Gruppo 01 - Classificazione di Sequenze

Salvatore Cusimano
Francesca Gianferrara
Giulia Tortorici

Introduzione

Per capire a pieno il progetto dobbiamo fare riferimento ad alcuni concetti in ambito biologico, partendo anche da quelle che sono le definizioni più semplici: organismo eucariote e procariote; da cosa sono composti il DNA e l'RNA; definizione di promotore; cosa è la trascrizione di un gene.

Un organismo *eucariote* può essere monocellulare (unica cellula) o pluricellulare (funghi, e organismi appartenenti al mondo animale e vegetale) in cui il materiale genetico è protetto da un nucleo delimitato da una membrana. Un organismo *procariote*, invece, possiede cellule prive di un nucleo ben definito e delimitato dalla membrana nucleare, solitamente sono unicellulari o coloniali (batteri, archea).

Per quanto riguarda il *DNA* è una grande molecola che possiede le informazioni genetiche per lo sviluppo, l'omeostasi e la riproduzione degli esseri viventi. Il DNA non è altro che un acido nucleico a doppio filamento, composto da: uno zucchero, un gruppo fosfato e una base azotata. Possiede la potenzialità di autoduplicazione e permette la replicazione e la trascrizione dell'informazione chimica in essi contenuta. Le basi azotate si classificano come purine e pirimidine. Le purine sono adenina (A) e guanina (G), mentre, le pirimidine sono citosina (c) e timina (T). Esse sono fondamentali perché dal loro appaiamento dipende la struttura a doppia elica del DNA.

L*'RNA* è un acido nucleico a singolo filamento, una delle differenze dal DNA (quella a noi rilevante per questo progetto) è data dalla sostituzione della base azotata "timina" con "l'uracile" (U). Possiede vari ruoli biologici, quali la codifica, regolazione ed espressione dei geni, in particolare la sintesi proteica.

La *trascrizione* è il processo mediante il quale le informazioni contenute nel DNA vengono trascritte enzimaticamente in una molecola complementare di RNA. Concettualmente, si tratta del trasferimento dell'informazione genetica dal DNA all'RNA.

Il **promotore è una regione di DNA** costituita da specifiche sequenze dette consenso, al quale si lega la RNA polimerasi (un enzima) per iniziare la trascrizione di un gene, o più geni; (più geni possono andare a definire l' *operone,* l'unità funzionale del DNA che controlla la sintesi dell'RNA messaggero). Solitamente essi si trovano a monte del gene dal quale iniziano la trascrizione e sono lunghi circa 200 bp; (un esempio è proprio la TATA box di cui parleremo in seguito)

I promotori si possono differenziare in diversi modi, ma a noi interessa l'importante differenza che c'è tra: *Eucarioti* e *Procarioti*; in quanto le caratteristiche che costituiscono tali sequenze sono diverse, nel nostro caso ci interessano maggiormente quelle procariotiche.

Nei **procarioti il promotore** si compone di tre parti:

- un sito di inizio (quasi sempre una base azotata purinica).
- una sequenza che si trova a -10 bp rispetto al sito di inizio della trascrizione indicato con +1, formata da sei paia di basi, con funzioni analoghe alla TATA box eucariotica (cioè formata da coppie di basi azotate T-A, T-A) e sequenza TATAAT.

- una sequenza a -35 (rispetto al sito di inizio della trascrizione) contenente sei paia di basi con sequenza TTGACA.

Esistono elementi regolatori prossimali per determinare quando e quanto frequentemente un gene è espresso. Sono posti tra -50 e -200 rispetto al sito d'inizio della trascrizione:

1. GC box: tipica dei geni housekeeping e opera come intensificatore
2. CAAT box: posta a -75 e opera come intensificatore CRE elemento di risposta a cAMP

Queste informazioni risultano essere essenziali per comprendere i risultati.

Obiettivo del progetto

L'obiettivo principale è sviluppare un modello di classificazione binaria in grado di distinguere sequenze di DNA promotrici da sequenze non promotrici. Questa classificazione binaria è cruciale per comprendere i meccanismi di regolazione genica e sviluppare applicazioni innovative in ambiti come la biologia sintetica, la genomica funzionale e la medicina di precisione.

Il Dataset

Il dataset che è stato fornito in un foglio excel in formato csv che presenta:

- Sequenze promotrici e sequenze non promotrici.
- 106 righe di cui 53 classe 0 e 53 classe 1.
- 2 colonne, cioè label e sequences.
- Si intuisce già da sopra che il dataset è bilanciato.

WORKFLOW

Il flusso di lavoro che abbiamo eseguito vede:

1. Importazione del **Dataset.**
2. **Preprocessing** dei Dati.
3. **Estrazione** delle **Caratteristiche.**
4. **Splittaggio**, Pulizia dei Dati ed imputazione.
5. Eventuale **bilanciamento** del Dataset.
6. Metodi di **Selezione** delle **caratteristiche.**
7. **Training** e Tuning degli iperparametri.
8. **Testing.**
9. Interpretazione dei **risultati ottenuti.**

Abbiamo analizzato il dataset fornito con il **deep learning** applicando una semplice CNN ai fini di estrarre le caratteristiche principali e poi effettuare la classificazione

In ultima analisi mettiamo a confronto la **metodologia utilizzata** a lezione per la classificazione delle sequenze, mettendola a confronto con quella da noi sperimentata

DATA PREPARATION

E' uno step fondamentale, si occupa di raccogliere, trasformare, organizzare, analizzare e interpretare i dati a noi forniti, ottimizzandoli per l'utilizzo del machine learning.

Importazione dataset e installazione dipendenze

Primo step è l'installazione di BioPython, la versione da noi installata è la 1.84. Installiamo anche iFeatureOmegaCLI, il quale consente di estrarre sequenze proteiche in formato FASTA.
Una volta importato il file csv, lo andiamo a stampare e risulta essere in questo modo:

```
   label                                           sequence
0      1  tactagcaatacgcttgcgttcggtggttaagtatgtataatgcgc...
1      1  tgctatcctgacagttgtcacgctgattggtgtcgttacaatctaa...
2      1  gtactagagaactagtgcattagcttatttttttgttatcatgcta...
3      1  aattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaata...
4      1  tcgataattaactattgacgaaaagctgaaaaccactagaatgcgc...
```

IFeatureOmegaCLI richiede il formato FASTA, quindi è necessario convertire il dataset in questo formato per poter effettuare l'estrazione delle caratteristiche.

Estrazione caratteristiche e Final dataset

Utilizziamo iFeatureOmega e rd-kit per l'estrazione di tutte le caratteristiche messe a disposizione, indipendentemente dall'utilità.

iFeatureOmega offre la possibilità di estrarre fino a 49 tipologie di caratteristiche differenti. Alla fine di questa fase viene creata una variabile *final_dataset*, dove abbiamo concatenato i dati, le caratteristiche che abbiamo deciso di utilizzare dopo una serie di esperimenti volti a dimostrare quali fossero quelle capaci di generare le percentuali di accuratezza più alta.

Inizialmente abbiamo provato con la concatenazione di tutte e 49 le caratteristiche, ma venivano più di 22.000 di colonne finali; Poi di queste 49 ne abbiamo selezionate 8 in base all'utilità *(interazioni elettriche, frequenza dei nucleotidi, posizione)*, per poi concatenarle e vedere che l'accuratezza risultava migliore solo con 2.

Vedendo meglio i vari steps eseguiti abbiamo:

- Definito la variabile dna = iFeatureOmegaCLI.iDNA('').
- Attraverso la funzione dna.display_feature_types() vediamo quali tipologie di caratteristiche possiamo estrarre con iFeatureOmega.
- Utilizziamo get_descriptor e get_encodings per estrarre i 49 tipi di caratteristiche in relazione al nostro insieme di dati.

- Dopo avere deciso quali tra i 49 tipi di caratteristiche ci interessano utilizziamo pd.concat per concatenarli alla variabile final_dataset.

Successivamente attraverso metodi di selezione delle caratteristiche punteremo a selezionare quelle più utili ai fini del nostro task.

Significato nella letteratura di iFeatureOmega per le caratteristiche estratte

Vediamo le 3 caratteristiche principali con cui abbiamo lavorato, cioè CKSNAP di tipo 1 , Kmer di tipo 1 e PseEIIP.

CKSNAP di tipo 1 (Composizione di coppie di acidi nucleici distanziate di k posizioni, Tipo 1): è un metodo utilizzato per l'encoding delle sequenze di acidi nucleici, basato sulla frequenza di coppie di nucleotidi separati da k nucleotidi. Questo approccio permette di trasformare le sequenze di acidi nucleici in vettori numerici, rendendoli adatti per applicazioni di machine learning, come la classificazione di sequenze, l'allineamento e la predizione. Se k = 0, allora non sarà presente alcun gap e le coppie di nucleotidi sono consecutive; in questo caso il vettore si calcola come ad esempio prendiamo AA $AA=m/P-1$ dove m è quante volte AA appare nella sequenza, mentre P sarebbe la lunghezza della sequenza.

Se k = 1, allora le coppie di nucleotidi saranno distanziate da 1 gap (e così via).

Kmer di tipo 1 : è una sequenza di **k nucleotidi consecutivi**, dove k è un valore predefinito ed è pari 3. i kmers sono tutte le possibili sequenze di 3 nucleotidi che si possono formare utilizzando i 4 nucleotidi A, C, G e T (o A, C, G, U per RNA). la sua frequenza viene calcolata: $f(t)=N/N(t)$ dove:

- $N(t)$ è il numero di volte in cui il kmer t appare nella sequenza.
- N è la lunghezza totale della sequenza.

PseEIIP : estraendo questa caratteristica possiede 64 colonne totali, esprime le proprietà elettroniche e di carica delle basi nucleotidiche (A, T, G, C) di una tripletta all'interno di sequenze di DNA o RNA ; di fatto mappa ogni base nucleotidica in un "valore elettronico" che può essere utilizzato per estrarre informazioni rilevanti dalla sequenza per successive analisi predittive. Viene definito un vettore V di dimensione 64 che fa il prodotto di EIIP per la f del trinucleotide dove f è la frequenza normalizzata dell'i-th trinucleotide.

Data Preprocessing

Fase fondamentale nel processo di addestramento dei modelli di machine learning e ha l'obiettivo di pulire, trasformare ed organizzare i dati:

1. ***cleaning*** : pulire i dati, questo perché all'interno ci potrebbero essere dei campioni, numeri, dati tabulari che possiedono dei valori NaN (non disponibili) e che vanno sostituiti.
2. ***scaling*** : standardizzazione delle unità di misura al fine di garantire l'uniformità dei dati, permettendone facilmente il confronto e l'analisi.

3. ***balancing***: andare a bilanciare i dati nelle classi (qualora dovesse essere sbilanciato), ovvero se una classe ha un maggiore o minore rappresentazione rispetto a quella con la quale deve essere confrontata.

Splittaggio

Dopo avere estratto le caratteristiche , abbiamo costruito la X e la Y; Eseguito lo splittaggio conferendo il 20% al test e l'80% al training utilizzando la funzione *train_test_split*, successivamente con i **metodi di cross-validation** abbiamo dedicato un 10% alla validation;

Pulizia dei Dati

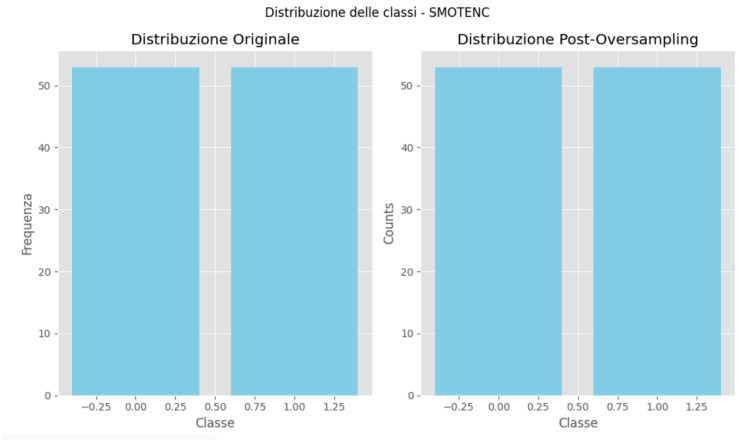
Rappresenta una delle fasi cruciali nel preprocessing, poichè la presenza di valori mancanti può compromettere la performance della macchina, con il metodo **X.isna.sum()** abbiamo visto se ci fossero valori mancanti, ma dato che abbiamo estratto le caratteristiche da 0 non potevano esserci. Applicare il seguente metodo risulta essere lo stesso metodo isna a righe o colonne è lo stesso (in questo caso noi scegliamo di vedere se ci sono righe con NaN, ma ovviamente potevamo applicarlo anche alle colonne ottenendo i medesimi risultati).

Bilanciamento

Il dataset fornito era già bilanciato, infatti ci sono 53 sequenze di classe 0 e 53 sequenze di classe 1. Il bilanciamento può essere visto anche graficamente:

Abbiamo utilizzato vari e nuovi metodi grafici, al fine di far intendere che ne esistono diversi, sono state utilizzate: *matplotlib, seaborn, plotply,bokeh.*

Solitamente, i dataset forniti sono quasi sempre sbilanciati, in tal caso si possono applicare serie di metodi di bilanciamento come SMOTE, SMOTENC, ADASYN, BorderlineSMOTE, KMeansSMOTE, SVMSMOTE. (Ovviamente non abbiamo fatto il bilanciamento sulla stessa variabile, ma per ogni metodo abbiamo definito la sua variabile come *"X_train_smotenc", "X_train_ada", "X_train_svm_smote").* Graficamente, a scopo didattico, utilizzando il metodo SMOTENC notiamo che non c'è stato nessun cambiamento:



Alla fine abbiamo mantenuto come variabile finale da richiamare durante la normalizzazione la variabile *X_train.*

Normalizzazione

Per quanto riguarda la normalizzazione abbiamo applicato StandardScaler sulla variabile X_train ed abbiamo definito la nuova variabile standardizzata "*X_train_scaled"*. Si occupa di andare a scalare le variabili nello stesso intervallo per renderle comparabili.
Abbiamo cercato nuovi metodi di Standardizzazione: RobustScaler, il quale scala le features utilizzando metodi statistici che sono robusti rispetto agli outliers. Abbiamo ritenuto più opportuno utilizzare il primo metodo con la variabile *X_train_scaled*.

Selezione delle Caratteristiche

Altra fase di estrema importanza, ci permette di andare ad addestrare il modello in maniera più efficace, andando a selezionare quelle più rilevanti ed informative, al fine di migliorare le sue prestazioni. Agisce principalmente andando a rimuovere le caratteristiche ridondanti o non rilevanti.
Abbiamo lavorato su 3 principali metodi:

1. *Filter methods:* agisce direttamente sulle caratteristiche, si applicano dei test statistici per vedere quanto quella caratteristica è correlata con l'output del modello; questi approcci sono veloci, perché vanno a confrontare separatamente ogni singola caratteristica con l'output del modello (prima scrematura delle variabili).
2. *Wrapper methods:* agisce sul dataset, vanno applicati dopo il metodo dei filtri, i quali provano tutte le combinazioni delle caratteristiche poiché una caratteristica presa singolarmente potrebbe non avere significato.
3. *Metodi embedded:* solitamente pongono particolare attenzione all'ottimizzazione del codice in termini di velocità di esecuzione, uso della memoria e risparmio energetico.

Filter methods
Nel progetto abbiamo utilizzato:

- **Metodo Near-Zero-Variance**: è una tecnica di selezione delle feature che mira a rimuovere le variabili che non forniscono informazioni utili al modello di machine learning. Le feature con varianza molto bassa, o quasi nulla, sono in genere caratteristiche che non cambiano molto tra le osservazioni o che sono costantemente uguali o quasi uguali in tutto il dataset.

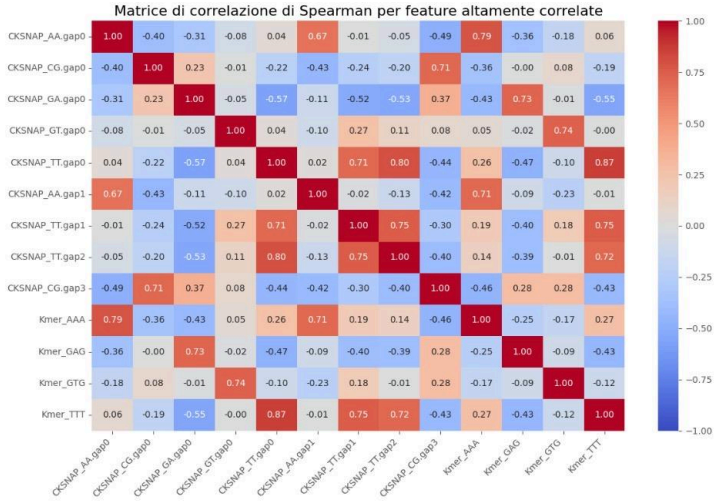
  Con questo metodo modificando il parametro **threshold** (valore soglia) varieranno il numero di caratteristiche scartate: più è alto questo valore più caratteristiche scartiamo, più è basso questo valore meno caratteristiche scartiamo. Le features dopo questo controllo rimangono 64, perciò andiamo ad applicare un secondo filtro.

  (Abbiamo provato a modificare il valore soglia con valori pari a 0.01 o 0.1, il numero di caratteristiche non varia, invece con un valore pari a 1 il numero si riduce, ma in questo modo rischiamo di togliere anche caratteristiche che potrebbero essere utili ai fini del nostro task)

- **Metodo di Spearman**: serve per rimuovere le features altamente correlate tra loro. Si confrontano righe e colonne della matrice e si scartano le celle con un valore sopra una **threshold** già stabilito (0.7). Si calcola la matrice di correlazione ed infine si genera un grafico per osservarne la correlazione.

Questo metodo scarta le caratteristiche correlate tra di loro, se non ci sono caratteristiche correlate non scarta niente, come ad esempio nel caso in cui si utilizza una sola tipologia di caratteristica. Infatti la matrice risulterà **vuota,** nel caso in cui utilizziamo PseEIIP oppure CKSNAP oppure KMER da soli ovviamente il numero di caratteristiche dopo l'applicazione del metodo di Spearman rimane invariato perchè le varie colonne sono tutte differenti tra di loro.

Nel caso in cui utilizziamo caratteristiche concatenate tra loro come: CKSNAP e KMER vediamo come *CKSNAP_TTgap0* ,risulti essere correlata alla variabile *Kmer_TTT*. Quindi verrà scartata e la matrice risulterà in questo modo:

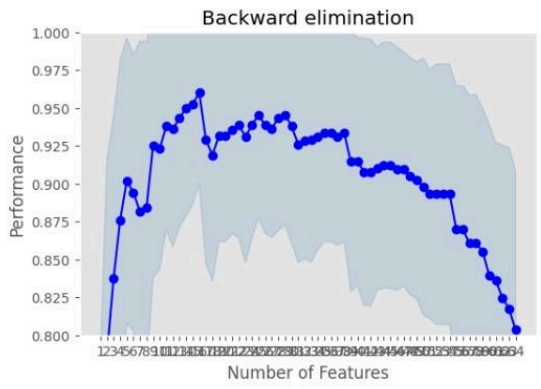


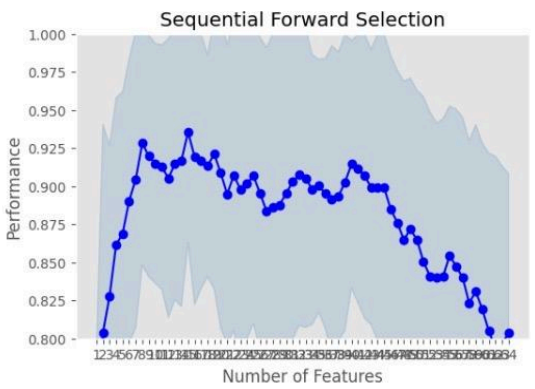
Wrapper methods
Nel progetto abbiamo utilizzato:

- **Metodo Forward Selection** per selezionare le migliori caratteristiche, ovvero quelle più significative nel predire la variabile dipendente, in modo da costruire un modello più semplice, evitando l'inclusione di variabili irrilevanti o ridondanti. Tuttavia, presenta alcune limitazioni, tra cui il rischio di overfitting e la possibilità di non trovare la combinazione ottimale di variabili, infatti può essere utilizzato in combinazione con altre tecniche.

- **Backward elimination** a differenza del metodo forward selection (il quale inizia con un modello vuoto e aggiunge progressivamente le variabili), parte da un modello completo (con tutte le variabili predittive) e rimuove progressivamente le variabili meno significative. Permette di semplificare il modello, ma anche in tale caso ci può essere rischio di overfitting.

Graficamente vediamo come variano le performance in base al numero di caratteristiche: all'aumento delle caratteristiche, diminuiscono le performance, contrariamente, quando diminuiscono le caratteristiche aumentano le performance.



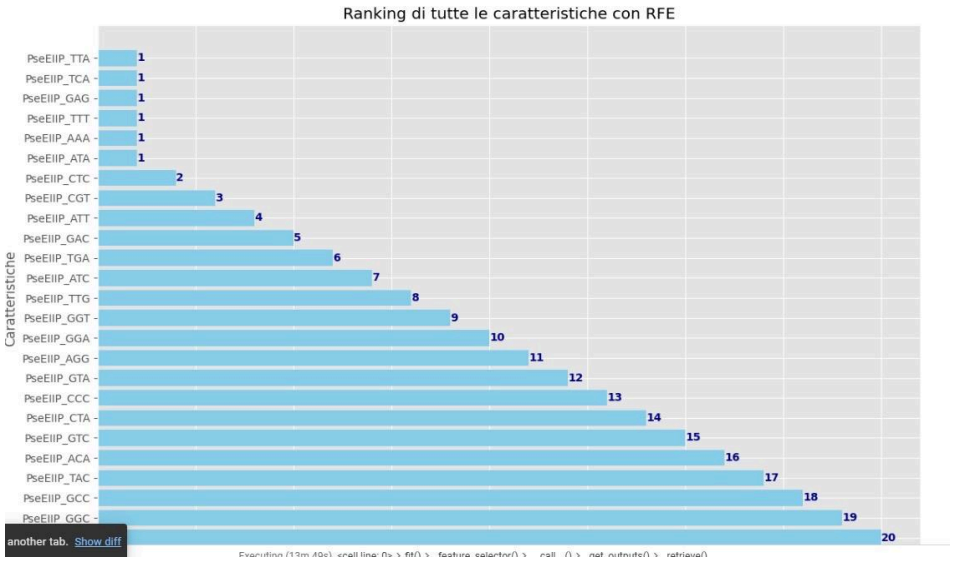


Abbiamo applicato alla variabile *X_train* prima il controllo della varianza e poi la matrice di spearman definendo una nuova variabile che si chiamerà *df_correlation*. *df_correlation* viene prima utilizzata dal metodo backward e parallelamente dal metodo forward per ottenere le migliori caratteristiche. Per la selezione delle caratteristiche tramite questi metodi abbiamo usato *kneighborsclassifier* invece di xgboost. Infine abbiamo unito i risultati ottenuti dai due metodi tramite una lista ed abbiamo eliminato i doppioni, al fine di ottenere le migliori caratteristiche.

Tramite apposite funzioni come *sfs.k_features_names* stampiamo i nomi delle caratteristiche migliori ottenute secondo questi metodi.

- **Recursive elimination**: RFE inizia con un modello che include tutte le variabili. Successivamente, elimina iterativamente la caratteristica meno importante (utilizzando un ranking) e rifitta il modello sul sottoinsieme rimanente. Questo processo continua fino a quando non viene selezionato il numero desiderato di caratteristiche. è stato applicato alla '`RandomForestClassifier`'. In questo caso prende le 6 migliori caratteristiche.

Tramite metodi grafici vediamo quali sono le caratteristiche più rilevanti:



Questo ranking di tutte le caratteristiche, quando il ranking è 1, segna le caratteristiche più importanti, quelle con un valore inferiore sono quelle che contribuiscono poco con le performance del modello. Le migliori caratteristiche risultano essere: *PseEIIP_TTA, PseEIIP_TTT, PseEIIP_AAA , PseEIIP_ATA, PseEIIP_GAG, PseEIIP_TCA.*

Metodi embedded

Metodo LASSO : è una tecnica di regolarizzazione utilizzata in regressione lineare e in modelli statistici per evitare l'overfitting. Introduce una **penalizzazione L1** sulla somma dei valori assoluti dei coefficienti: oltre a ridurre l'errore del modello, forza alcuni coefficienti a diventare **zero**. Possiamo dire che il metodo Lasso effettui **selezione automatica delle variabili**.
Abbiamo messo a paragone i 2 metodi embedded e wrapped, vedendo che le caratteristiche finali stampate erano simili; Quindi il metodo Lasso è stato applicato alla variabile df_correlation ottenuta dopo aver scartato le caratteristiche secondo il metodo Spearman e non tiene conto dei risultati dei metodi wrapped

Ai fini del nostro progetto abbiamo ritenuto più opportuno utilizzare i metodi wrapped per la classificazione, abbiamo quindi al termine della selezione delle caratteristiche aggiornato la nostra variabile *X_train* iniziale con le nuove colonne selezionate.

Cosa succede modificando il numero di features finali da selezionare?

(Nota: da qua in poi per lasciare la relazione più "leggibile" abbiamo caricato i vari screenshots importanti in github, bisogna cliccare nella parola in blu per vedere eventuali screen)

Abbiamo provato a modificare il numero di "colonne" finali per individuare quelle capaci di generare la migliore accuratezza. In particolare, utilizzando il metodo **Recursive Feature Elimination**, abbiamo selezionato inizialmente le 5 colonne migliori, poi 6, 7 e infine 8. Per ciascuna di queste combinazioni, abbiamo analizzato come variava l'accuratezza finale del modello.
L'obiettivo era osservare l'impatto del numero di caratteristiche selezionate sull'accuratezza del modello. Secondo la teoria, è consigliabile utilizzare 1 caratteristica ogni 10-15 righe di dati. Pertanto, considerando che disponiamo di 106 sequenze iniziali, ci aspettiamo un range ottimale di caratteristiche compreso tra 5 e 8. Osserviamo cosa succede:

Test con 5 caratteristiche - Test con 6 caratteristiche - Test con 7 caratteristiche

Dalle immagini di sopra*(click nel blu)* analizzando accuratezza e le varie metriche vediamo come alcuni modelli performano meglio con 5 caratteristiche altri con 6 ed altri ancora con 7;

Otteniamo la minore differenza di accuratezza tra training e testing con un numero di caratteristiche pari a 6. I modelli addestrati con 6 caratteristiche hanno una migliore capacità di generalizzazione

Come cambia l'accuratezza del modello cambiando il tipo di caratteristica?

Vediamo adesso vari esperimenti ed in particolare come cambia l'accuratezza del modello se cambiamo il tipo di caratteristica concatenata alla nostra variabile final_dataset

- Utilizzando solo KMER - Utilizzando solo CKSNAP - Con CKSNAP e KMER insieme
- Utilizzando solo PseEIIP - PseEIIP e CKSNAP insieme - PseEIIP e KMER insieme

Come si vede dai vari screenshots di sopra i migliori risultati si ottengono utilizza PseEIIP da sola

Cosa succede se non selezioniamo le caratteristiche?

Vediamo cosa succede se non si selezionano le colonne migliori:

- Test senza caratteristiche selezionate nei dati di Training - Stesso test nei dati di Testing
- Con caratteristiche selezionate nei dati di Training - Stesso test nei dati di Testing

Dagli screen si osserva che, senza la selezione delle caratteristiche, il modello mostra una scarsa capacità di generalizzazione su nuovi dati. Tuttavia, applicando la selezione delle colonne migliori con la stessa macro-caratteristica, il modello acquisisce una capacità di generalizzazione significativamente migliore.

Tuning degli iper-parametri

A seguito del lavoro svolto sui dati e la loro relativa estrazione-selezione delle caratteristiche possiamo passare alla fase di tuning degli iperparametri; abbiamo inoltre utilizzato la Cross-Validation.

Modificando i parametri dei vari modelli ovviamente si ottimizza o meno l'accuratezza finale del modello.

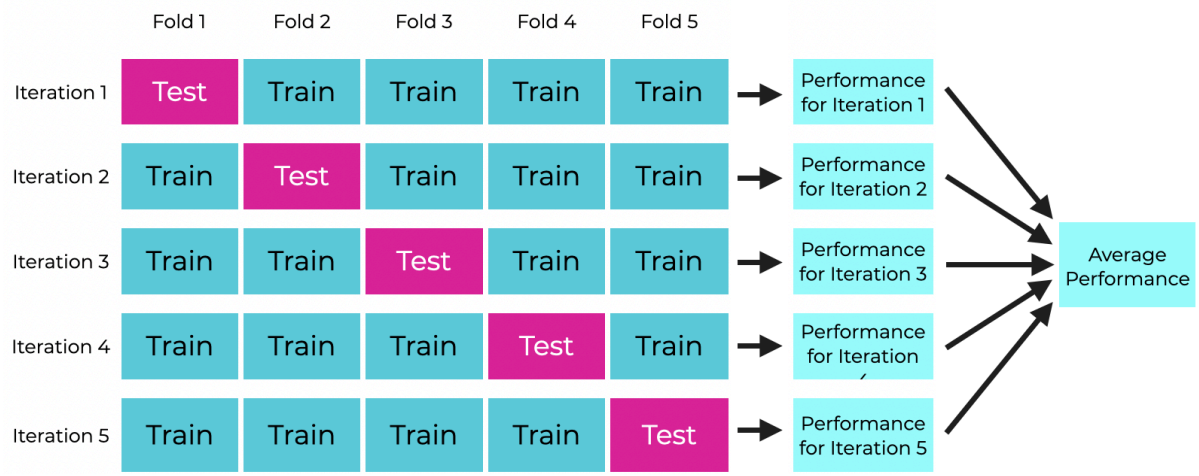
Nel progetto non andiamo a modificare i singoli parametri manualmente vedendo quale porta ad avere un'accuratezza migliore, ma saranno metodi quali gridsearch o randomizedsearch a dirci quali sono le combinazioni di parametri migliori che portano ad avere i livelli di accuratezza più alti.

Cross validation

La cross validation è una strategia per allenare un modello di machine learning, spesso utilizzata quando possediamo un dataset ridotto, evita, infatti, l'ulteriore suddivisione del dataset in dati di training e dati di test. Nella fase di test possiamo provare l'abilità di generalizzazione del modello che abbiamo addestrato, fornendogli dei dati nuovi e vedendo il suo comportamento.
Nel progetto vengono riportati diversi metodi per la cross validation, come: *ShuffleSplit, StratifiedShuffleSplit, PredefinedSplit*[1] *, GroupKFold, StratifiedGroupKFold*, ma ai fini dell'elaborato abbiamo privilegiato lo Stratified Group K-Fold, simile al group k-fold ma in versione stratificata, la quale garantisce una distribuzione dei dati più bilanciata ed efficace.

Secondo questo metodo il dataset viene diviso in dati di training e dati di test. I dati di training a loro volta saranno divisi in k fold o sottoinsiemi, il modello di machine learning a questo punto verrà addestrato k volte, ognuna con uno dei fold come validation test e su un numero k-1 di fold come training set.

[1] I primi tre metodi li abbiamo reperiti in questa sezione:
https://www.google.com/url?q=https%3A%2F%2Fscikit-learn.org%2F1.5%2Fmodules%2Fcross_validation.html



Ad ogni iterazione della cross-validation avremo valori differenti, ma simili, facendone una media è possibile analizzare le metriche ottenute del nostro modello. Questa strategia ci offre, quindi, un'**ottimizzazione degli iperparametri**, effettuando diverse combinazioni dei loro valori per verificare quali di esse restituisce le prestazioni migliori al modello stesso.

Tuning degli iperparametri

Il tuning degli iperparametri è il processo di ottimizzazione degli iperparametri, ci permette di sceglierne i valori corretti al fine di migliorare le prestazioni predittive del nostro modello, questo può essere svolto con due metodi differenti, il Grid Search e il Random Search.

- Il **Grid Search** prova scrupolosamente le combinazioni degli iperparametri, quindi è molto preciso ma anche più lento.
- Il **Random Search** viene utilizzato per dataset di dimensioni maggiori, prende, infatti, sottoinsiemi randomici di iperparametri riportando solo il più efficiente.

Questi due metodi vanno applicati ai diversi algoritmi, nel nostro caso abbiamo svolto entrambi i metodi parallelamente per l'algoritmo di Support Vector Machine (SVM), e solo il Random Search per gli altri modelli, per ridurre i tempi di ricerca.

Support Vector Machine (SVM).

L'SVM è un algoritmo che permette di addestrare modelli di machine learning per problemi di regressione o classificazione binaria. L'obiettivo principale del Support Vector Machine è quello di trovare l'**iperpiano di separazione** al miglior grado possibile dei punti di dati di una classe da quelli di un'altra classe. L'iperpiano migliore sarà quello con un margine più ampio (definito Decision Boundary), il margine è dato dalla distanza dei due **vettori di supporto** tra le due classi. Quindi massimizzando il margine, garantisco una buona generalizzazione del modello.

Le SVM permettono un'efficiente classificazione binaria anche di dati più complessi ad n dimensioni, grazie al **Kernel** caratterizzato da diverse funzioni (lineari, sigmoidali, polinomiali) i dati vengono trasferiti in un'altra dimensione dove è più semplice effettuare la classificazione.

Per l'addestramento del modello abbiamo bisogno di definire gli **iperparametri** che rendono SVM il più accurato possibile.

- **Kernel**: da utilizzare nel modello SVC, possiamo avere un kernel lineare, "linear", "poly" se polinomiale e "rbf" se gaussiano.
- **C** rappresenta il parametro di regolarizzazione, permette di modellare quanti errori ammettere nella ricerca di questo piano, a seconda se ammettiamo troppi pochi pochi errori possiamo andare incontro al fenomeno di overfitting.
- **Degree**: rappresenta il grado della funzione polinomiale quando il kernel è "poly", maggiore è questo valore, minore sarà la capacità di generalizzazione del modello.

Modello addestrato con iperparametri: accuratezza = 88,19%; ' C = 10'; 'coeff = 0'; 'degree = 3'; 'gamma = scale'; 'kernel:sigmoid'.

Albero decisionale

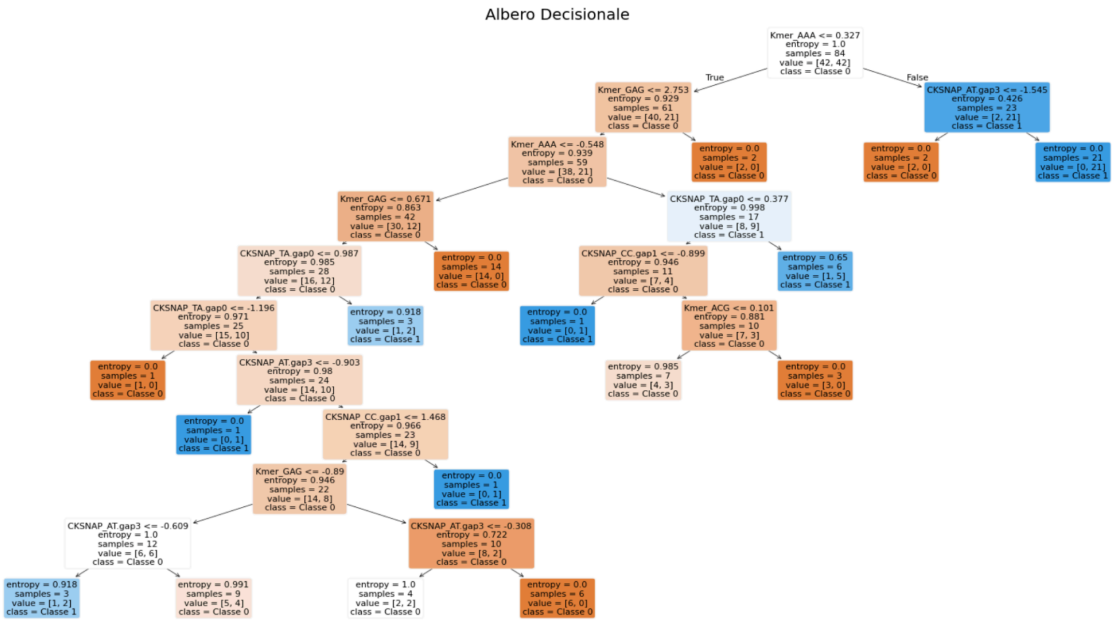
È un algoritmo di apprendimento supervisionato non parametrico che viene utilizzato per problemi di classificazione e problemi di regressione. L'albero decisionale presenta un'architettura gerarchica: le caratteristiche più rilevanti saranno disposte nei **nodi radice**, da questi si generano rami in uscita verso i **nodi interni o decisionali** che a seguito di valutazioni opportune vanno a costituire dei sottoinsiemi omogenei indicati come **nodi foglia**, i componenti di questo gruppo avranno tutti la stessa etichetta. L'albero decisionale è un algoritmo semplice e trasparente, si possono valutare i parametri più importanti, ma con un grado di accuratezza minore rispetto ad altri algoritmi.

Per il tuning degli iperparametri del decision tree nel nostro progetto abbiamo privilegiato il Random Search perché più veloce; parametri:
criterion come funzione di valutazione, se "gini" o "entropy".

- **Splitter** per la strategia di divisione dei nodi, "best" o "random".
- **Max_depth** per la profondità massima dei nodi.
- **Min_samples_split** minimo numero di campioni per effettuare una divisione.
- **Min_samples_leaf** minimo numero di campioni per foglia.
- **Max_features** numero Massimo di caratteristiche per divisione.

Modello addestrato con parametri: accuratezza= 83,19%; 'splitter': 'best'; 'random_state': 42; 'min_samples_split': 10; 'min_samples_leaf': 1; 'max_features': 'log2'; 'max_depth': 10; 'criterion': 'entropy'

Visualizzazione albero decisionale: metodo trasparente.



Random forest.

Il random forest è un algoritmo utilizzato per problemi di regressione e problemi di classificazione, può raggiungere elevati livelli di accuratezza e ottiene ottime performance anche con dataset molto ampi. Il random forest combina modelli di machine learning semplice, quali gli alberi decisionali, disponendoli in parallelo, sarà quindi basato sulla **tecnica di bagging**: addestramento di diversi alberi decisionali in parallelo. Il risultato sarà un insieme di performance dei diversi decision tree che porteranno ad una performance finale e complessiva tramite una media dei valori ottenuti singolarmente.

Utilizziamo nuovamente il Random Search e definiamo la griglia degli iperparametri:

- **n_estimators** per il numero di alberi della foresta.
- **max_depth** rappresenta la profondità massima dell'albero.
- **min_samples_split** sono i minimi numeri richiesti per dividere un nodo.
- **min_samples_leaf** sono il minimo numero di campioni per foglia.
- **max_features** rappresentano il numero di caratteristiche da considerare per il miglior split.
- **Bootstrap** se il campionamento viene effettuato con o senza sostituzione.
- **Criterion** che valuta la qualità di uno split con scelte di information gain o indice di Gini.

Modello addestrato con iperparametri: accuratezza: 86,66%; 'n_estimators': 200; 'min_samples_split': 2; 'min_samples_leaf': 4; 'max_features': 'sqrt'; 'max_depth': 20, 'bootstrap': True.

XGBoost.

È un algoritmo che combina modelli di machine learning semplici, quali gli alberi decisionali, disponendoli in serie, a differenza del random forest dove avevamo una disposizione in parallelo. Applica, quindi, la **tecnica di boosting** che segue una costruzioni di modelli sequenziali: si parte da un primo modello più semplice, del quale si calcolano gli errori residui per restituire una base di partenza al modello successivo; in questo modo riusciamo a massimizzare la performance del modello.

Come iperparametri ritroviamo:

- **learning_rate** che controlla la velocità con cui un modello apprende i dati, ma valori troppo alti di questo iperparametro possono causare instabilità del modello.
- **n_estimators**, ovvero il numero di fasi di boosting da eseguire, ma se troppo elevato si rischia l'overfitting. Devo, così, trovare un buon equilibrio tra questi due iperparametri.
- **objective** che specifica la funzione da ottimizzare, che può essere "binary:hing" o binary:logistic", ovvero quello utilizzata nel progetto, la quale ottimizza la perdita logistica.
- **max_depth** rappresenta il numero di nodi del decision tree.

Metodi nuovi di training sperimentati

Per la fase di sperimentazione abbiamo ricercato e utilizzato altri algoritmi quali[2]:

- **Dummy Classifier**
- **Ada Boost Classifier**
- **Voting Classifier**
- **Gradient Boosting Classifier**
- **MultiLayer-Perceptrons (MLP)**
- **Naive Bayes**

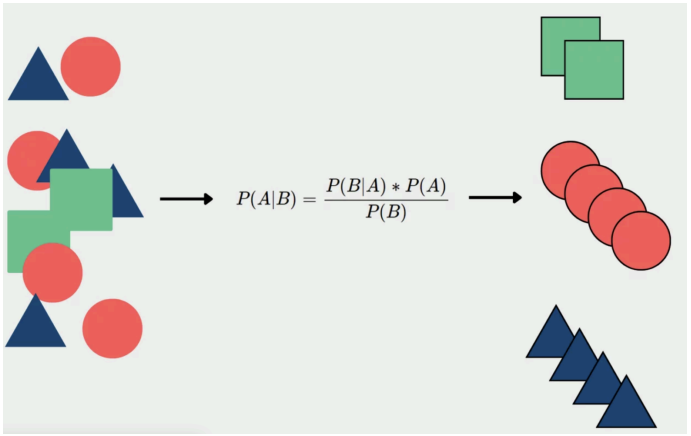
Decidiamo però di entrare più nel dettaglio con MultiLayer-Perceptrons, Naive-Bayes e Voting Classifier.

Naive Bayes (NB)

Il Naive Bayes è un modello di machine learning usato per problemi di classificazione binaria o multiclasse sfruttando il **Teorema di Bayes:** *Siano A e B due eventi con probabilità non nulle, la probabilità condizionata di A rispetto a B è uguale al prodotto tra la probabilità condizionata di B rispetto ad A e la probabilità di A tutto fratto la probabilità di B.*

---

[2] Link utilizzati per la documentazione degli altri algoritmi utilizzati: https://scikit-learn.org/1.5/api/sklearn.ensemble.html



Viene chiamato "Naive" perché presuppone le **variabili completamente indipendenti** l'una dall'altra ma che contribuiscono tutte alla probabilità della classe; è un modello molto semplice e veloce che permette di ottenere ottime performance quando l'ipotesi di indipendenza delle caratteristiche è confermata, ma rischia di portare in errore quando le variabili non sono indipendenti.

MultiLayer-Perceptrons (MLP)

Il MLP è un algoritmo che rappresenta una rete neurale artificiale con neuroni artificiali i quali emulano il meccanismo di funzionamento dei neuroni biologici, superando una determinata soglia permettendo di attivare il neurone successivo. La rete neurale avrà un'organizzazione strutturale a strati e tutti i neuroni saranno interconnessi tra di loro, avremo:

- **Strato input**, composto da nodi che contengono le features del dataset.
- **Strato intermedio o nascosto**, con nodi che ricevono il loro input dall'output degli strati precedenti e mandano a loro volta input agli strati successivi.
- **Strato di output**, che produce output desiderato dalla task del modello.

Il MLP è un modello **black box** perché non sappiamo cosa avviene al suo interno a livello dei nodi nascosti o intermedi, riesce a prendere in input le sequenze e le colonne del nostro dataset.

Utilizziamo ancora Random Search per la ricerca dei valori di iperparametri di MLP.

- **Hidden_layer_sizes** definisce l'architettura dei layer (strati) nascosti.
- **Activation** rappresenta la funzione di attivazione che può essere "logistic" (sigmoide) ovvero una funzione non lineare utilizzata per problemi di classificazione binaria.
- **Solver** un algoritmo di ottimizzazione utilizzato durante l'addestramento del modello, il quale può essere "adam" per dataset grandi, oppure "lbfgs" per dataset piccoli.
- **Learning_rate** ovvero il tipo di tasso di apprendimento in base alla diminuzione della perdita in addestramento. Il valore rimane costante al diminuire della perdita, al contrario sarà diviso per 5.
- **Learning_rate_int** sarebbe il tasso di apprendimento iniziale.

- **Batch_size** per stabilire la dimensione dei batch.

Modello addestrato con iperparametri: accuratezza = 91,52%; 'solver': 'sgd'; 'learning_rate_init': 0.01; 'learning_rate': 'adaptive'; 'hidden_layer_sizes': (50,); 'batch_size': 32; 'alpha': 0.001; 'activation': 'tanh'.

Voting Classifier

Un voting classifier è un algoritmo di machine learning che utilizza diversi classificatori insieme i quali sono capaci di determinare una classe come output. La classe scelta viene "votata" maggiormente dai diversi classificatori in modo tale da ottimizzare le previsioni che vengono effettuate dal modello stesso.

Distinguiamo due tipi di classificazione:

- **Hard Voting** seleziona la classe con più alta probabilità di essere prevista da ogni classificatore.
- **Soft Voting** tra tutte le classi viene selezionata con un effettuando la media di uscita di ogni classe, quella con una media risultante maggiore.

Utilizziamo Random Search per la ricerca dei valori di iperparametri di Voting Classifier.

- **Svm_C** definisce la regolarizzazione per SVM.
- **Svm_kernel** definisce il nucleo per la SVM.
- **Rf_n_estimators** rappresenta il numero di decision tree del random forest.
- **Rf_max_depth** definisce la profondità massima degli alberi decisionali.
- **Rf_min_samples_split** indica il numero minimo di campioni per dividere un nodo.
- **Rf_min_samples_leaf** definisce il numero minimo di campioni per un nodo foglia.
- **Log_reg** per la regressione logistica.

Nello specifico del progetto abbiamo utilizzato il metodo **soft voting**, nel quale vengono calcolate delle classi dai nostri classificatori, successivamente viene scelta la classe con la risultante della media maggiore. I classificatori scelti sono random forest, regressione logistica e Support Vector Machine.

Modello addestrato con iperparametri: accuratezza= 89,30%; 'svm__kernel': 'rbf'; 'svm__C': 1, 'rf__n_estimators': 200; 'rf__min_samples_split': 2; 'rf__min_samples_leaf': 5; 'rf__max_depth': Non; 'log_reg__solver': 'liblinear'; 'log_reg__C': 1.

Fase di Test

Nella fase di test vengono utilizzati i modelli precedentemente addestrati durante la fase di training, inizializzando la variabile **X_test_reduced** con le caratteristiche più rilevanti e successivamente vengono analizzate le metriche.

Metriche:

- **L'accuratezza**: misura la percentuale di previsioni corrette sul totale dei casi.

$$Accuratezza = \frac{Numero\ di\ previsioni\ corrette}{Numero\ totale\ dei\ casi}\ x\ 100\%$$

- La **sensitività**: misura la capacità del modello di classificare i casi correttamente positivi.

$$Sensitività = \frac{Veri\ Positivi}{Veri\ positivi + falsi\ negativi}$$

- La **specificità**: misura la capacità del modello ad identificare correttamente i casi negativi.

$$Specificità = \frac{Veri\ Negativi}{Veri\ Negativi + Falsi\ Positivi}$$

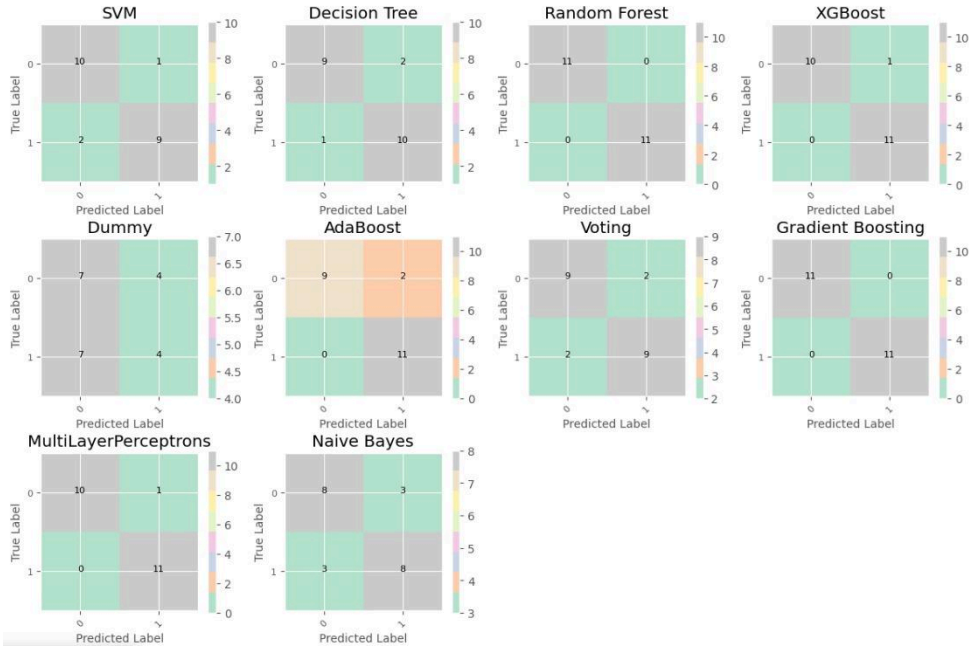
- La **curva ROC**: è una curva tracciata che rappresenta l'andamento della sensibilità, quindi tasso di vera positività (TPR) rispetto al reciproco della specificità, quindi il tasso della falsa positività (FPR).
- La **funzione di perdita** è la differenza tra l'accuratezza misurata durante la fase di training e l'accuratezza misurata durante la fase di testing.

Risultati ottenuti confrontando la **matrice di confusione**, calcolato sui modelli addestrati utilizzati.

	Model	Accuracy	Sensitivity	Specificity	PPV	NPV	AUROC
2	Random Forest	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
7	Gradient Boosting	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
3	XGBoost	0.954545	1.000000	0.909091	0.916667	1.000000	0.966942
8	MultiLayerPerceptrons	0.954545	1.000000	0.909091	0.916667	1.000000	0.975207
5	AdaBoost	0.909091	1.000000	0.818182	0.846154	1.000000	0.975207
0	SVM	0.863636	0.818182	0.909091	0.900000	0.833333	0.966942
1	Decision Tree	0.863636	0.909091	0.818182	0.833333	0.900000	0.942149
6	Voting	0.818182	0.818182	0.818182	0.818182	0.818182	0.950413
9	Naive Bayes	0.727273	0.727273	0.727273	0.727273	0.727273	0.859504
4	Dummy	0.500000	0.363636	0.636364	0.500000	0.500000	0.500000

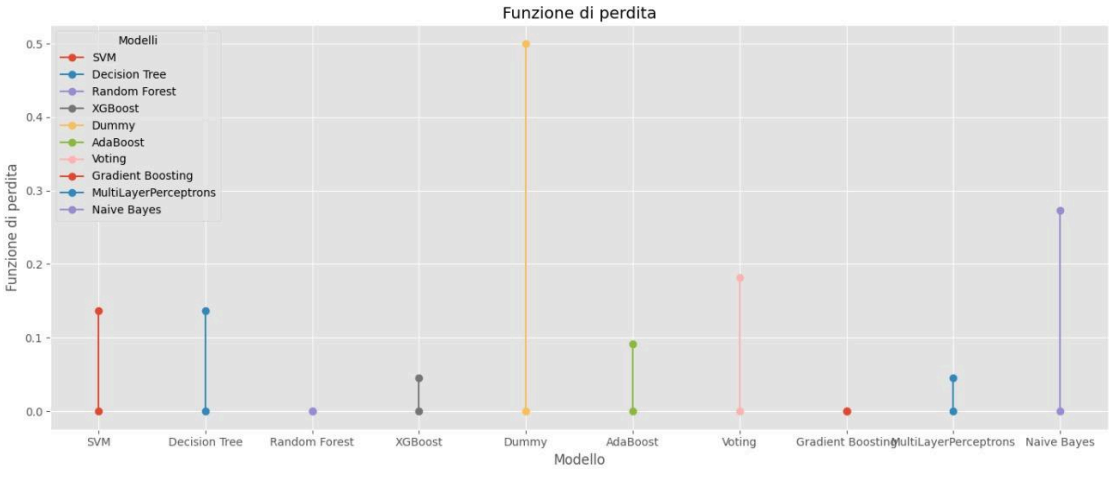
In termine di accuratezza, con un valore pari a 100%, i modelli più performanti sono Random Forest e Gradient Boosting; abbiamo ottenuti tra tutti i modelli un valore migliore dei livelli di sensibilità rispetto alla specificità, quindi i modelli riescono a classificare, in generale, meglio le classi positive rispetto alle negative.

Plottiamo le matrici di confusione per ogni modello.



- **TN** nel primo quadrante indica la corretta classificazione della classe negativa.
- **FP** nel secondo quadrante indica la missclassification della classe positiva.
- **FN** nel terzo quadrante indica la missclassification della classe negativa.
- **TP** nel quarto quadrante che inidica la corretta classificazione della classe positiva.

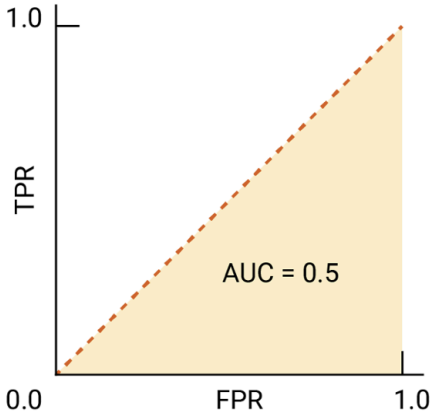
Rappresentazione grafica della funzione di perdita.

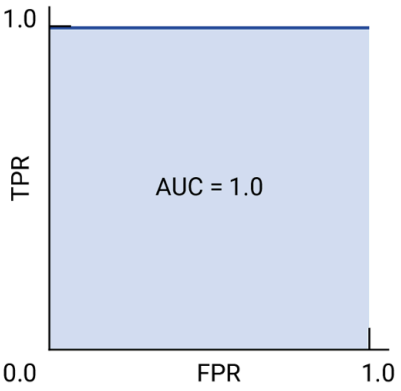


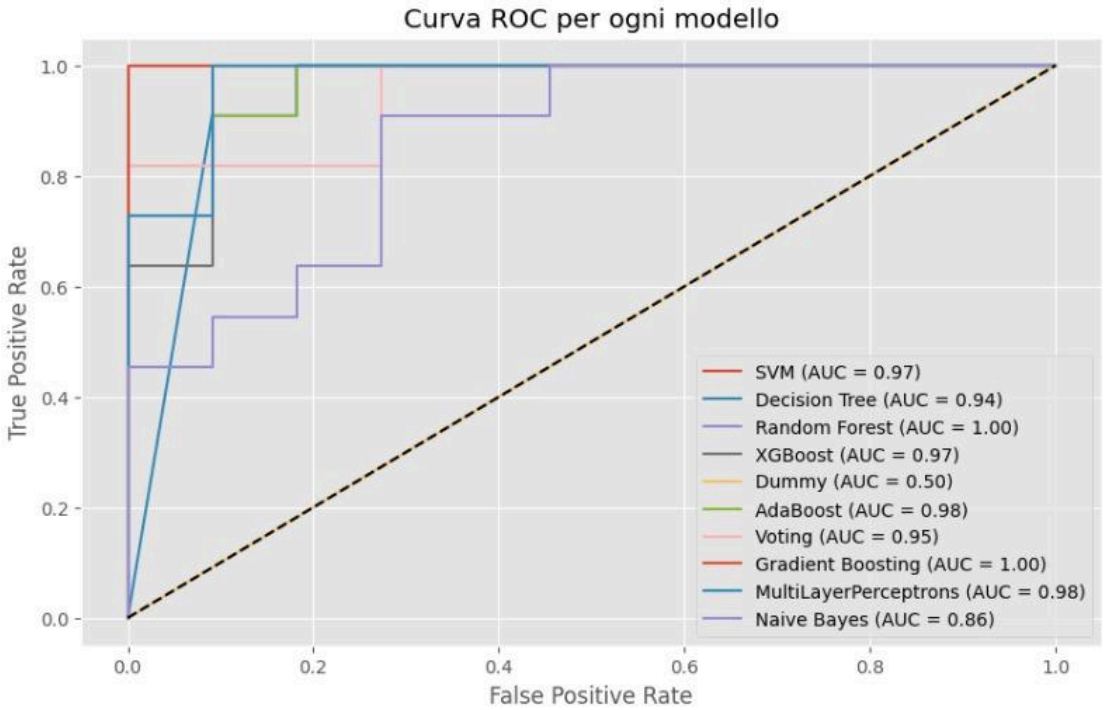
La funzione di perdita va a zero per i modelli di Random Forest e Gradient Boosting, in quanto abbiamo ottenuto un valore del 100% nell'accuratezza, al contrario la funzione di perdita maggiore è del modello Dummy che ha un'accuratezza del 50%, quindi pessima.

CURVA ROC

La Curva ROC, come detto prima, viene tracciata calcolando il tasso di veri positivi (TPR) e il tasso di falsi positivi (FPR), quindi rappresentando graficamente il TPR su quello FPR. Nell'immagine a sinistra otteniamo un caso di Curva ROC pessima, ed un valore di AUC = 0,5; con un modello di ipotesi assolutamente casuali. Nell'immagine a destra è rappresentato un valore di AUC = 1, con una rappresentazione di un modello ideale.







Qui vengono rappresentati i risultati dei nostri modelli: Random Forest e Gradient Boosting con risultati migliori. Il valore AUC di Dummy Classifier è pari a 0.5 quindi un modello pessimo, che ha la stessa capacità predittiva di un lancio della moneta.

Interpretazione dei risultati con il metodo visto a lezione

Ricercando informazioni del dataset fornito online potevamo vedere attraverso siti come BLAST che le sequenze biologiche di partenza sono di **Escheria Coli.**

Come abbiamo già detto, esistono **differenze importanti** tra i promotori procariotici ed i promotori eucariotici, questo ci consente di attuare già a priori tutta una serie di considerazioni in merito le sequenze consenso.

Dalla letteratura medica sappiamo che i **promotori procariotici hanno 2 sequenze consenso** principali:

- **Sequenza consenso1:** T (77%) A (76%) T (60%) A (61%) A (56%) T (82%)
- **Sequenza consenso2:** T (69%) T (79%) G (61%) A (56%) C (54%) A (54%)

Possiamo vedere che la sequenza consenso non appare sempre in tutti i promotori nello stesso modo, ma c'è una frequenza di distribuzione variabile.

Con i metodi utilizzati abbiamo estratto la caratteristica **PSeEIIP** la quale esprime *le proprietà elettroniche di carica nucleotidiche* e la selezione delle 6 caratteristiche salienti quali: *PseEIIP_TTA, PseEIIP_TCA, PseEIIP_GAG, PseEIIP_TTT, PseEIIP_AAA, PseEIIP_ATA*, il nostro modello ha imparato a comprendere come ai fini del nostro task la **sequenza TA ripetuta** sia importante per effettuare la classificazione.

Il nostro modello sta imparando a classificare nei dati di training combinando le caratteristiche più salienti; spostandosi nei dati di *testing* siamo riusciti ad ottenere, con modelli ottimali quali Multy-Layer-Perceptrons e XG boost del **95%** e Random Forest e Gradient Boosting del **100% di accuratezza.**

Conclusioni

1. La percentuale di trovare la sequenza TATAAT nel file excel per intero o comunque una sua sotto-sequenza di 5 nucleotidi come TATAA o ATAAT è molto bassa(circa 17/53), questo perché la sequenza dei nucleotidi non si distribuisce sempre nello stesso modo;
2. Quanto affermato sopra, implica che una persona che prova a classificare una sequenza promotrice ha solamente circa il 20% di riuscire a classificarla correttamente, e dovrà quindi applicare una serie di ragionamenti prima di dire se la sequenza è promotrice o meno, infatti passando da 6 nucleotidi TATAAT a 4 nucleotidi come TAAT o TATA iniziamo a trovare queste sotto-sequenze sia nelle sequenze promotrici sia in quelle non promotrici;
3. Abbiamo cercato come avviene il riconoscimento di sequenze promotrici in laboratorio ed abbiamo trovato una tecnica nota come '**CAGE**' ma ce ne sono anche altre.
4. Il punto centrale, è che se prendiamo come riferimento 'CAGE', questa tecnica riesce a raggiungere livelli di accuratezza importanti per il riconoscimento dei promotori (queste tecniche hanno livelli di accuratezza che vanno dal 95% in poi), il problema è che le metodiche come 'CAGE' richiedono tempo e costi per l'applicazione.
5. Situazioni come il Corona Virus hanno dimostrato come a volte la velocità sia tutto ed è proprio qua che subentrano queste tecniche;
6. Attraverso l'utilizzo di questi metodi di classificazione le regioni promotrici potrebbero essere identificate con elevata accuratezza oltre che velocità e ciò fornisce analisi per ulteriori ricerche biologiche e per la medicina di precisione;

7. Possiamo vedere come la ricerca si muove molto attraverso l'utilizzo delle reti neurali per la risoluzione di questo tipo di problematiche (https://pmc.ncbi.nlm.nih.gov/articles/PMC6848157/#s3).

In conclusione, non possiamo dire molto in merito la reale efficienza del nostro modello perché comunque l'addestramento è stato fatto solo su 106 sequenze iniziali e le sequenze promotrici di procarioti per quanto più semplici di quelle eucarioti hanno comunque ulteriori gradi di variabilità, infatti anche E. Coli ha diversi fattori sigma che riconoscono le diverse sequenze e ciò significa che se ampliamo i nostri dati molto probabilmente i livelli di accuratezza si abbassano notevolmente, tuttavia si evince chiaramente come l'utilizzo di questo tipo di metodiche di IA sia essenziale per velocizzare la ricerca.

Promoter on Deep Learning

Abbiamo analizzato un metodo alternativo a quello mostrato a lezione, ed abbiamo applicato allo stesso task il deep learning; Il problema è che le reti neurali hanno bisogno di molti dati e noi abbiamo solo 106 sequenze

Quindi il **deep learning** per affrontare un problema importante nella genomica funzionale: **il riconoscimento di sequenze promotrici nel DNA.**

L'obiettivo è utilizzare una rete neurale in grado di andare a **trovare eventuali "features"** presenti nelle sequenze promotrici di DNA.

I nucleotidi più frequentemente riconosciuti che determinano la **sequenza consenso** nei promotori procariotici sono **TATAAT, TTGACA** la rete neurale deve apprenderlo

Flusso di lavoro con il deep learning

Il **flusso di lavoro** che abbiamo seguito con questo metodo vede: “ Importazione del Dataset, Preprocessing dei dati, Selezione dell'architettura, Training, Valutazione dei risultati ed Interpretazione dei risultati”

Importazione del Dataset e Preprocessing dei dati;

Abbiamo suddiviso il dataset in 2 file “labels.txt” e “sequences.txt”; La nostra rete neurale riconosce un determinato formato, quindi tramite una tecnica chiamata “**onehotencoder**” trasformiamo le sequenze in un formato leggibile alla rete; Questa tecnica, codifica ogni base in una sequenza sotto forma di vettore quadridimensionale, con una dimensione separata per ciascuna base.

Splitaggio dei Dati - Selezione dell’architettura e Training

Viene eseguito lo **splittaggio dei dati** in: training, testing e validation, attribuendo rispettivamente 70-10-20% con seme randomico pari a 42; Usiamo una semplice **CNN**: Conv1d; Questa rete dovrà imparare ad estrarre le caratteristiche più importanti, nel nostro caso la migliore combinazione di nucleotidi ai fini del task.

Scegliamo come **principali parametri:** “kernel_size=12, filters=128, maxpooling1d=4, dense=128, epochs=50”; Otteniamo come numero totale di parametri 186,882; Nello step

successivo eseguiamo il training; Plottiamo i grafici per vedere l'accuratezza e la funzione di perdita.

Interpretazione dei risultati con il deep learning

1. Il modello riesce a classificare con **un'accuratezza di circa l'80%**
2. La differenza tra la **funzione di loss in training e validation** è molto alta ed il modello va in **over-fitting**
3. Questo modello impara ad **estrarre i motivi** che potrebbero essere alla base di una sequenza promotrice