Introduzione

Questo documento fornisce una guida completa su come realizzare un classificatore basato su reti neurali utilizzando scikit-learn per distinguere tra due specie marziane: i "Simmy" e i "Robby". Questo progetto richiede competenze in:

- Manipolazione e pulizia dei dati (pandas)
- Visualizzazione dei dati (matplotlib, seaborn)
- Preprocessing e normalizzazione dei dati
- Implementazione di reti neurali (MLPClassifier)
- Valutazione delle prestazioni del modello
- Salvataggio e riutilizzo dei modelli addestrati

Contesto del Problema

Durante una missione su Marte, sono stati raccolti dati riguardanti due specie trovate nel Mare Erythraeum: i Simmy e i Robby. Gli scienziati hanno chiesto di creare un modello di riconoscimento basato su caratteristiche come:

- Peso
- Altezza
- Larghezza
- Colore
- Numero di arti

È cruciale distinguere tra queste due specie perché:

- I Simmy puliscono spontaneamente i pannelli solari dei rover dalla polvere marziana
- I Robby possono causare gravi danni ai macchinari se si sentono minacciati

L'algoritmo di riconoscimento verrà caricato sul computer dei rover per permettere al programma di guida autonoma di evitare i Robby e avvicinarsi ai Simmy.

1. Preparazione dell'Ambiente e Caricamento dei Dati

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from joblib import dump, load
```

```
# Caricamento del dataset
nomefile = 'marziani.csv'
data = pd.read_csv(nomefile)
print(data.head())
print(">>colonne: ", data.columns)
print(">>tipi\n", data.dtypes)
```

Il file CSV contiene informazioni sulle caratteristiche fisiche delle due specie marziane. Le colonne del dataset includono:

- `specie`: la classificazione (Simmy o Robby)
- `peso`: il peso dell'essere
- `altezza`: l'altezza dell'essere
- `larghezza`: la larghezza dell'essere
- `colore`: il colore dell'essere (blu, rosso, viola)
- `arti`: il numero di arti dell'essere

2. Analisi Esplorativa dei Dati

2.1 Comprensione del Dataset

```
# Verificare quante specie ci sono e quanti campioni
print(">>Specie")
print(data['specie'].unique())
print(">>Describe")
print(data['specie'].describe())
```

Questo codice mostra che ci sono due specie (Simmy e Robby) presenti in proporzioni uguali nel dataset.

2.2 Analisi Statistica per Specie

```
# Statistiche per ogni caratteristica, divise per specie
for specie in data.specie.unique():
    dati = data[data['specie'] == specie]
    print('>>', specie)
    for x in data.columns[1:]:
        print(dati[x].describe())
```

Dall'analisi statistica emerge che:

- I Simmy sono mediamente più pesanti e alti dei Robby
- I Robby sono leggermente più larghi
- Il colore più diffuso nei Robby è il blu, nei Simmy è il rosso

- Il numero di arti non sembra essere un fattore discriminante significativo

2.3 Visualizzazione dei Dati

```
# Conversione dei colori da categorici a numerici
colori = np.sort(data['colore'].dropna().unique())
print(colori)

d = data.copy()
mapping = {'blu': 0, 'rosso': 1, 'viola': 2}
d['colore'] = d['colore'].map(mapping)

# Visualizzazione della correlazione tra caratteristiche
sns.set_theme(font_scale=2)
sns.pairplot(d, diag_kind="hist", hue='specie', dropna=True)
sns.set()
```

Questa visualizzazione:

1. Converte i valori categorici di colore (blu, rosso, viola) in valori numerici (0, 1, 2)
2. Crea una matrice di grafici a dispersione che mostrano le relazioni tra tutte le coppie di caratteristiche
3. Differenzia i punti per specie usando colori diversi
4. Mostra istogrammi sulla diagonale per visualizzare la distribuzione di ciascuna caratteristica

3. Pulizia e Preparazione dei Dati

3.1 Gestione dei Valori Mancanti

```
# Identificare le colonne con dati mancanti
cols_with_missing = [col for col in d.columns if d[col].isnull().sum()]
print(cols_with_missing)

# Selezione delle caratteristiche più rilevanti
cols_selected = ['peso', 'altezza', 'larghezza']

# Rimuovere le righe con dati mancanti solo nelle colonne selezionate
d = data.dropna(subset=cols_selected)
print(d.shape)

# Controllo delle colonne rimanenti con dati mancanti
print([col for col in d.columns if d[col].isnull().sum()])
```

La pulizia dei dati:

1. Identifica quali colonne contengono valori mancanti
2. Seleziona solo le caratteristiche più informative per la classificazione (peso, altezza, larghezza)
3. Rimuove solo le righe che hanno valori mancanti in queste caratteristiche selezionate
4. Verifica che non ci siano altri problemi con i dati rimanenti

Dall'analisi grafica precedente, abbiamo notato che colore e numero di arti non sono altamente discriminanti tra le due specie, quindi ci concentriamo sulle tre caratteristiche fisiche principali.

3.2 Preparazione delle Feature e del Target

```
X = d[cols_selected]  # Dati di input (features)
y = d['specie']       # Output desiderato (target)
print(X.head())
print(y.head())
```

4. Divisione in Training e Test Set

```
# Divisione del dataset in training (70%) e test (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=0)

print(X_train.head())
print("Numero di campioni in X train: ", X_train.shape[0])
print(y_train.value_counts())
print(y_train.head())
```

La divisione del dataset è una fase cruciale:

- Il training set (70% dei dati) viene utilizzato per addestrare il modello
- Il test set (30% dei dati) viene utilizzato per valutare le prestazioni del modello su dati mai visti
- `random_state=0` assicura che la divisione sia riproducibile

5. Standardizzazione dei Dati

```
# Calcolo di media e deviazione standard dal training set
mean = X_train.mean()
std = X_train.std()

# Standardizzazione (z-score normalization)
X_train_std = ((X_train-mean)/std)
X_test_std = ((X_test-mean)/std)
```

```
print(f">>X train Normalizzato \n {X_train_std.describe()}")
```

La standardizzazione è fondamentale per le reti neurali perché:

1. Rende le caratteristiche confrontabili nonostante scale diverse
2. Aiuta l'algoritmo di addestramento a convergere più velocemente
3. Evita che una caratteristica con valori molto grandi domini sulle altre

Si nota che dopo la standardizzazione:

- La media di ogni caratteristica è 0
- La deviazione standard è 1
- La distribuzione dei dati viene preservata

6. Costruzione e Addestramento del Modello

```
# Creazione del modello MLP (Multi-Layer Perceptron)
model = MLPClassifier(hidden_layer_sizes=(100,100), random_state=1,
max_iter=300)

# Addestramento del modello
model.fit(X_train_std, y_train)
```

Il modello scelto è un Multi-Layer Perceptron (MLP), una rete neurale feed-forward con:

- 2 strati nascosti, ciascuno con 100 neuroni
- Un numero massimo di 300 iterazioni per l'addestramento
- Un valore fisso per il generatore di numeri casuali per garantire riproducibilità

7. Valutazione del Modello

```
# Test del modello
print("Train")
print(y_train.values[:5])
print(model.predict(X_train_std[:5]))

print("Test")
print(y_test.values[:5])
print(model.predict(X_test_std[:5]))

# Predizione su un nuovo caso
caso = [4.8, 31.4, 70.8]
caso_std = ((caso-mean)/std)
print("Caso")
print(model.predict([caso_std]))
```

```
# Calcolo dell'accuratezza
train_score = model.score(X_train_std, y_train)
print(f"Accuratezza dati di TRAIN: {round(train_score, 3)}")
test_score = model.score(X_test_std, y_test)
print(f"Accuratezza dati di TEST: {round(test_score, 3)}")
```

L'accuratezza del modello è eccellente (circa 98%), il che significa che:

- Il modello è in grado di classificare correttamente il 98% dei campioni
- Il modello generalizza bene sui dati di test (non ha overfitting)
- Il modello può essere utilizzato con fiducia per classificare nuove osservazioni

8. Analisi dell'Overfitting

```
# Monitorare l'accuratezza in funzione del numero di epoche
start = 100
stop = 300
passo = 10

vEpochs = np.arange(start, stop, passo)
vAccTrain = []
vAccTest = []

for e in vEpochs:
    mlp = MLPClassifier(hidden_layer_sizes=(100,100), random_state=1,
max_iter=e)
    mlp.fit(X_train_std, y_train)
    vAccTrain.append(mlp.score(X_train_std, y_train))
    vAccTest.append(mlp.score(X_test_std, y_test))

# Grafico dell'accuratezza
plt.plot(vEpochs, vAccTrain, c='b', label='Train')
plt.plot(vEpochs, vAccTest, c='g', label='Test')
plt.legend()
```

Questo grafico ci permette di:

1. Individuare il punto ottimale di arresto dell'addestramento
2. Rilevare eventuali segni di overfitting (quando l'accuratezza sul training continua a migliorare ma peggiora sul test)
3. Trovare il miglior compromesso tra bias e varianza

Dall'analisi emerge che intorno alle 115 epoche si verifica un leggero overfitting, dove l'accuratezza sui dati di training continua a migliorare ma inizia a peggiorare sui dati di test.

9. Ottimizzazione del Modello

```
# Creazione di un modello ottimizzato con il numero ottimale di epoche
model01 = MLPClassifier(hidden_layer_sizes=(100,100), random_state=1,
max_iter=115)
model01.fit(X_train_std, y_train)

train_score01 = model01.score(X_train_std, y_train)
test_score01 = model01.score(X_test_std, y_test)

print(f"Accuratezza dati di TRAIN: {round(train_score01, 3)}")
print(f"Accuratezza dati di TEST: {round(test_score01, 3)}")
print(model01.predict([caso_std]))
```

Limitando il numero di epoche a 115, otteniamo:

- Una leggera diminuzione dell'accuratezza sul training set
- Un miglioramento dell'accuratezza sul test set
- Un modello che generalizza meglio su nuovi dati

10. Salvataggio e Caricamento del Modello

```
from joblib import dump, load

# Salvataggio del modello
dump(model01, 'marziani.joblib')

# Caricamento del modello
modelImportato = load('marziani.joblib')
print(modelImportato.predict([caso_std]))
```

La serializzazione del modello consente di:

1. Salvare il modello addestrato per uso futuro
2. Evitare di ripetere il processo di addestramento
3. Distribuire il modello per essere utilizzato in altri sistemi (come il rover marziano)

Conclusione e Raccomandazioni

Il modello di rete neurale sviluppato è in grado di classificare con un'alta accuratezza le due specie marziane sulla base delle loro caratteristiche fisiche. Questo permetterà ai rover di identificare correttamente i Simmy (utili per la pulizia dei pannelli solari) e i Robby (potenzialmente dannosi).

Punti chiave del progetto:

1. Le caratteristiche più discriminanti sono peso, altezza e larghezza.
2. La standardizzazione dei dati è essenziale per il buon funzionamento della rete neurale.

3. L'architettura ottimale è un MLP con due strati nascosti da 100 neuroni ciascuno.
4. L'accuratezza del modello ottimizzato è superiore al 98%.
5. Il modello è stato salvato per essere caricato direttamente sul sistema di guida autonoma dei rover.

Possibili miglioramenti:

- Esplorare diverse architetture di rete (variare il numero di strati e neuroni)
- Implementare tecniche di regolarizzazione per contrastare l'overfitting
- Raccogliere più dati, specialmente negli intervalli di confine tra le due classi
- Aggiungere altre caratteristiche discriminanti se disponibili in futuro

Questo progetto dimostra l'efficacia dell'apprendimento automatico nella classificazione di specie aliene e può essere esteso ad altre applicazioni in futuri studi su Marte.

Appendice: Metriche Alternative per Problemi di Regressione

Nel caso di problemi di regressione (previsione di valori continui anziché classificazione), si utilizza solitamente il MAE (Mean Absolute Error) come metrica di valutazione:

```
from sklearn.metrics import mean_absolute_error

predictions = model.predict(X_test_std)
mae = mean_absolute_error(predictions, y_test)
print(mae)
```

Il MAE misura la media degli errori assoluti tra le previsioni e i valori reali, fornendo un'indicazione dell'errore medio di previsione in unità della variabile target.