2020 Pattern Recognition and Machine Learning Technical Report

Ruian He, 16307110216,

I. TASK DESCRIPTION

USE the decision tree for classification based on Breast cancer dataset available at kaggle[1].

II. DATA DESCRIPTION

Breast Cancer Wisconsin (Diagnostic) Data Set is used to predict whether the cancer is benign or malignant.Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. [2]

A. Attribute Information

1) ID number
2) Diagnosis (M = malignant, B = benign)
3) Ten real-valued features are computed for each cell nucleus
    a) radius (mean of distances from center to points on the perimeter)
    b) texture (standard deviation of gray-scale values)
    c) perimeter
    d) area
    e) smoothness (local variation in radius lengths)
    f) compactness (perimeter ^{2}/area – 1.0)
    g) concavity (severity of concave portions of the contour)
    h) concave points (number of concave portions of the contour)
    i) symmetry
    j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

All feature values are recoded with four significant digits.

B. Missing attribute values

none

C. Class distribution

357 benign, 212 malignant

[1]https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

III. DATA PREPROCESSING

First we use pandas module to read from the csv file. As the diagnosis is a string (M/B), we must binarize the value for later calculation.Then we split the dataset into training and testing set using sklearn module. Finally we get 455 examples of training set with 30 features and 1 label, and 114 examples of test set.

IV. ALGORITHM INTRODUCTION

A. Algorithm principle

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. We take the ID3(Iterative Dichotomiser 3) algorithm with information gain as a criterion for choosing features. [3]

Information gain is the reduction in entropy or surprise by transforming a dataset and is often used in training decision trees. Information gain is calculated by comparing the entropy of the dataset before and after a transformation. We assume D is the set of samples and a is the feature with V possible values.

$$\text{Gain}(D,a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|}\text{Ent}(D^v)$$

And entropy, as it relates to machine learning, is a measure of the randomness in the information

being processed and is defined as follows. In the following formula, p_k is the proportion of class k samples.

$$\text{Ent}(D) = -\sum_{k=1}^{y} p_k \log_2 p_k$$

The larger the information gain, the better the classifier. So we may take the the original set S as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set S and calculates the entropy $H(S)$ or the information gain $IG(S)$ of that attribute. It then selects the attribute which has the smallest entropy (or largest information gain) value. The set S is then split or partitioned by the selected attribute to produce subsets of the data. The algorithm continues to recurse on each subset, considering only attributes never selected before. [1]

B. Algorithm implementation

The following code is from a python class `DecisionTree` and the code is split into different section in order to show the function of different parts.

First according to the definition, we can simply implement entropy and information gain calculation.

```
def _entropy(self, y):
  c = np.bincount(y)
  p = c[np.nonzero(c)] / y.size
  return -np.sum(p * np.log2(p))

def _conditional_entropy(self, feature, y
    ):
  feature_values = np.unique(feature)
  h = 0
  for value in feature_values:
    y_sub = y[feature == value]
    p = y_sub.size / y.size
    h += p * self._entropy(y_sub)
  return h

def _information_gain(self, feature, y):
  return self._entropy(y) - \
    self._conditional_entropy(feature,y)
```

Secondly, we define the data structure of tree nodes and we need to store which samples and which features the node have. Then we use "divide and conquer" algorithm to split the samples into two totally different set, and assign them to the child of the current node. The informational gain, feature and threshold is used for the division.

```
class Node(object):
  def __init__(self,sample_index,
      feature_index):
    self.sample_index = sample_index
    self.feature_index = feature_index

    self.childs = []
    self.predict = None

    self.gain = 0
    self.feature = None
    self.threshold = None
```

When we building decision tree, we select feature on each node and select threshold on each feaature. The best threshold and the best feature must have the hightest information gain. Then we can split the samples using the feature and threshold, that is we take the bigger part as one child and the smaller part as another.

```
def _select_threshold(self, X, y, feature
    ):
  feature_values = np.unique(X[feature])

  best_gain = 0
  best_threshold = None
  for value in feature_values:
    pred_feature = X[feature]>value
    gain = self._information_gain(
        pred_feature,y)
    if gain > best_gain:
      best_gain = gain
      best_threshold = value

  return best_threshold, best_gain

def _select_feature(self, node):
  X = self.X.loc[node.sample_index,node.
      feature_index]
  y = self.y.loc[node.sample_index]

  for feature in node.feature_index:
    threshold,gain = self.
        _select_threshold(X, y, feature)
    if gain > self.gain_threshold \
       and gain > node.gain:
      node.gain = gain
      node.feature = feature
      node.threshold = threshold

  return node.feature,node.threshold
```

The termination condition of the division is that the information gain is below some threshold, for example the maximum gain for some node is below 0.01 and we stop spliting and take the majority label as prediction.

```
def _build_tree(self,node):
  X = self.X.loc[node.sample_index,node.
      feature_index]
  y = self.y.loc[node.sample_index]

  feature,threshold = self.
      _select_feature(node)

  if feature is not None:
    feature_index = node.feature_index[:]
    feature_index.remove(feature)
    node.childs.append(self._build_tree(
        self.Node(X.index[X[feature] >
        threshold], feature_index)))
    node.childs.append(self._build_tree(
        self.Node(X.index[X[feature] <=
        threshold],feature_index)))
  else:
    node.predict = np.argmax(np.bincount(
        y))

  return node
```

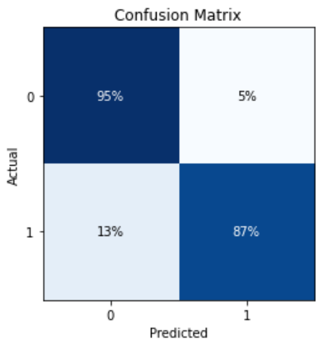
Finally, we can use the above functions to build decision tree, and predict test set by going down from root node.

```
def train(self, X, y):
  self.X = X
  self.y = y
  self.root = self.Node(list(X.index),
     list(X.columns))
  self._build_tree(self.root)

  print("build␣complete!")

def _predict_node(self, x, node):
  if node.feature is None:
    return node.predict
  elif x[node.feature] > node.threshold:
    return self._predict_node(x, node.
       childs[0])
  else:
    return self._predict_node(x, node.
       childs[1])

def predict(self, X):
  return [self._predict_node(X.loc[index
     ],self.root) for index in X.index]
```

V. Experimental Results and Analysis

We get 91.22% accuracy on our test set when the gain threshold is 0.1, and the f1 score is 0.9. The confusion matrix is as follows. But we get lower accuracy when we use lower threshold, which may result from overfitting, though it may have more nodes and edges.

Fig. 1. Confusion Matrix



At last, we can visualize our decision tree with `graphviz` module which can generate digraph using few commands.The tree structure figure is in the appendix.

We can learn from the figure that actually we divided most of the data in first two layers. So the deeper nodes make little contribution to the classification, we can cut them using another threshold.

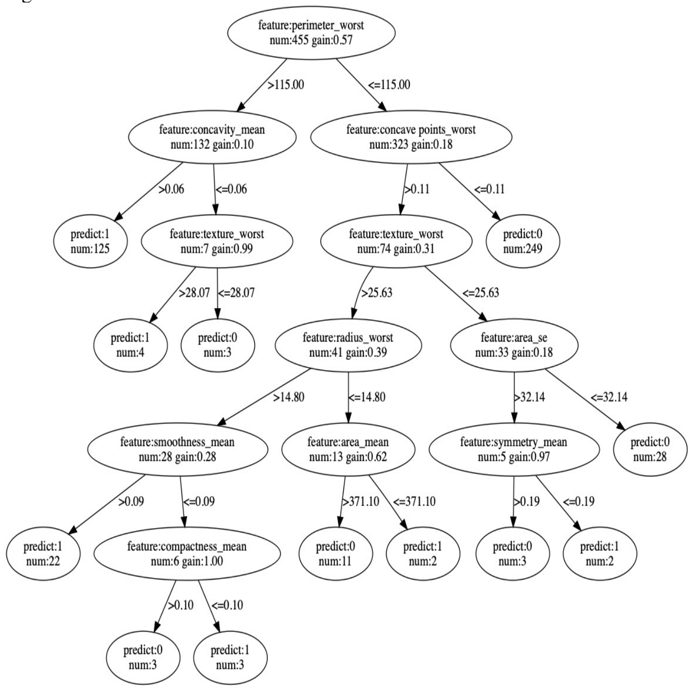
References

[1] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[2] W Nick Street, William H Wolberg, and Olvi L Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *Biomedical image processing and biomedical visualization*, volume 1905, pages 861–870. International Society for Optics and Photonics, 1993.

[3] Z.H. Zhou. *Machine Learning*. Tsinghua University Press, 2016.

Fig. 2. Decision Tree Structure


feature:perimeter_worst num:455 gain:0.57
>115.00
<=115.00
feature:concavity_mean num:132 gain:0.10
feature:concave points_worst num:323 gain:0.18
>0.06
<=0.06
>0.11
<=0.11
predict:1 num:125
feature:texture_worst num:7 gain:0.99
feature:texture_worst num:74 gain:0.31
predict:0 num:249
>28.07
<=28.07
>25.63
<=25.63
predict:1 num:4
predict:0 num:3
feature:radius_worst num:41 gain:0.39
feature:area_se num:33 gain:0.18
>14.80
<=14.80
>32.14
<=32.14
feature:smoothness_mean num:28 gain:0.28
feature:area_mean num:13 gain:0.62
feature:symmetry_mean num:5 gain:0.97
predict:0 num:28
>0.09
<=0.09
>371.10
<=371.10
>0.19
<=0.19
predict:1 num:22
feature:compactness_mean num:6 gain:1.00
predict:0 num:11
predict:1 num:2
predict:0 num:3
predict:1 num:2
>0.10
<=0.10
predict:0 num:3
predict:1 num:3