AI and Machine Learning, Homework7

Author: Ying Yiwen
Number: 12210159

Contents

1 Introduction

We need to compute entropy, conditional entropy and information gain to build decision tree.

Entropy is an important measure of the uncertainty or purity of a data set. The higher the entropy value, the more scattered the samples in the dataset and the less pure the categories. The lower the entropy, the more concentrated the samples in the dataset and the purer the category.

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x) \tag{1}$$

```
count_y = np.bincount(y) # Count the number of each output label
prob_y = count_y[np.nonzero(count_y)] / y.size # Compute the probability of each output label
entropy_y = -np.sum(prob_y * np.log2(prob_y)) # Compute the entropy of output
```

$$H(Y|X) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(y|x) \tag{2}$$

```
for v in feature_values:
  y_sub = y[feature == v]
  prob_y_sub = y_sub.size / y.size
  h += prob_y_sub * self._entropy(y_sub)
```

In the construction of decision trees, we use **information gain** to select the best partition features, that is, to select the features with the maximum information gain to split the data set. We want to choose a feature that minimizes the uncertainty of the dataset and makes the distribution of categories in the dataset consistent.

$$IG(Y|X) = H(Y) - H(Y|X) \tag{3}$$

```
ig_feature = self._entropy(y) - self._conditional_entropy(feature, y)
```

Then we need to choose the best attribute, with the highest information gain.

```python
if features_list:
  gains = np.apply_along_axis(self._information_gain, 0, X[:, features_list], y)
  index = np.argmax(gains)
  if gains[index] > self.gain_threshold:
    return index
```

The construction of the decision tree usually uses a greedy algorithm, starting from the root node, gradually constructing each subnode according to the information gain of the features, and recursively dividing the dataset into purer and purer subsets by recursively dividing the dataset until the stopping conditions are met (e.g., the maximum depth is reached, the number of samples is less than a certain threshold, all features have been used, the information gain is below the threshold, and the class purity reaches the expectation). Sometimes, after building a complete tree, we prune to avoid overfitting and reduce the complexity of the tree to improve the generalization ability of the model.

```python
# Divide the training set according to this selected feature
# Then use the subset of training examples in each branch to create a sub-tree
feature_values = np.unique(X[:, node.feature_index])
for v in feature_values:
  # Obtain the subset of training examples
  idx = X[:, node.feature_index] == v
  X_sub, y_sub = X[idx], y[idx]
  # Build a sub-tree
  node.children[v] = self._build_tree(X_sub, y_sub, features_list.copy())
```

The training of decision tree is to build the tree:

```python
self.tree_ = self._build_tree(X_train, y_train, list(range(n)))
```

And the predicition of it is to recursively find the value:

```python
node = self.tree_
while node.children:
  child = node.children.get(x[node.feature_index])
  if not child:
    break
  node = child
return node.value
```

Thus, the main logic of this homework is:

```python
# Load the dataset
data = np.loadtxt('../dataset/lenses/lenses.data', dtype=int)

# Features and labels
X = data[:, 1:-1]
y = data[:, -1]

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)

# Create and train the decision tree
dt01 = DT.DecisionTree()
dt01.train(X_train, y_train)

# Print the tree
```

```python
print(dt01.tree_)

# Use the trained tree to make predictions on the test set
y_pred = dt01.predict(X_test)

# Calculate accuracy
accuracy = np.sum(y_pred == y_test) / len(y_test)
print(f"Accuracy: {accuracy}")

# Optionally, you can visualize the decision tree
features_dict = {
  0: {'name': 'age', 'value_names': {1:'young', 2:'pre-presbyopic', 3:'presbyopic'}},
  1: {'name':'prescript', 'value_names': {1: 'myope', 2: 'hypermetrope'}},
  2: {'name': 'astigmatic', 'value_names': {1: 'no', 2: 'yes'}},
  3: {'name': 'tear rate', 'value_names': {1:'reduced', 2:'normal'}},
}

label_dict = {
  1: 'hard',
  2: 'soft',
  3: 'no_lenses',
}

dtp = DecisionTreePlotter(dt01.tree_, feature_names=features_dict, label_names=label_dict)
dtp.plot()
```

2 Result

By splitting the dataset into training set and testing set (9:1), we can get the performance of the model.
Accuracy: 1.0

The tree is like:
Internal node <3>:

- 1 -> Leaf node (3)
- 2 -> **Internal node <2>:**
  - 1 -> **Internal node <0>:**
    - 1 -> Leaf node (2)
    - 2 -> Leaf node (2)
    - 3 -> **Internal node <1>:**
      - 1 -> Leaf node (3)
      - 2 -> Leaf node (2)
  - 2 -> **Internal node <1>:**
    - 1 -> Leaf node (1)
    - 2 -> **Internal node <0>:**
      - 1 -> Leaf node (1)
      - 2 -> Leaf node (3)
      - 3 -> Leaf node (3)

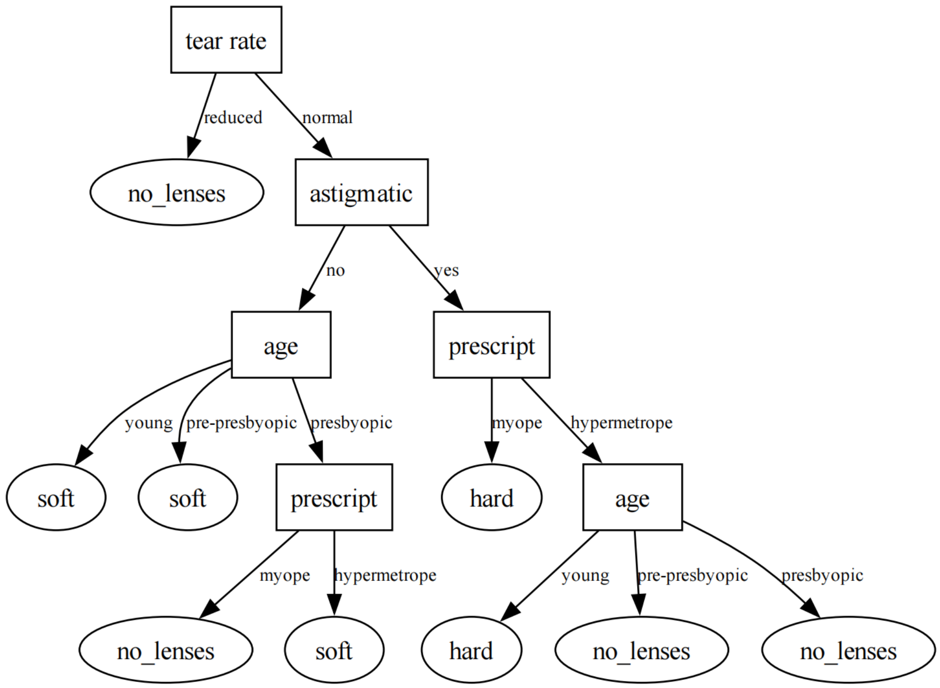

Fig. 1: Tree Structure