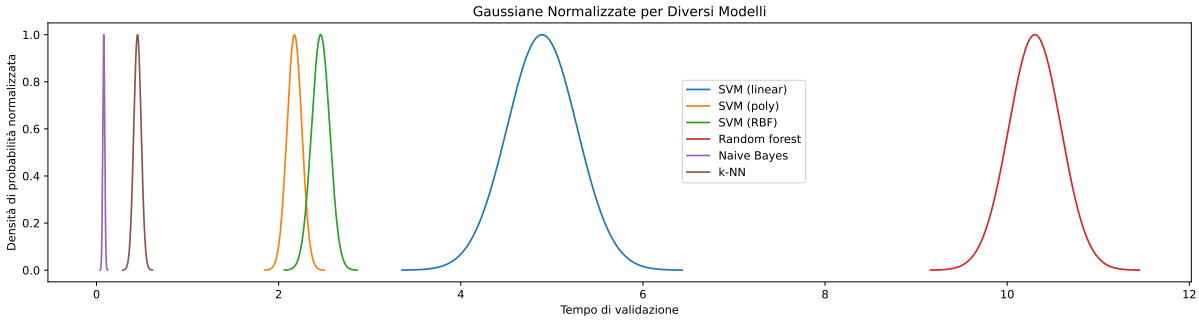


Figure 4: Gaussiane normalizzate per i sei modelli differenti

È evidente come il modello che impiega meno tempo ad eseguire la 10-fold cross-validation (Naive Bayes) è anche lo stesso che riporta accuratezze peggiori. Allo stesso tempo il modello più accurato di tutti (Random Forest) pecca nel tempo di validazione. Un buon compromesso per questo caso specifico può essere il classificatore k-NN, il quale riporta un'accuratezza superiore al 90% ma un tempo di validazione non di poco inferiore rispetto ai 3 modelli SVM e Random Forest.

8.3 Addestramento e testing

Classificatore	**Training time (s)**	**Prediction time (s)**	**Missclassified**	**Test accuracy**
SVM (Linear)	0.5675	0.0368	64	0.931
SVM (Poly)	0.2316	0.0477	54	0.941
SVM (RBF)	0.2673	0.0619	55	0.940
Random Forest	1.1087	0.0141	51	0.945
Naive Bayes	0.0015	0.0226	162	0.824
k-NN	0.0008	0.1170	65	0.929

Table 4: Confronto delle metriche misurate per diversi classificatori.

Anche dai risultati di addestramento e testing le accuratezze risultano molto simili a quelle predette dalla 10-fold cross-validation. In questo caso però i tempi di predizione evidenziano come (seppur mantenendo un tempo molto basso con questo test set nello specifico) il k-NN risulti impiegare più tempo dei concorrenti. Come detto in questo caso risultano comunque essere tempi molto bassi ma avendo magari un dataset più impegnativo questa differenza risulterebbe essere significativa.
Un buon compromesso sono i classificatori SVM (infatti tra i più utilizzati in Machine Learning), soprattutto con kernel polinomiale o radiale, con un'accuratezza di predizione intorno al 95% e dei tempi sia di training che di testing nella media.