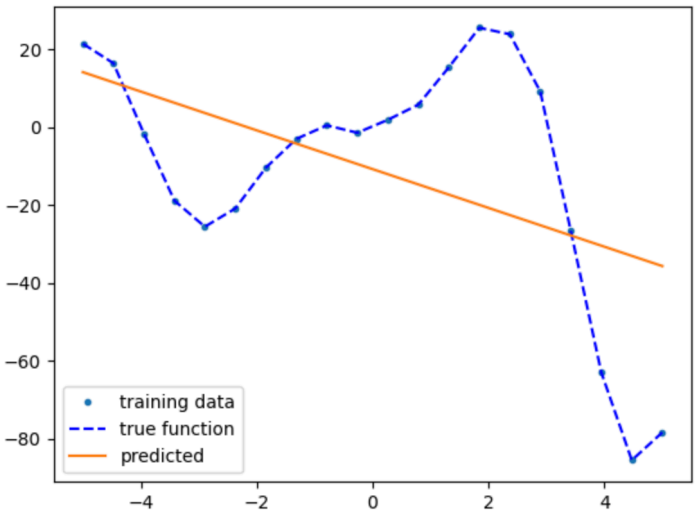
Francesca Pecorari SM3201259

Challenge two - Introduzione al Machine Learning

Parte 1

Dopo aver creato un dataset si procede nell'analisi dapprima applicando una Ridge Regression, continuando poi con l'utilizzo della kernelizzazione con kernel gaussiano e polinomiale, si commentano di seguito i risultati:

Ridge Regression

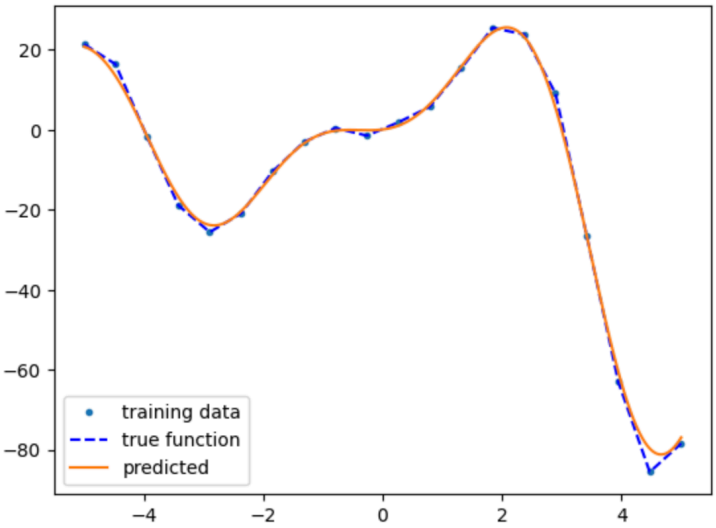


Dal grafico ottenuto si capisce che la regressione ridge non ha una buona performance sul dataset da analizzare, i dati non sono lineari e la retta di predizione non si adatta minimamente a questi, infatti l'errore quadratico medio che si trova è molto alto, pari a 719.2607027117482.
Si procede quindi con la kernelizzazione per vedere l'effetto di questa tecnica sulla predizione.

Gaussian Kernel Ridge Regression

Il primo tipo di kernel che è stato utilizzato in questo caso è quello gaussiano, i parametri sono stati calcolati con la grid search tra quelli nei seguenti range:

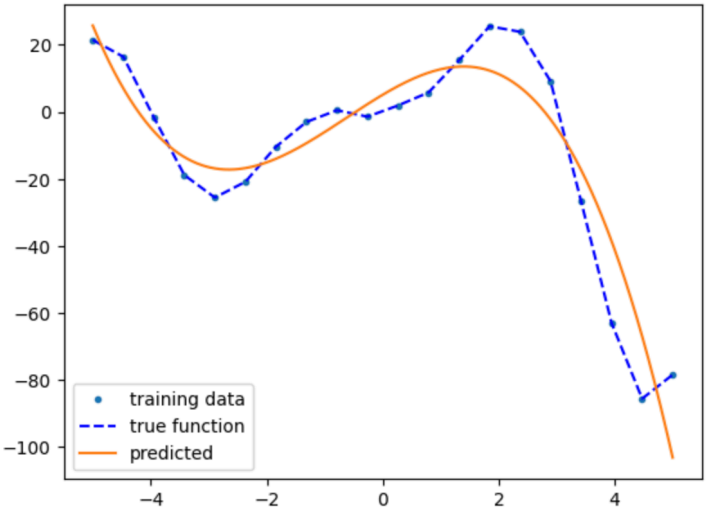
- per il parametro di regolarizzazione 0.1, 1.0, 10.0
- per il coefficiente del kernel 0.1, 1.0, 10



Dal grafico ottenuto si vede che la curva di predizione si adatta molto meglio ai nostri dati, infatti il valore del MSE trovato è molto più basso 0.1386169572646159, vicino a zero, questo potrebbe significare anche che si stia facendo overfitting.
Il valore dei parametri considerati migliori e quindi quello utilizzato è 0.1, lo stesso sia per il parametro di regolarizzazione che per il coefficiente del kernel.

Polynomial Kernel Ridge Regression

Il secondo tipo di kernel utilizzato è quello polinomiale, anche in questo caso i parametri sono stati calcolati usando la grid search tra i seguenti valori:
- per il parametro di regolarizzazione [0.1, 1.0, 10.0]
- per il grado del kernel polinomiale [2, 3, 4]
- per il coefficiente del kernel (0.1, 1.0, 10)



Dal grafico ottenuto si nota che la predizione è migliore rispetto a quella della Ridge Regression non kernelizzata, ma peggiore rispetto a quella con il kernel gaussiano. In questo caso si ottiene un MSE di 100.73555761387001, che è comunque molto alto.
I valori scelti per i parametri sono:
- 1 per la regolarizzazione
- 3 per il grado del polinomio
- 1 per il coefficiente del kernel

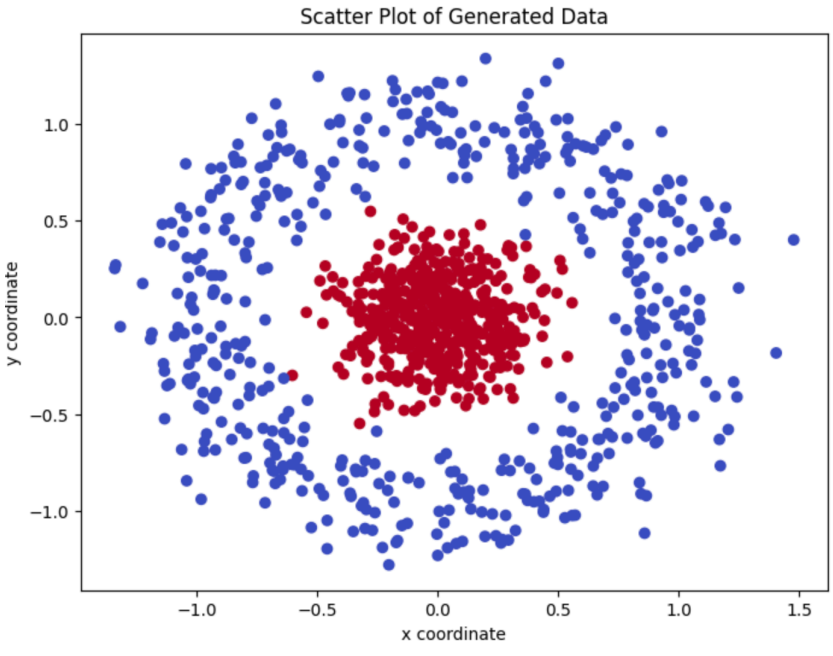
Il kernel migliore da utilizzare risulta quindi essere quello gaussiano, le cui predizioni sono già accurate, si potrebbe approfondire la bontà del modello testandolo su ulteriori dati se possibile.

Parte 2

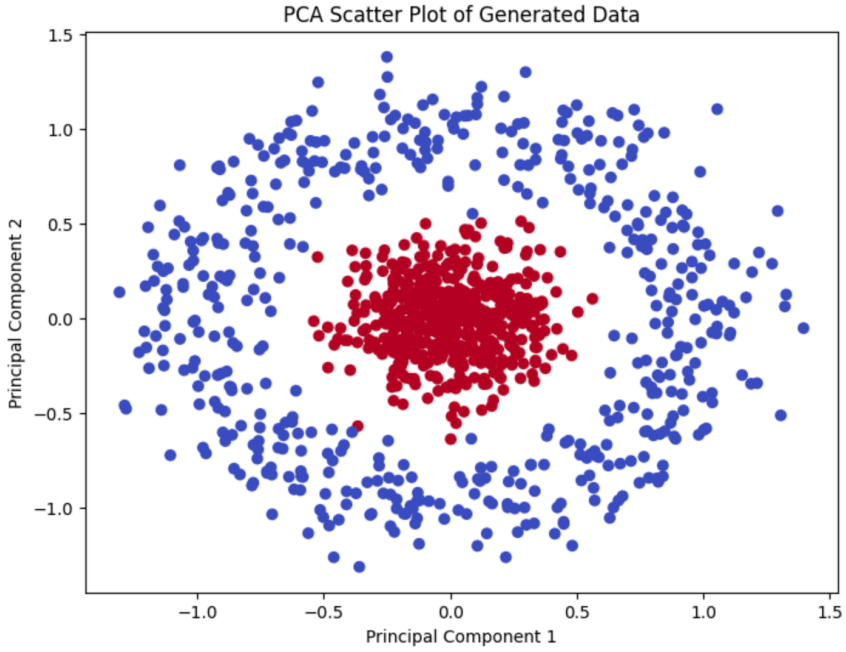
Nella seconda parte della challenge è richiesto di applicare la PCA a un set di dati creato con "sklearn.datasets.make_circles()", successivamente è richiesto di applicare la KernelPCA con un kernel a scelta, in questo caso è stato utilizzato il kernel gaussiano; è stata poi verificata l'accuratezza del kernel sui dati di test utilizzando SVM.

PCA

Inizialmente si visualizza la disposizione dei dati con uno scatter plot e poi si procede nell'analisi delle componenti principali.

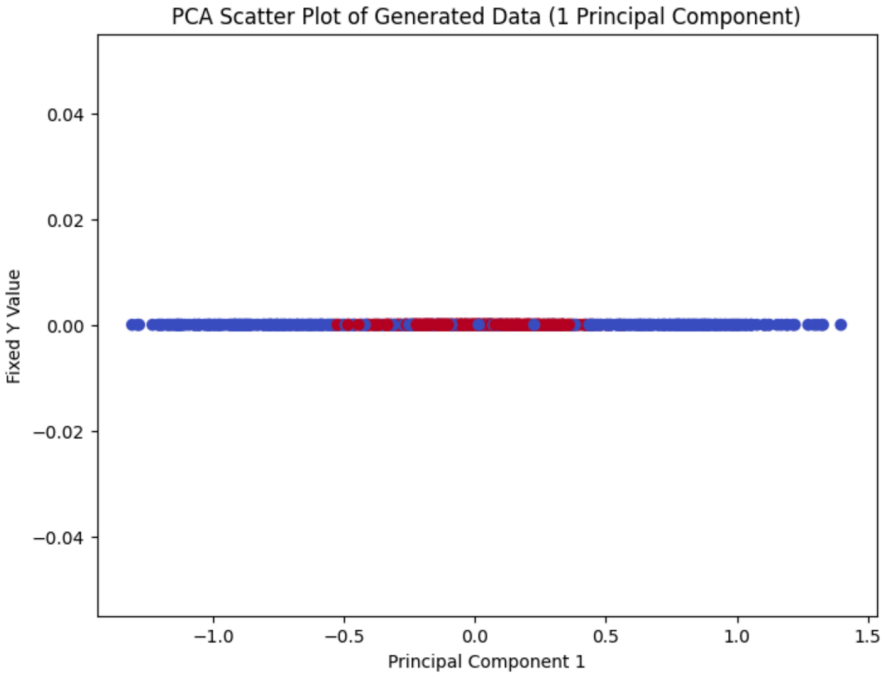


Applicando una PCA classica con numero di componenti pari a due il grafico che si ottiene è il seguente.



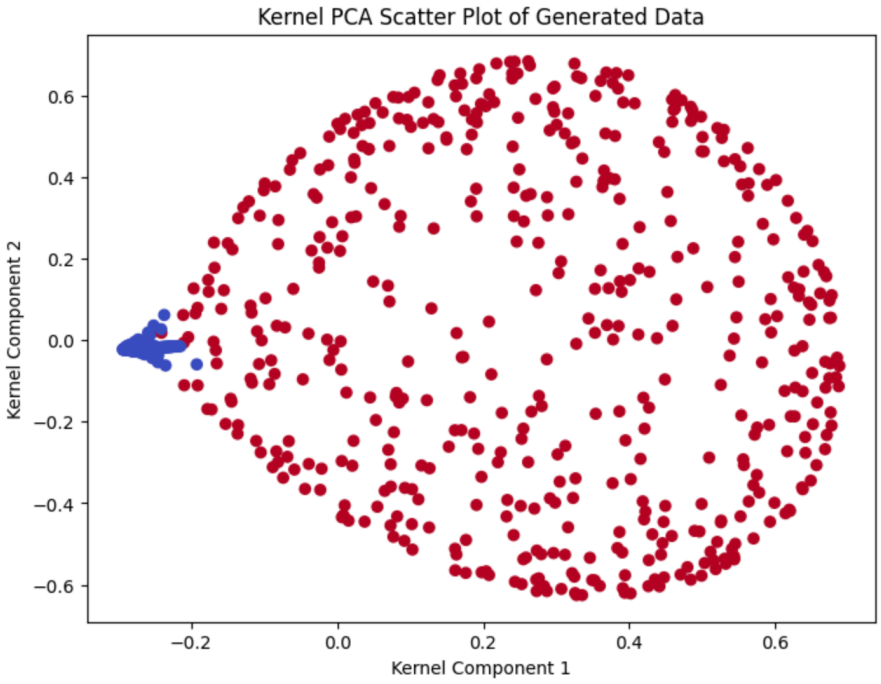
Si nota che essendo i dati non lineari o separabili linearmente la PCA classica non riesce a catturarne la struttura, procediamo quindi nell'analisi.

Anche riducendo il numero di componenti principali a uno i dati non sembrano separasi bene secondo la loro struttura, notiamo infatti che le due categorie sono completamente mescolate tra loro.



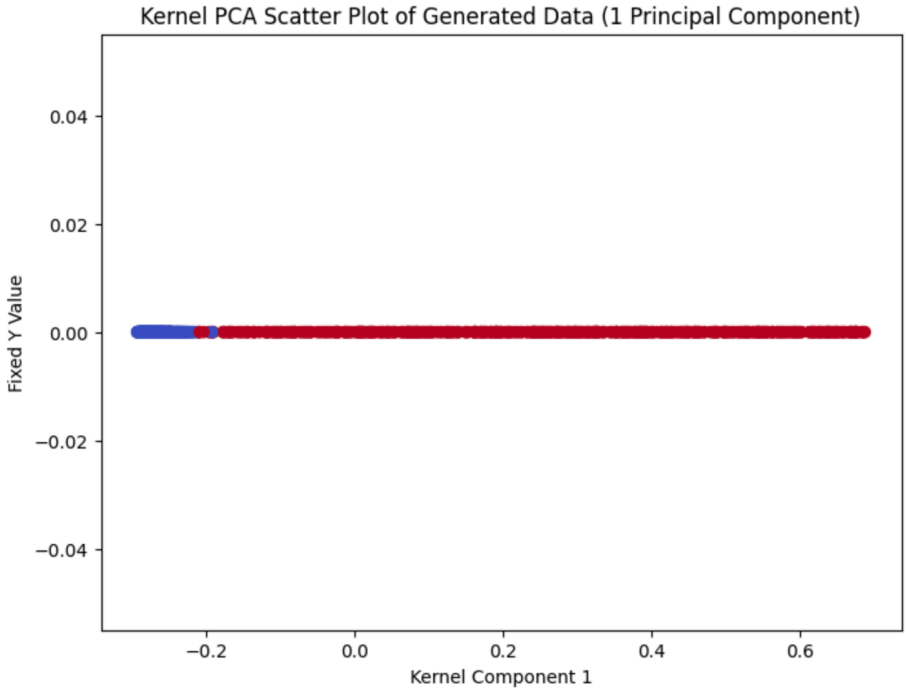
Applichiamo quindi una KernelPCA con kernel gaussiano.

Il grafico delle componenti principali che si ottiene in questo caso, avendo impostato il numero di PC a due, è il seguente.



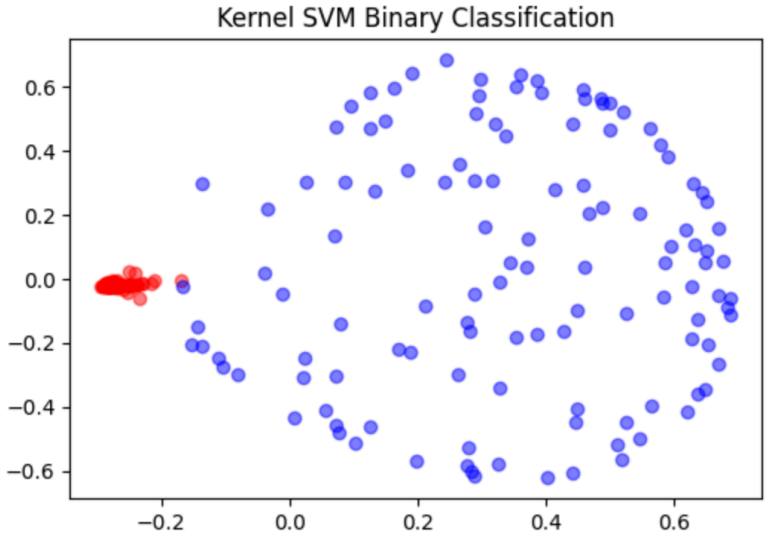
Si nota come in questo caso il modello sia in grado di cogliere la sottostruttura dei dati riuscendo anche a individuare relazioni non lineari tra i dati.

Anche riducendo il numero di componenti principali a uno si riesce comunque a cogliere la sottotruttura dei dati.



Applicando SVM con kernel gaussiano si ottiene un'accuracy sui dati di test di 0.984, un valore molto alto, questo significa che le label dei dati sono state predette per la grande maggioranza in modo corretto e che il modello ha imparato dei pattern importanti nella fase di training che riescono a generalizzare bene su dati mai visti prima.

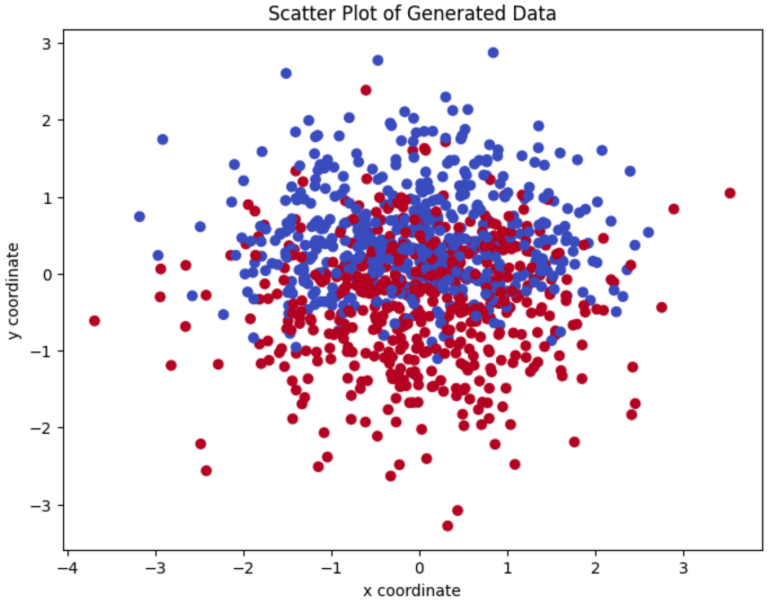
Si vede anche dal grafico ottenuto che le due classi di dati sono ben separate e distinte.



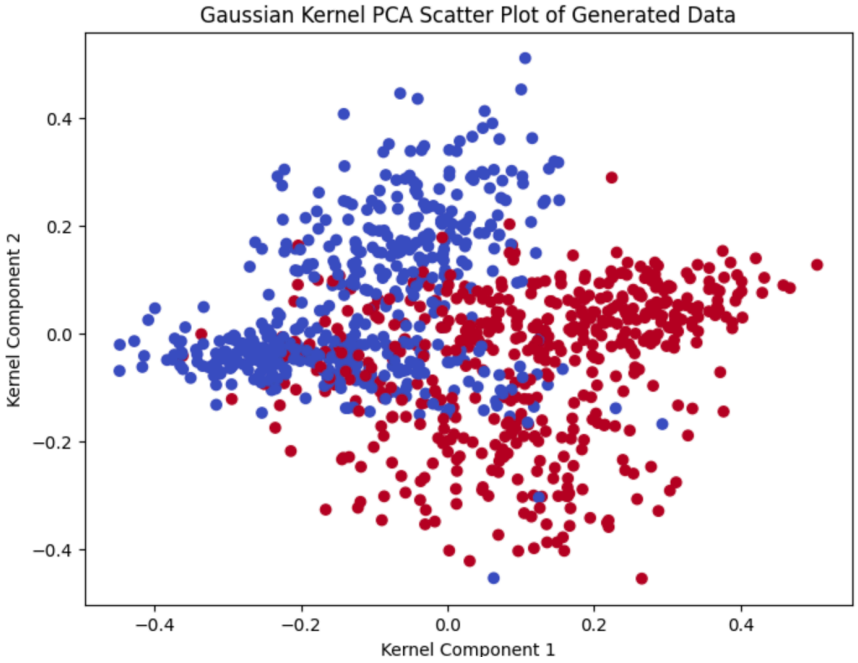
Parte 3

Nell'ultima parte la richiesta era quella di fare una KernelPCA con diversi tipi di kernel e confrontarli utilizzando anche SVM. L'analisi è stata eseguita su un dataset creato con "sklearn.datasets.make_classification()".

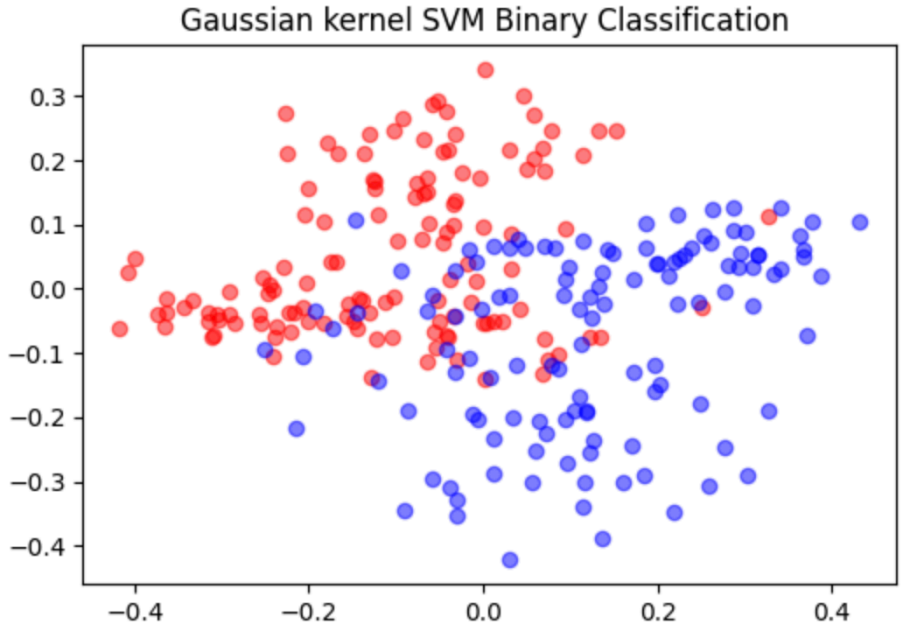
Lo scatter plot iniziale dei dati è il seguente.



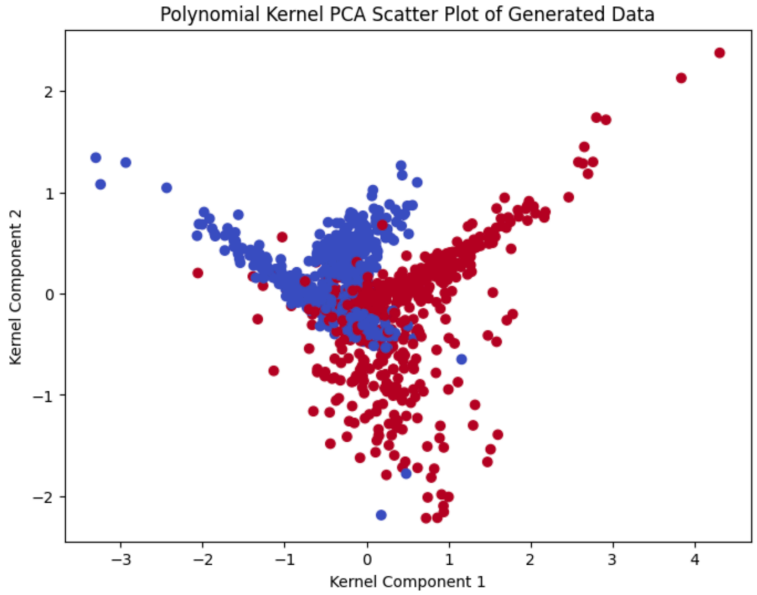
La prima KernelPCA che è stata applicata è quella con kernel gaussiano, il numero di componenti principali è stato impostato a due e il grafico ottenuto è il seguente.



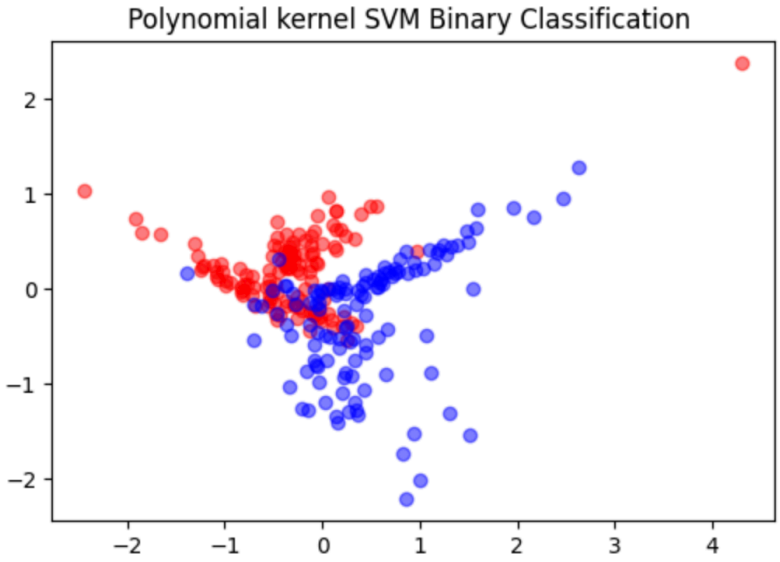
L'accuracy che si ottiene con SVM con kernel gaussiano è di 0.876 e vediamo anche dal grafico ottenuto che le classi sono per lo più separate, ma comunque sovrapposte.



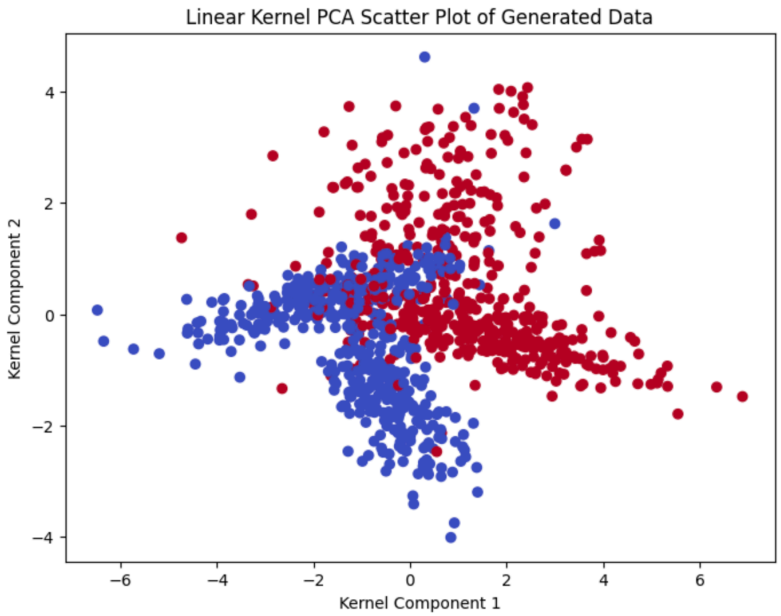
Il secondo modello studiato è quello con kernel polinomiale, il numero di componenti principali è anche in questo caso due e il grafico che si ottiene è il seguente.



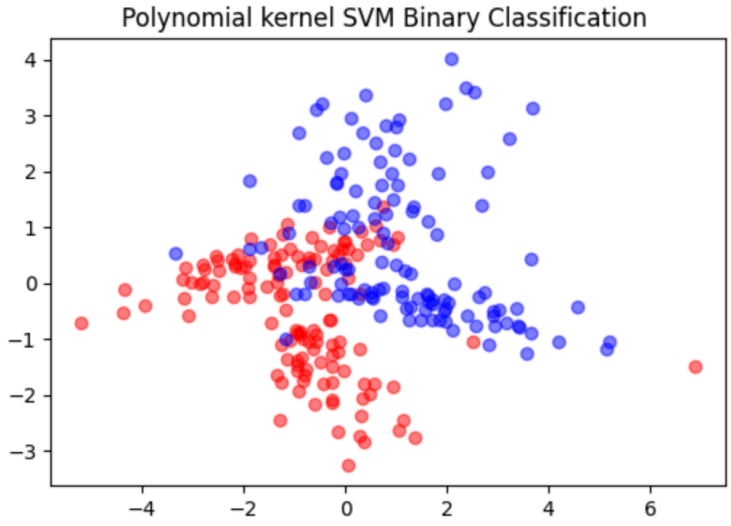
In questo caso, già da questo primo grafico si nota una performance leggermente inferiore rispetto a quella del kernel gaussiano, infatti l'accuracy trovata con SVM è di 0.808, anche dal grafico ottenuto si vede una separazione delle classi meno accurata, soprattutto al centro le due classi tendono a sovrapporsi.



L'ultima kernelizzazione utilizzata è quella con il kernel lineare, anche qui con numero di componenti principali impostato a due, il grafico ottenuto è il seguente.



Le due classi non sembrano separarsi in maniera netta nemmeno in questo caso, l'accuracy trovata applicando SVM è di 0.864 e il grafico ottenuto è il seguente.



Confrontando i risultati ottenuti con i diversi tipi di kernel sembra che il gaussiano risulti anche in questo caso il migliore, facendo però un confronto usando SVM con cross-validation il kernel lineare risulta migliore; la differenza tra le accuracy trovate è comunque minima, tutti i modelli performano bene con un'accuracy di almeno l' 80%.

Confrontando l'analisi di questo ultimo dataset con quella fatta nella parte due si nota che l'effetto della kernelizzazione sia molto più significativo sul dataset creato con make_circles rispetto che sull'ultimo preso in considerazione, questo è probabilmente dovuto al fatto che i dati ottenuti in questa ultima parte fossero già più facili da separare in modo lineare.