PATTERN RECOGNITION ALGORITHM

L’algoritmo che abbiamo deciso di portare è un semplice sistema di pattern recognition chiamato Support Vector Machine(SVM), scritto in Python tramite librerie scikit-learn e matplotlib.

Il SVM è un classificatore che cerca di trovare il miglior iperpiano di separazione tra diverse classi di dati o meglio è una tecnica di machine learning che può essere utilizzata per problemi di pattern recognition. Nell'ambito di questo algoritmo specifico, il pattern recognition si riferisce alla capacità del modello SVM di riconoscere e separare pattern o strutture nei dati.

Il SVM è particolarmente efficace nel risolvere problemi di classificazione, dove l'obiettivo è assegnare una classe a ciascun elemento di un insieme di dati. Nel codice fornito, il modello SVM è addestrato su un insieme di dati bidimensionale (usando solo due caratteristiche del dataset Iris) e viene visualizzato il decision boundary, che è essenzialmente la frontiera che separa le diverse classi.

Il pattern recognition in questo contesto si riferisce alla capacità dell'SVM di riconoscere pattern nei dati e di creare un modello che possa separare efficacemente le diverse classi. Questo è un esempio di come l'applicazione di algoritmi di machine learning come l'SVM possa essere utilizzata per problemi di riconoscimento di pattern nei dati.

Questo codice esegue le seguenti operazioni:

- Carica il dataset Iris, che contiene misurazioni di tre specie di iris (fiori) con quattro caratteristiche ciascuna.
- Seleziona solo le prime due caratteristiche del dataset per semplificare la visualizzazione.
- Suddivide il dataset in un set di addestramento (80%) e un set di test (20%).
- Crea un modello SVM (Support Vector Machine) con kernel lineare.
- Addestra il modello SVM sul set di addestramento.
- Effettua previsioni sul set di test.

- Valuta l'accuratezza del modello SVM confrontando le previsioni con le etichette reali nel set di test.
- Visualizza i punti del dataset con colori diversi per le diverse classi di iris.
- Visualizza il decision boundary del modello SVM sulla base delle previsioni fatte. Il decision boundary è la linea che separa le regioni dello spazio bidimensionale assegnate a diverse classi.

SVM cerca di trovare il decision boundary che massimizza la distanza tra i punti di diverse classi. Nel codice, la griglia (`xx` e `yy`) viene utilizzata per creare una mappa di colori che mostra come il modello classifica diverse regioni nello spazio bidimensionale.

Il codice utilizza la funzione `contour` di Matplotlib per visualizzare le regioni di decisione del modello SVM.

Queste regioni sono definite dai punti della griglia sulla quale vengono effettuate le previsioni tramite la funzione `decision_function`.

Le linee tratteggiate rappresentano il decision boundary e le regioni separate dalla linea continua corrispondono alle diverse classi assegnate dal modello SVM

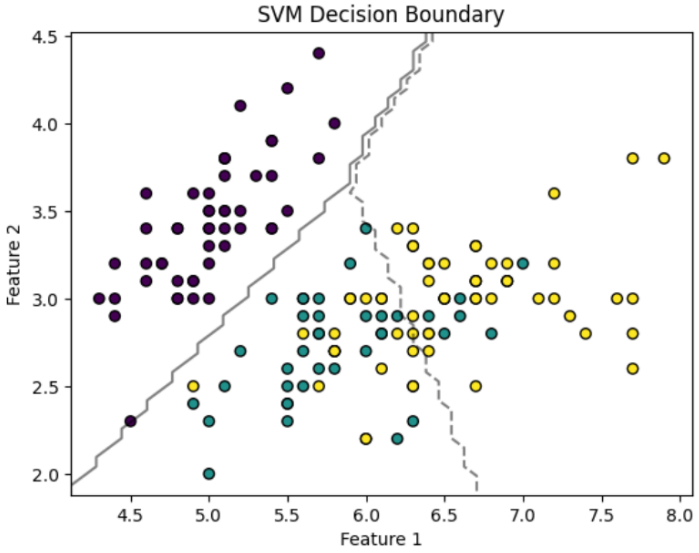
In questo modo, stai utilizzando la funzione `predict` per ottenere le previsioni di classe invece delle distanze dalla superficie di decisione. Questo dovrebbe darti una visualizzazione più chiara delle regioni di decisione del modello SVM.

```
!pip install scikit-learn
!pip install matplotlib
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Carica il dataset Iris
iris = datasets.load_iris()
X = iris.data[:, :2] # Prendi solo le prime due caratteristiche per la
visualizzazione
```

```
y = iris.target

# Suddividi il dataset in training set e test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Crea un modello SVM con kernel lineare
svm_model = SVC(kernel='linear')


# Addestra il modello SVM sul training set
svm_model.fit(X_train, y_train)

# Effettua le previsioni sul test set
y_pred = svm_model.predict(X_test)

# Valuta l'accuratezza del modello
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

# Visualizza i punti e il decision boundary del modello
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolors='k',
marker='o')
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# Creazione della griglia per valutare il modello
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50),
np.linspace(ylim[0], ylim[1], 50))

Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Visualizzazione delle regioni di decisione
ax.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
linestyles=['--', '-', '--'])

plt.title('SVM Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

```
Accuracy: 0.9
```


SVM Decision Boundary
Feature 2
Feature 1
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0