Documentazione del Progetto AI

Questo documento fornisce una descrizione dettagliata del codice sorgente per i file `models.py` e `decision_tree.py`.

---

1. Documentazione: `models.py`

Descrizione Generale

Il file `models.py` utilizza la libreria **scikit-learn** per addestrare, valutare e confrontare tre diversi algoritmi di classificazione: **Decision Tree**, **k-Nearest Neighbors (k-NN)** e **Random Forest**. Lo script supporta due dataset: *Mushroom* e *Rice*. Include funzionalità per la visualizzazione dei risultati e l'analisi dell'importanza delle feature.

Librerie Utilizzate

- **Manipolazione dei dati**: `pandas`, `numpy`, `scipy.io.arff`
- **Addestramento e test dei modelli**: `sklearn` (model_selection, preprocessing, tree, neighbors, ensemble, decomposition, inspection, metrics)
- **Visualizzazione**: `matplotlib`, `seaborn`

Funzioni Principali

Gestione e Preprocessing dei Dati

- `preprocces_data_mushrooms(input_data)`
  - **Scopo**: Preprocessing sul dataset Mushroom.
  - **Operazioni**:
    - Codifica la variabile target (`class`) in 0/1 tramite `LabelEncoder`.
    - Rimuove la colonna `veil-type` (inutile in quanto ha un solo valore).
    - Gestisce i valori mancanti in `stalk-root` sostituendo '?' con 'missing'.
    - Applica **One-Hot Encoding** (`pd.get_dummies`) alle feature categoriche.
    - Applica **StandardScaler** per normalizzare le feature.
  - **Output**: Dati normalizzati (`x_scaled`), il vettore target target (`y`), la lista dei nomi delle feature e delle classi.
- `preprocces_data_rice(input_data)`
  - **Scopo**: Preprocessing sul dataset Rice .
  - **Operazioni**:

    - Decodifica stringhe UTF-8 se necessario (comune nei file ARFF).
    - Codifica la variabile target con `LabelEncoder`.
    - Normalizza le feature numeriche con **StandardScaler**.
  - **Output**: Dati normalizzati (`x_scaled`), il vettore target target (`y`), la lista dei nomi delle feature e delle classi.

Addestramento Modelli

Tutti i modelli vengono istanziati con iperparametri specifici:

- `train_tree_model(x_train, y_train)` : Addestra un `DecisionTreeClassifier` con `max_depth=5` per prevenire l'overfitting.
- `train_knn_model(x_train, y_train)` : Addestra un `KNeighborsClassifier` con `n_neighbors=5` .
- `train_rf_model(x_train, y_train)` : Addestra un `RandomForestClassifier` con `n_estimators=10` alberi.

Valutazione e Metriche

- `model_accuracy(model, x_test, y_test)` : Calcola l'accuratezza.
- `cross_validation_score(model, x_data, y_data, cv=10)` : Esegue una **10-fold Cross-Validation** per ottenere una stima più precisa delle prestazioni (restituisce media e deviazione standard).
- `calculate_and_print_metrics(model, x_test, y_test, title)` : Calcola e stampa a video **Precision** e **Recall**.
- `save_confusion_matrix(...)` :
  - Genera una heatmap della matrice di confusione usando `seaborn` .
  - Per classificazione binaria, assegna alle celle delle etichette: "True Neg", "False Pos", "False Neg", "True Pos" includendo conteggi e percentuali.

Visualizzazione

- `visualize_decision_tree(...)` : Genera un'immagine (`.png`) della struttura dell'albero decisionale addestrato, mostrando i nodi e le regole di split.
- `visualize_knn_boundaries(...)` :
  - Riduce la dimensionalità dei dati a 2 componenti usando **PCA**.
  - Addestra un k-NN temporaneo sui dati 2D per visualizzare le regioni decisionali (confini tra classi) su un piano cartesiano.
- `plot_feature_importance(...)` :
  - Per modelli ad albero (Decision Tree, Random Forest): usa l'attributo `feature_importances_` .

- Per k-NN, che non contiene esplicitamente la feature_importance: calcola la `Permutation Importance` valutando la dimunzione delle prestazioni quando una feature viene considerata.
- Salva un grafico a barre con le feature più influenti.

---

2. Documentazione: `decision_tree.py`

Descrizione Generale

Il file `decision_tree.py` contiene un'implementazione **da zero** di un Albero Decisionale. A differenza di `models.py`, qui non viene usato l'algoritmo di scikit-learn per costruire l'albero, ma viene implementata ricorsivamente la costruzione dei nodi, il calcolo dell'impurità e la predizione.

Struttura del Codice

Preprocessing

Le funzioni `load_and_preprocess_data_mushrooms` e `load_and_preprocess_data_rice` replicano la logica di preparazione dati vista in `models.py`.

Classe `DecisionNode`

Rappresenta un singolo nodo dell'albero. Contiene:

- `feature`: L'indice della feature usata per lo split (se nodo decisionale).
- `threshold`: Il valore di soglia per lo split.
- `left` / `right`: Riferimenti ai nodi figli.
- `value`: Il valore della classe predetta (se nodo foglia).

Algoritmo di Costruzione (`build_tree`)

Questa funzione ricorsiva costruisce l'albero addestrandolo sui dati:

1. **Condizioni di arresto**:
   - Se non ci sono più campioni.
   - Se l'impurità è 0 (tutti i campioni appartengono alla stessa classe).
2. **Ricerca del Miglior Split**:
   - Itera su tutte le feature (`num_features`) e su tutti i valori unici (`thresholds`) presenti nei dati.
   - Per ogni combinazione, divide i dati e calcola il guadagno di informazione.

3. **Criterio di Impurità (`gini_impurity`)**:
    - Implementa l'indice di Gini: $G = 1 - \sum(p_i)^2$, dove p_i è la probabilità che un elemento appartenga ad una certa classe una votla arrivato in quel nodo.
    - Sceglie lo split che minimizza la Gini Impurity.
4. **Ricorsione**: Crea i sottoalberi sinistro e destro chiamandosi ricorsivamente sui sottoinsiemi di dati divisi.

Predizione (`decision_tree_predict`)

Funzione per classificare un campione:

- Attraversa l'albero a partire dalla radice.
- Ad ogni nodo, confronta il valore della feature del campione con la `threshold` del nodo.
- Scende a sinistra o destra fino a raggiungere un nodo foglia (`value` non nullo), che restituisce la classe predetta.

Valutazione

- `cross_validation_score(X, y, cv=10)` : Implementa manualmente il ciclo di cross-validation.
    - Usa `StratifiedKFold` di sklearn per dividere i dati preservando le proporzioni delle classi.
    - Per ogni fold, ricostruisce l'albero da zero e calcola l'accuratezza.
- `evaluate_model(...)` : Calcola Accuracy, Precision e Recall confrontando le predizioni dell'albero custom con i valori reali.