Relazione Tirocinio

Introduzione

Durante il tirocinio sono stati visti problemi di Machine Learning, precisamente *clustering* e *classificazione*; il clustering l'ho applicato sul dataset Iris, invece la classificazione sul dataset Iris e sulle sequenze genomiche.

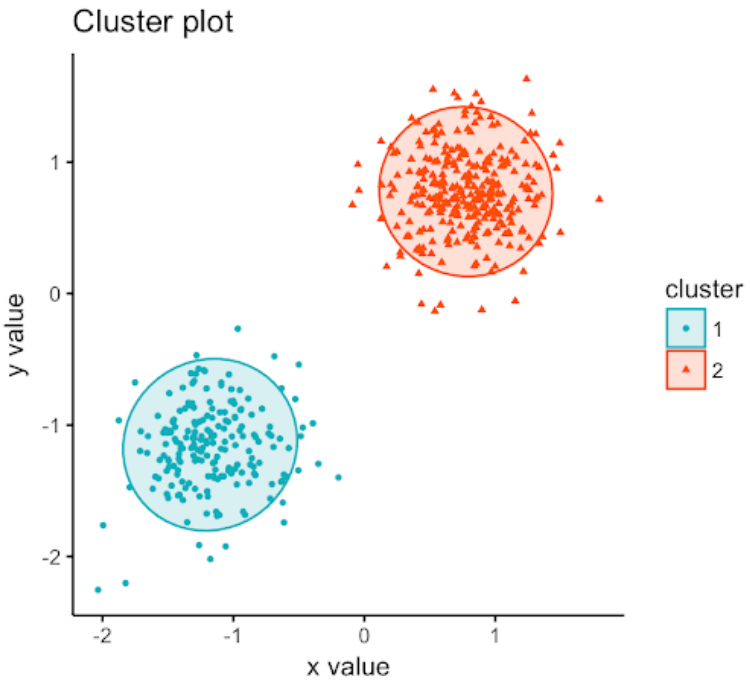
Per la realizzazione degli algoritmi ho usato il linguaggio di programmazione Python e ho importato le librerie:

- Pandas e Numpy per la gestione di strutture dati
- Sklearn per le funzioni di machine learning
- Pylab per plottare i risultati su matrici di confusione o grafi

Clustering

Il **Clustering** individua gruppi (cluster) di pattern con caratteristiche simili.

- Le classi del problema non sono note e i pattern non etichettati -> la natura non supervisionata del problema lo rende più complesso della classificazione.
- Spesso nemmeno il numero di cluster è noto a priori
- I cluster individuati nell'apprendimento possono essere poi utilizzati come classi.



Classificazione

La **Classificazione** assegna una classe a un pattern.

- Necessario apprendere una funzione capace di eseguire il mapping dallo spazio dei pattern allo spazio delle classi
- Si usa spesso anche il termine riconoscimento.
- Nel caso di 2 sole classi si usa il termine binary classification, con più di due classi multi-class classification.

La Classe è insieme di pattern aventi proprietà comuni.

Apprendimento

Distinguiamo tre tipi di apprendimento:

Supervisionato (Supervised): sono note le classi dei pattern utilizzati per l'addestramento.

- il training set è etichettato.
- situazione tipica nella classificazione, regressione e in alcune tecniche di riduzione di dimensionalità (es. Linear Discriminant Analysis).

Non Supervisionato (Unsupervised): non sono note le classi dei pattern utilizzati per l'addestramento.

- il training set non è etichettato.
- situazione tipica nel clustering e nella maggior parte di tecniche di riduzione di dimensionalità.

Semi-Supervisionato (Semi-Supervised)

- il training set è etichettato parzialmente.
- la distribuzione dei pattern non etichettati può aiutare a ottimizzare la regola di classificazione.

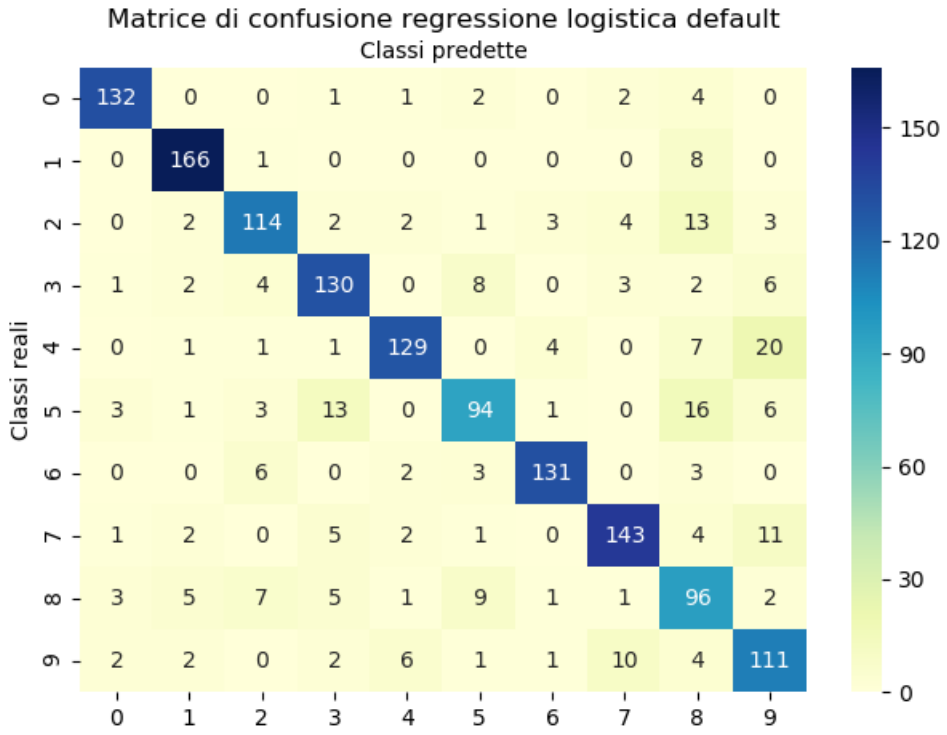
Valutazione delle prestazioni

In un problema di Classificazione, l'accuratezza di classificazione [0…100%] è la percentuale di pattern correttamente classificati. L'errore di classificazione è il complemento.

$$Accuratezza = pattern\ correttamente\ classificat / pattern\ classificati$$

$$Errore = 100\% - Accuratezza$$

La matrice di confusione (confusion matrix) è molto utile nei problemi di classificazione (multiclasse) per capire come sono distribuiti gli errori.



Dato un classificatore binario e **T = P + N** pattern da classificare (**P** positivi e **N** negativi), il risultato di ciascuno dei tentativi di classificazione può essere:

- *True Positive (TP):* un pattern positivo è stato correttamente assegnato ai positivi.
- *True Negative (TN):* un pattern negativo è stato correttamente assegnato ai negativi.
- *False Positive (FP):* un pattern negativo è stato erroneamente assegnato ai positivi. Detto anche errore di Tipo I o False.
- *False Negative (FN):* un pattern positivo è stato erroneamente assegnato ai negativi. Detto anche errore di Tipo II o Miss.

$$TPR\ True\ Positive\ Rate = TP/P$$

$$TNR\ (True\ Negative\ Rate) = TN/N$$

$$FPR\ (False\ Positive\ Rate) = FP/N$$

$$FNR\ (False\ Negative\ Rate) = FN/P$$

Con questa notazione l'accuratezza di classificazione può essere scritta come:

$$Accuracy = TP + TN \,/\, T$$

$$T = (P + N)$$

La **Precision** indica quanto è accurato il sistema.

$$Precision = TP \,/\, TP + FP$$

La **Recall** quanto è selettivo.

$$Recall = TP \,/\, TP + FN =$$
$$= TP \,/\, P = TPR$$

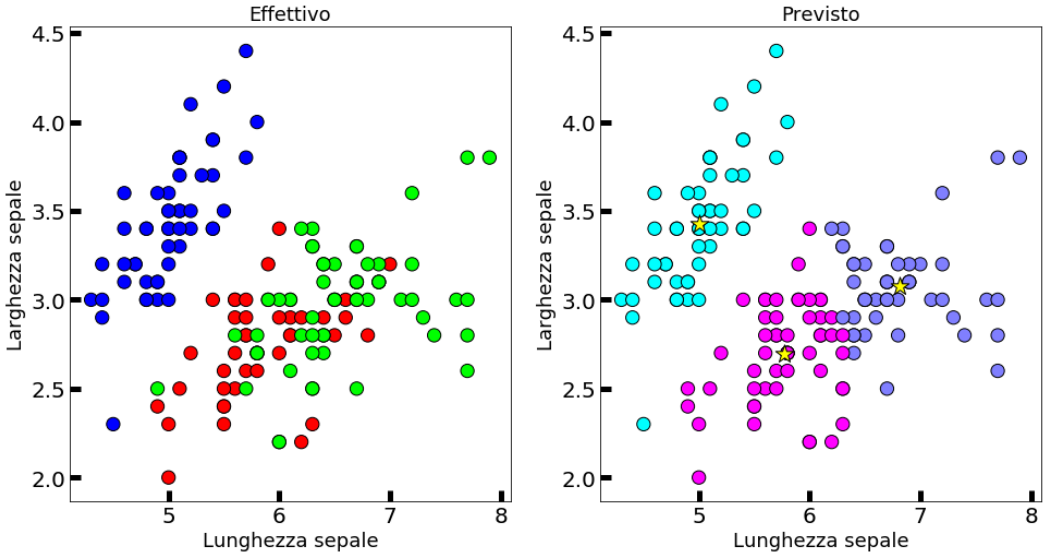
Iris Dataset

Il **dataset Iris** è un dataset multivariato, consiste in 150 istanze di Iris misurate e classificate secondo tre specie: *Iris setosa*, *Iris virginica* e *Iris versicolor*.

Le quattro caratteristiche (feature) considerate sono la lunghezza e la larghezza del sepalo e del petalo.

K-means

L'algoritmo **K-means** è un algoritmo di clustering partizionale che permette di suddividere un insieme di oggetti in K gruppi sulla base delle loro caratteristiche. Ho applicato questo algoritmo sul dataset Iris.



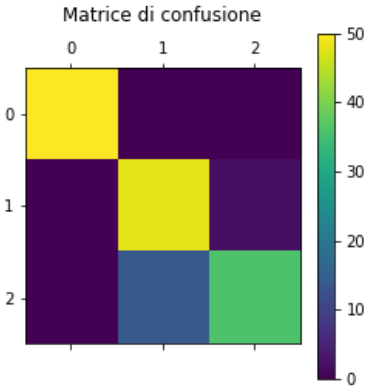
Prima di tutto ho caricato il dataset e preso i dati riguardanti la lunghezza e la larghezza del sepalo e del petalo (input) con le corrispondenti etichette (output), poi ho costruito il modello k-means con 3 cluster (gruppi) e trovato i punti centrali.

I risultati ottenuti sono i seguenti:

134 pattern corretti su 150
Accuratezza: 89.33333333333333 %
Errore: 0.10666666666666669

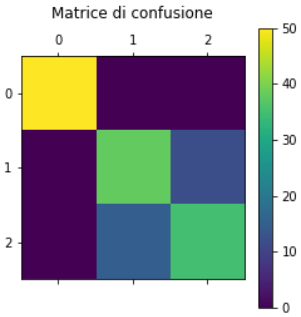
0 = setosa
1 = versicolor
2 = virginica

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	0.77	0.96	0.86	50
2	0.95	0.72	0.82	50
accuracy			0.89	150
macro avg	0.91	0.89	0.89	150
weighted avg	0.91	0.89	0.89	150



- ***K-means 2 feature***

  Ho eseguito lo stesso algoritmo ma considerando solo le prime due caratteristiche

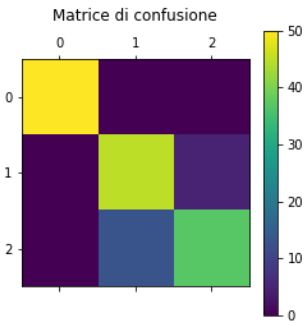


123 pattern corretti su 150
Accuratezza: 82.0 %
Errore: 0.18000000000000005

Il risultato ottenuto è meno accurato rispetto al precendente

- ***K-means 3 feature***

  Ho eseguito lo stesso algoritmo prendendo le prime tre caratteristiche

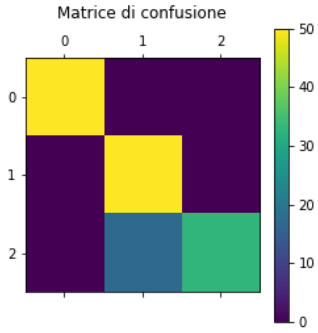


132 pattern corretti su 150
Accuratezza: 88.0 %
Errore: 0.12

Il risultato è molto simile a quello ottenuto con 4 caratteristiche

- ***K-means 4 cluster***

Questa volta ho costruito il modello k-means con 4 cluster
Il quarto gruppo di cluster in un secondo momento lo unisco a uno dei tre gruppi in base all'occorrenza più frequente

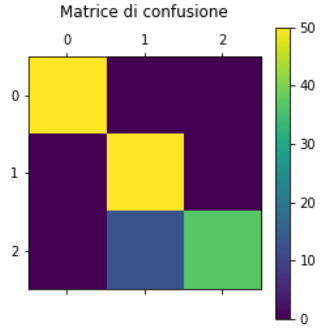


133 pattern corretti su 150
Accuratezza: 88.66666666666667 %
Errore: 0.11333333333333329

Il risultato è molto simile a quello ottenuto con 3 cluster

- ***K-means 5 cluster***

Ho costruito il modello k-means con 5 cluster

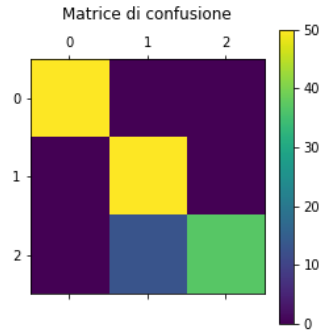


137 pattern corretti su 150
Accuratezza: 91.33333333333333 %
Errore: 0.08666666666666667

Il risultato è migliore perché c'è maggiore precisione sulla separazione delle etichette

- ***K-means 6 cluster***

Ho costruito il modello k-means con 6 cluster



137 pattern corretti su 150
Accuratezza: 91.33333333333333 %
Errore: 0.08666666666666667

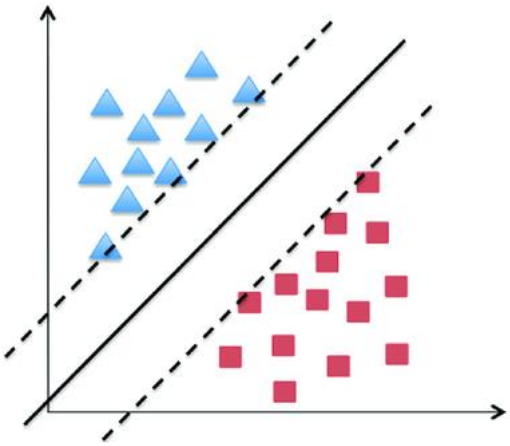
Il risultato è uguale al precedente perché il sesto cluster è aggiuntivo

SVM

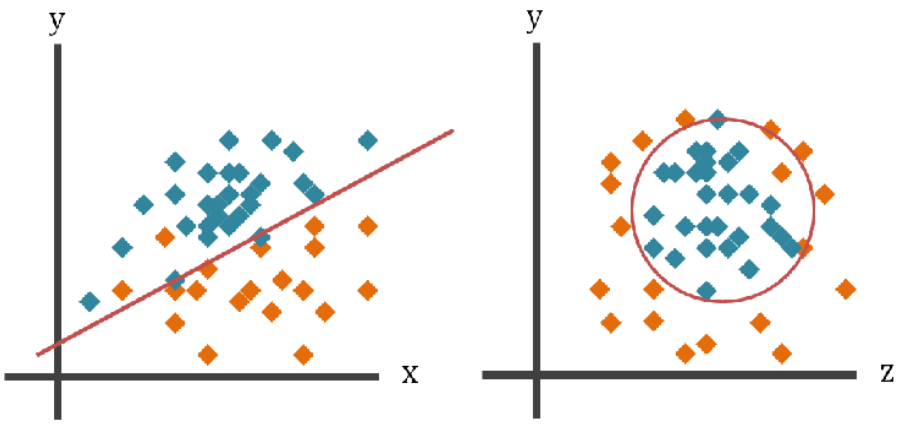
SVM (*Support-Vector Machines*) sono dei modelli di apprendimento supervisionato associati ad algoritmi di apprendimento per la regressione e la classificazione.

Dato un insieme di esempi per l'addestramento (*training set*), ognuno dei quali etichettato con la classe di appartenenza fra le due possibili classi, un algoritmo di addestramento per le SVM costruisce un modello che assegna i nuovi esempi ad una delle due classi, ottenendo quindi un classificatore lineare binario non probabilistico.

Un modello SVM è una rappresentazione degli esempi come punti nello spazio, mappati in modo tale che gli esempi appartenenti alle due diverse categorie siano chiaramente separati da uno spazio il più possibile ampio. I nuovi esempi sono quindi mappati nello stesso spazio e la predizione della categoria alla quale appartengono viene fatta sulla base del lato nel quale ricade.



Oltre alla classificazione lineare è possibile fare uso delle SVM per svolgere efficacemente la classificazione non-lineare utilizzando il metodo kernel, mappando implicitamente i loro input in uno spazio delle feature multi-dimensionale.



Distinguiamo i seguenti tipi di kernel SVM:

- **Linear**
- **RBF**
- **Poly**
- **Sigmoid**
- **Precomputed**

Ho eseguito l'SVM su dataset iris utilizzando il modello lineare e quello rbf con le seguenti funzioni:

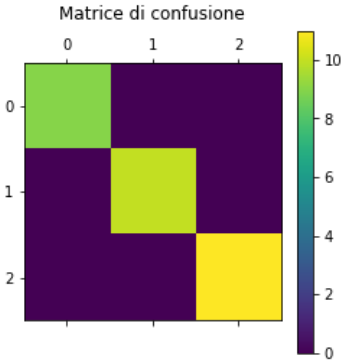
svm.SVC(kernel='linear', C=1).fit(X_train, y_train)

svm.SVC(kernel='rbf', gamma=0.7, C=1).fit(X_train, y_train)

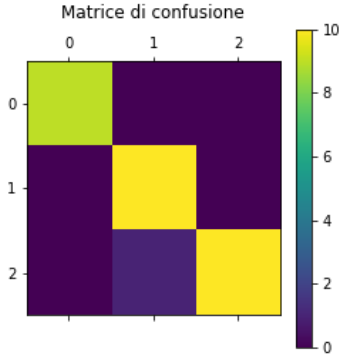
I parametri C e gamma sono rispettivamente il parametro di regolarizzazione e il kernel coefficient.

I risultati ottenuti sono i seguenti:

LINEARE



RBF



Training, Validation, Set

- **Training Set** (Train): è l'insieme di pattern su cui addestrare il sistema, trovando il valore ottimo per i parametri Θ.
- **Validation Set** (Valid): è l'insieme di pattern su cui tarare gli iperparametri H (ciclo esterno).
- **Test Set** (Test): è l'insieme di pattern su cui valutare le prestazioni finali del sistema.

Grid Search

La griglia di ricerca (**Grid Search**) è il modo più semplice di eseguire l'ottimizzazione dell'iperparametro.

Si tratta di una ricerca esaustiva attraverso un sottoinsieme specificato manualmente nello spazio dell'iperparametro di un algoritmo di apprendimento; il sistema è valutato su tutte le combinazioni di valori di tutti gli iperparametri.

Ho applicato il grid search sul dataset Iris, gli iperparametri specificati nella funzione sono

parameters = {'kernel': ['linear', 'rbf', 'poly', 'sigmoid'], 'gamma':[0.1, 0.01], 'C': [1, 10]}

In questo modo sono andata a creare dei modelli SVM con

- *kernel*: lineare, rbf, polinomiale e sigmoid;
- variabile *gamma* con valori: 0.1, 0.01
- *C* che assume valori: 1, 10.

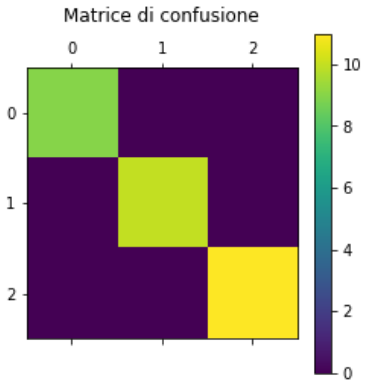
L'algoritmo ha elaborato tutte le possibili combinazioni di SVM specificate nei relativi parametri e ha stampato i risultati riguardo il modello migliore.

I risultati che ho ottenuto sono i seguenti:

Accuracy del modello migliore: 0.9750

{'C': 1, 'gamma': 0.1, 'kernel': 'linear'}

I modello migliore risulta essere quello lineare



K-Fold Cross-Validation

La **k-fold cross-validation** consiste nella suddivisione del dataset totale in k parti di uguale numerosità e, ad ogni passo, la k-esima parte del dataset viene ad essere il *validation dataset*, mentre la restante parte costituisce il *training dataset*.

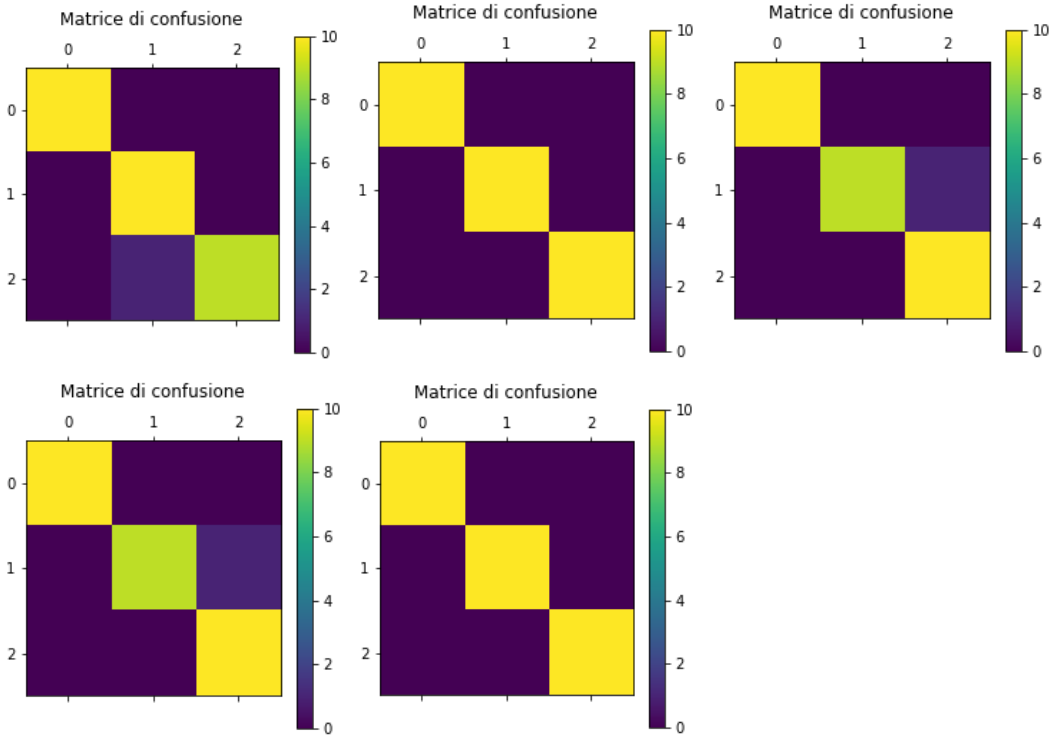
Così, per ognuna delle *k* parti si allena il modello, evitando quindi problemi di overfitting.

Ho applicato questo modello, con l'uso dell'svm lineare, sul dataset Iris utilizzando la seguente funzione per suddividere la sequenza di etichette date in input in 5 parti uguali

kf = StratifiedKFold(n_splits=5)

Con stratified i folds vengono scelti preservando una percentuale di campioni per ogni classe.

I risultati ottenuti sono i seguenti:



K-mer

I **K-mers** sono sottosequenze di lunghezza *k* contenenute in sequenze biologiche.

L'obiettivo è quello di realizzare un dataframe di sottosequenze (kmers) prendendo in input un file fasta con 6320 sequenze geniche suddivise in 13 classi diverse.

Per fare ciò ho elaborato il file contando le frequenze di pattern di lunghezza K presenti in ogni sequenza e riportato i risultati lungo le colonne rappresentanti i $\mathbf{4^k}$ kmer (disposizioni di {A,C,G,T}).
Il dataframe creato ha come indice di riga il nome della sequenza, invece le colonne sono indicizzate dalle sottosequenze e dal nome della classe.

Nome	Feature_1		Feature_n	Classe
	….			
	….			
	….			
	….			

Es. K=2

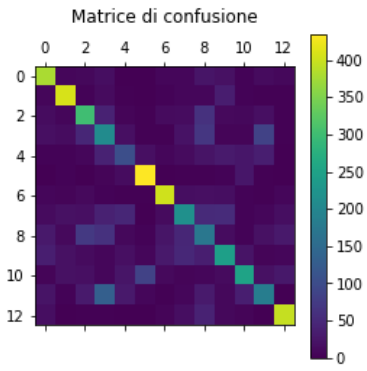
AA	AC	AG	AT	…	TA	TC	TG	TT
0	2	1	0		0	3	4	1

Ho eseguito l'algoritmo per K che va da 3 a 7 e riportato i dataframe ottenuti su file csv.

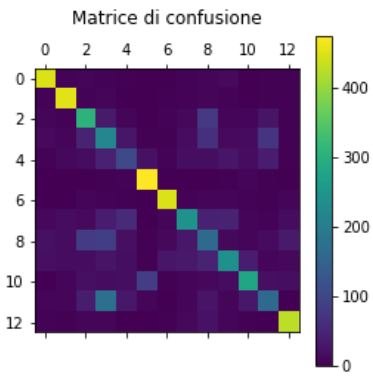
In un secondo momento ho utilizzato i dataframe per predire le classi di genomi tramite SVM.
Per ogni file ho estrapolato i dati corrispondenti alle feature, applicato la funzione StratifiedKFold con 5 split e sommato le matrici di confusione ottenute in ogni plit.

I risultati ottenuti sono i seguenti:

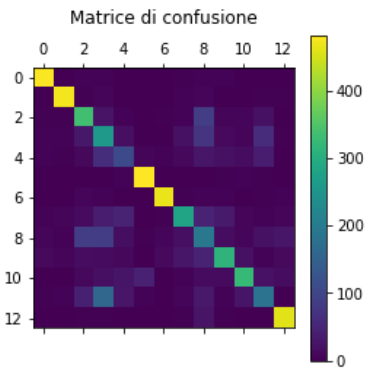
K = 3



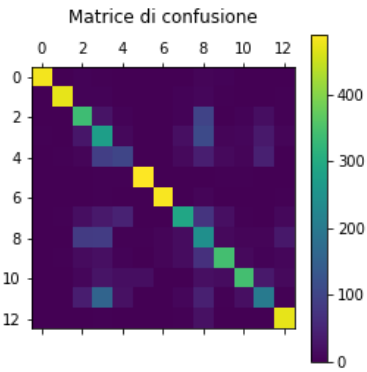
K = 4



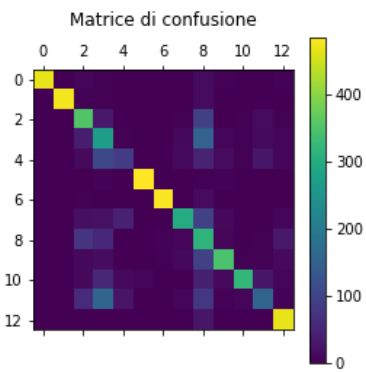
K = 5



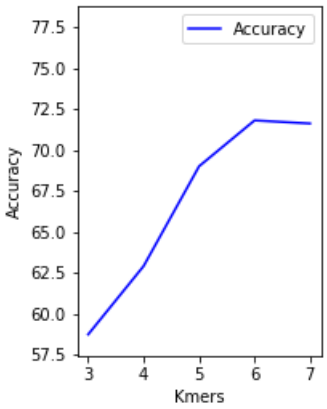
K = 6

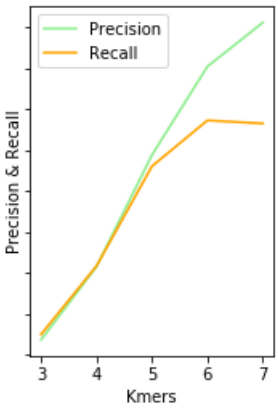


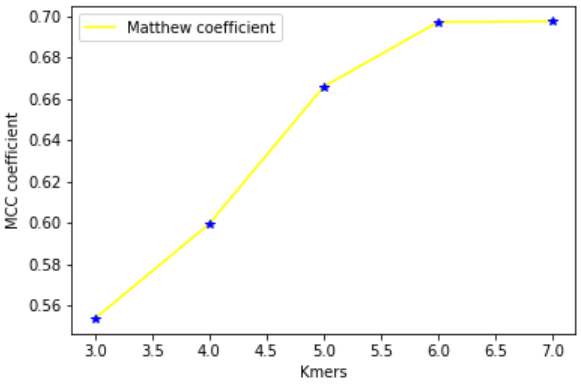
K = 7



Infine ho fatto un confronto di accuracy, precision, recall e MCC score tra i diversi dataframe mediando i risultati per ogni K:







Queste sono le funzioni che ho utilizzato per il calcolo dei valori:

score = accuracy_score(y_test, pred)*100

precision = precision_score(y_test, pred, average='weighted')*100

recall = recall_score(y_test, pred, average='weighted')*100

MCC = matthews_corrcoef(y_test, pred)

Ho scelto ‘weighted’ come parametro di media perché calcola le metriche per ciascuna etichetta e trova la media ponderata per il supporto (numero di istanze effettive per ciascuna etichetta).

Si può notare come l’accuratezza e gli altri valori crescono all’aumentare del numero di K.