Andrea Augello
Department of Engineering, University of Palermo, Italy

Alberi di decisione

Università degli Studi di Palermo

Introduzione

Introduzione

- Gli alberi di decisione sono un metodo di apprendimento supervisionato
- Sono utilizzati per la classificazione e la regressione
- Sono un modello di apprendimento interpretabile e facilmente visualizzabile
- Utilizzano semplici regole di decisione inferite dai dati di addestramento

Vantaggi e svantaggi

- Vantaggi
  - Semplice da capire e interpretare
  - Richiede poca preparazione dei dati
  - Può gestire sia dati numerici che categorici
  - Non necessita di assunzioni sulle distribuzioni dei dati
  - Può gestire problemi multi-classe
  - Richiede poche risorse per l'inferenza

Vantaggi e svantaggi

- Vantaggi
  - Semplice da capire e interpretare
  - Richiede poca preparazione dei dati
  - Può gestire sia dati numerici che categorici
  - Non necessita di assunzioni sulle distribuzioni dei dati
  - Può gestire problemi multi-classe
  - Richiede poche risorse per l'inferenza
- Svantaggi
  - Sono facilmente soggetti all'overfitting
  - Possono essere instabili, piccole variazioni nei dati possono portare a grandi variazioni nella struttura dell'albero
  - Necessitano di classi bilanciate
  - L'output non è continuo, ma a step

Come funziona?

Come funziona?

- All'arrivo di un vettore di input, l'albero di decisione esegue una serie di test per determinare la classe di appartenenza.
- Le decisioni sono tipicamente binarie, del tipo "*feature* $x_i \leq \alpha$*?*" quindi ogni nodo dell'albero ha due figli. (Esistono anche altri tipi di alberi)
- Divisioni successive dello spazio delle feature creano regioni corrispondenti alle classi. (Non tutte le feature vengono necessariamente utilizzate)

Algoritmo di addestramento

Funzione ADDESTRA ($dataset$, $features$)
Output: Albero di decisione

1: **if** STOPCONDITION **then**
2: **return** Nodo foglia con classe più appropriata
3: **end if**
4: $feature, soglia \leftarrow$ SCEGLISPLIT($dataset$, $features$)
5: **for** $e \in Dataset$ **do**
6: **if** $e[feature] \leq soglia$ **then**
7: $dataset_1 \leftarrow e$
8: **else**
9: $dataset_2 \leftarrow e$
10: **end if**
11: **end for**
12: $nodo \leftarrow$ NUOVONODO($feature$, $soglia$)
13: $nodo.figlio_1 \leftarrow$ ADDESTRA($dataset_1$, $features$)
14: $nodo.figlio_2 \leftarrow$ ADDESTRA($dataset_2$, $features$)
15: **return** $nodo$

Scelte implementative

Ad ogni step di divisione dell'albero ci sono delle scelte da fare:

- Come si sceglie la feature da utilizzare per lo split successivo?
- Come si sceglie il valore di soglia?
- Come si determina quando fermarsi?
- Come si determina la classe di un nodo foglia?

Esistono diverse strategie per rispondere a queste domande, che portano a diverse implementazioni di alberi di decisione.

Scelta di feature e soglie

Soglie

- Per ogni feature x_i, ogni possibile valore di soglia α può determinare uno split differente.
- Potenzialmente quindi ci sono infiniti split possibili tra cui scegliere.
- In pratica ci si limita ad i valori di soglia che corrispondono ai punti medi tra due valori consecutivi di x_i osservati nel dataset.
- I possibili punti di scelta ad ogni nodo sono quindi dati dal numero di feature moltiplicato per la cardinalità del dataset.

Come scegliere quale tra questi punti di scelta utilizzare?

Scelta di feature e soglie

Criterio di split

- Ogni divisione di un nodo genera due discendenti.
- È ragionevole voler scegliere la divisione che genera i discendenti più omogenei possibile rispetto alla divisione originale.

Scelta di feature e soglie

Criterio di split

- Ogni divisione di un nodo genera due discendenti.
- È ragionevole voler scegliere la divisione che genera i discendenti più omogenei possibile rispetto alla divisione originale.
- Due i criteri più utilizzati:

Indice di Gini

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

Entropia di Shannon

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i$$

- La diminuzione dell'impurità di un nodo è data dalla somma pesata delle impurità dei nodi figli.

$$\Delta I = I_{parent} - \frac{N_1}{N} I_1 - \frac{N_2}{N} I_2$$

Scelta di feature e soglie

Criterio di split

- Ogni divisione di un nodo genera due discendenti.
- È ragionevole voler scegliere la divisione che genera i discendenti più omogenei possibile rispetto alla divisione originale.
- Due i criteri più utilizzati:

Indice di Gini

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

Entropia di Shannon

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i$$

- La diminuzione dell'impurità di un nodo è data dalla somma pesata delle impurità dei nodi figli.

$$\Delta I = I_{parent} - \frac{N_1}{N} I_1 - \frac{N_2}{N} I_2$$

- Tra i possibili punti di scelta si sceglie quello che massimizza la diminuzione dell'impurità.

Quando fermarsi?

- Threshold di impurità
- Numero minimo di esempi in un nodo
- Profondità massima dell'albero

Cosa fare se un nodo foglia non è puro?

- Assegnare la classe più frequente

Un esempio manuale

Un esempio manuale

Dataset di esempio

- Due feature x_1 e x_2
- Tre classi
- 10 esempi

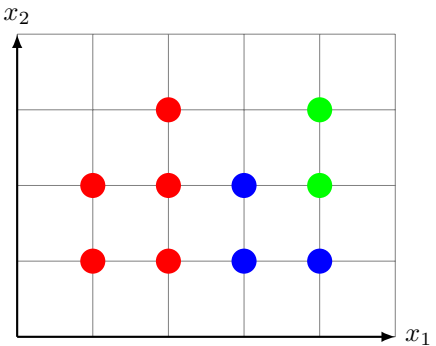
x_1	x_2	y
1	1	1
1	2	1
2	1	1
2	2	1
2	3	1
3	1	2
3	2	2
4	1	2
4	2	3
4	3	3

Un esempio manuale

Dataset di esempio

- Due feature x_1 e x_2
- Tre classi
- 10 esempi

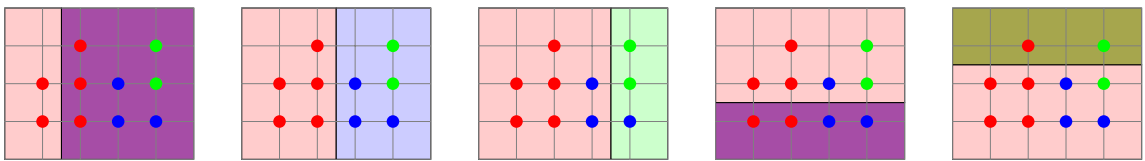
x_1	x_2	y
1	1	1
1	2	1
2	1	1
2	2	1
2	3	1
3	1	2
3	2	2
4	1	2
4	2	3
4	3	3



Entropia iniziale: 1.485

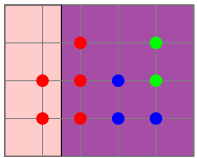
Un esempio manuale

Possibili split:

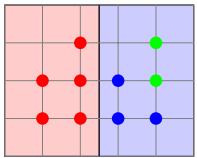


Un esempio manuale

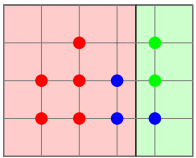
Possibili split:



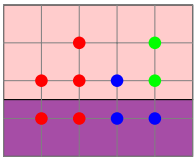
$\Delta I = 0.235$



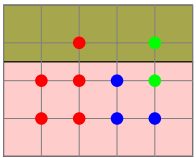
$\Delta I = 0.337$



$\Delta I = 0.605$



$\Delta I = 0.160$



$\Delta I = 0.105$

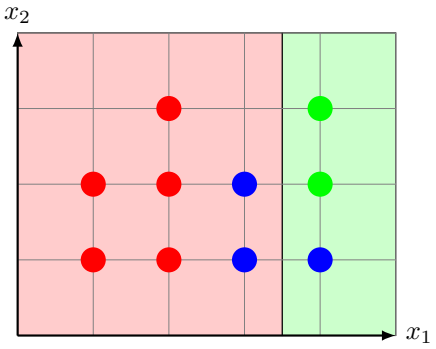
Un esempio manuale

Primo split:

- Scegliamo la feature x_1 (per fare meno conti)
- Scegliamo il valore di soglia $\alpha = 1.5$

Effetto:

- Entropia iniziale: $H = -\frac{5}{10}\log\frac{5}{10} - \frac{3}{10}\log\frac{3}{10} - \frac{2}{10}\log\frac{2}{10} = 0.485$
- Entropia nodo sinistro: $H_1 = -\frac{5}{7}\log\frac{5}{7} - \frac{2}{7}\log\frac{2}{7} = 0.863$
- Entropia nodo destro: $H_2 = -\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3} = 0.918$
- $\Delta I = 1.485 - \frac{7}{10}\cdot 0.863 - \frac{3}{10}\cdot 0.918 = 0.605$



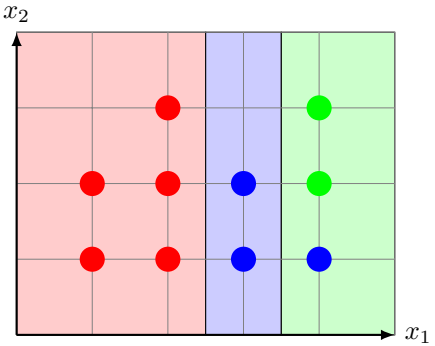
Un esempio manuale

Secondo split:

- Scegliamo la feature x_1
- Scegliamo il valore di soglia $\alpha = 1.5$

Effetto:

- Entropia iniziale: $H = 0.863$
- Entropia nodo sinistro: $H_1 = 0$
- Entropia nodo destro: $H_2 = 0$
- $\Delta I = 0.863$



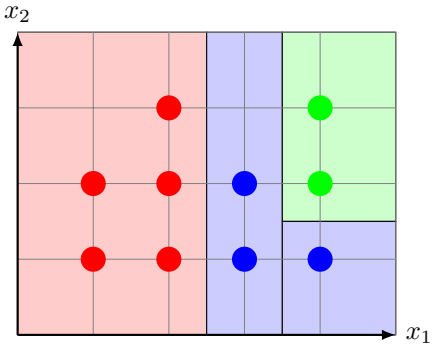
Un esempio manuale

Ultimo split:

- Scegliamo la feature x_1
- Scegliamo il valore di soglia $\alpha = 3.5$

Effetto:

- Entropia iniziale: $H = 0.918$
- Entropia nodo sinistro: $H_1 = 0$
- Entropia nodo destro: $H_2 = 0$
- $\Delta H = 0.918 - \frac{1}{3} \cdot 0 - \frac{2}{3} \cdot 0 = 0.918$

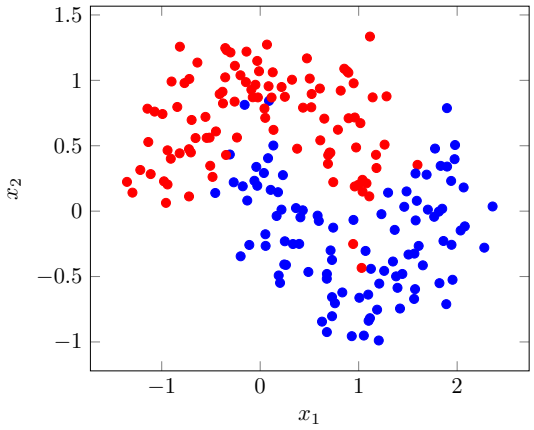


Implementazione

Creazione del dataset

```
from sklearn.datasets import
    make_moons

X, y = make_moons(n_samples=200,
    noise=0.25, random_state=3)

for i in range(0,200):
    print(*X[i], y[i])
```

```
#utils.py
def load_data(file):
    data = np.loadtxt(file)
    x = data[:, :-1]
    y = data[:, -1]
    return x, y
```



Implementazione

```
# Import delle librerie
from sklearn import tree
import utils
import matplotlib.pyplot as plt
X,y = utils.load_data("dataset.dat") # Caricamento del dataset
clf = tree.DecisionTreeClassifier(  # Creazione del classificatore
        max_depth=3,                # Profondità massima dell'albero
        )
clf = clf.fit(X, y)                 # Addestramento del classificatore
clf.predict(X)                      # Predizione
tree.plot_tree(clf)                 # Visualizzazione dell'albero
plt.show()
print(tree.export_text(clf))        # Visualizzazione testuale
utils.plot(X,y,clf)                 # Visualizzazione spazio feature
```

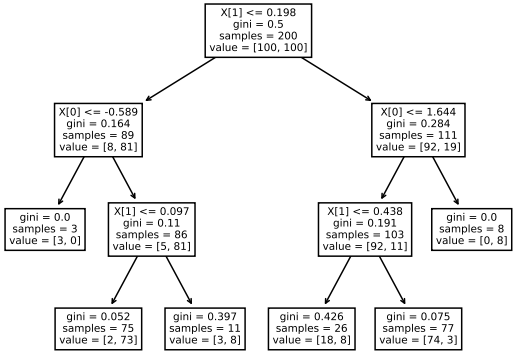
Opzioni principali per DecisionTreeClassifier

- `criterion`: criterio di divisione dei nodi
  - `gini`: indice di Gini
  - `entropy`: entropia
- `splitter`: strategia di divisione dei nodi
  - `best`: sceglie la miglior divisione
  - `random`: sceglie una divisione casuale
- `max_depth`: profondità massima dell'albero

Implementazione

```
|--- feature_1 <= 0.20
|   |--- feature_0 <= -0.59
|   |   |--- class: 0.0
|   |--- feature_0 >  -0.59
|   |   |--- feature_1 <= 0.10
|   |   |   |--- class: 1.0
|   |   |--- feature_1 >  0.10
|   |   |   |--- class: 1.0
|--- feature_1 >  0.20
|   |--- feature_0 <= 1.64
|   |   |--- feature_1 <= 0.44
|   |   |   |--- class: 0.0
|   |   |--- feature_1 >  0.44
|   |   |   |--- class: 0.0
|   |--- feature_0 >  1.64
|   |   |--- class: 1.0
```

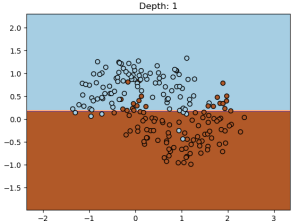
Sistema di decisione multilivello

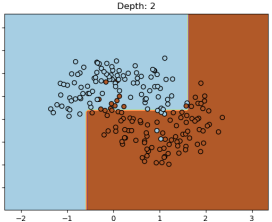


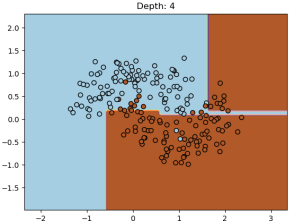
Visualizzare lo spazio delle feature

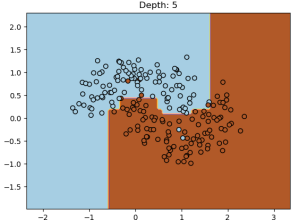
```
def plot(X,y, model, title=""):
    # define bounds of the domain
    min1, max1 = X[:, 0].min()-1, X[:, 0].max()+1
    min2, max2 = X[:, 1].min()-1, X[:, 1].max()+1
    # define the x and y scale
    x1grid = np.arange(min1, max1, 0.05)
    x2grid = np.arange(min2, max2, 0.05)
    # create all of the lines and rows of the grid
    xx, yy = np.meshgrid(x1grid, x2grid)
    # flatten each grid to a vector
    r1, r2 = xx.flatten(), yy.flatten()
    # horizontal stack vectors to create x1,x2 input for the model
    grid = [[x1, x2] for x1, x2 in zip(r1,r2)]
    # make predictions for the grid
    yhat = model.predict(grid)
    # reshape the predictions back into a grid
    zz = yhat.reshape(xx.shape)
    # plot the grid of x, y and z values as a surface
    plt.contourf(xx, yy, zz, cmap='Paired')
    # create scatter plot for samples from each class
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='Paired', edgecolors='black')
    plt.title(title)
    # show the plot
    plt.pause(3)
    plt.clf()
```

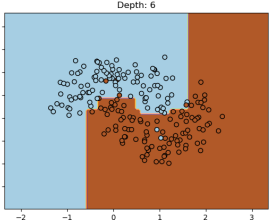
Visualizzare lo spazio delle feature

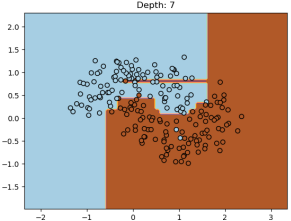












Valutare la bontà della classificazione

- Accuratezza: $\frac{TP+TN}{TP+TN+FP+FN}$
- Precisione: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$
- F1: $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$

Valutare la bontà della classificazione

- Accuratezza: $\frac{TP+TN}{TP+TN+FP+FN}$
- Precisione: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$
- F1: $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$

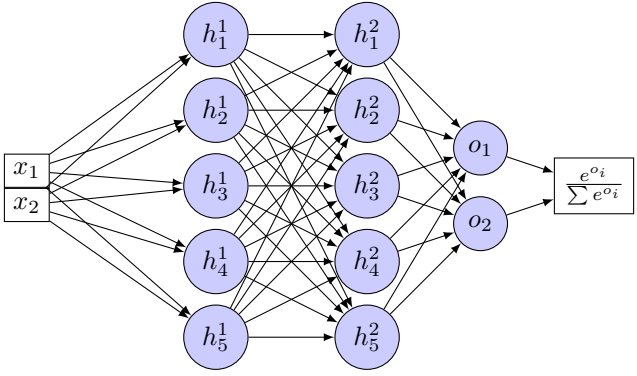
```
def classification_stats(y, yhat):
    accuracy = (yhat == y).sum() / len(y)
    precision = ((yhat == 1) * (y == 1)).sum() / (yhat == 1).sum()
    recall = ((yhat == 1) * (y == 1)).sum() / (y == 1).sum()
    f1 = 2 * precision * recall / (precision + recall)
    print(f"Accuracy: {accuracy} Precision: {precision} Recall:
        {recall} F1: {f1}")
```

Approcci alternativi

Approcci alternativi

Un problema di intelligenza artificiale può essere affrontato in molti modi diversi.

Proveremo ad affrontare lo stesso problema di classificazione utilizzando una rete neurale.



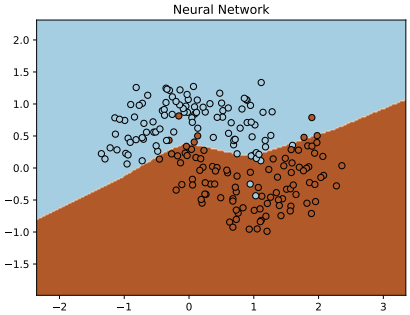
Reti neurali

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(2, 50)
        self.linear2 = nn.Linear(50, 50)
        self.linear3 = nn.Linear(50, 2)
        self.activation = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.activation(self.linear(x))
        x = self.activation(self.linear2(x))
        x = self.linear3(x)
        x = self.softmax(x)
        return x
```

Reti neurali

```
    def predict(self, x):
        x = torch.tensor(x, dtype=torch.float)
        with torch.no_grad():
            return self.forward(x).argmax(dim=1).numpy()

    def train_loop(self, X, y, epochs=900, lr=0.1):
        optimizer = torch.optim.SGD(self.parameters(), lr=lr)
        loss_fn = nn.CrossEntropyLoss()
        for epoch in range(epochs):
            y_pred = self.forward(X)
            loss = loss_fn(y_pred, y)
            if epoch % 100 == 0:
                print(f"Epoch: {epoch}, Loss: {loss.item()}")
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

Risultato della rete neurale



- Accuratezza: $\frac{TP+TN}{TP+TN+FP+FN} = 0.95$
- Precisione: $\frac{TP}{TP+FP} = 0.96$
- Recall: $\frac{TP}{TP+FN} = 0.94$
- F1: $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall} = 0.95$