Project 1: Random Forests

April 10, 2024

Theory discussed

Report on overleaf (updated last with table): Overleaf Link

Overview

`(taken verbatum from our report)`

Random forests are a collection of small decision trees. They are built using small, randomly selected samples. Forests are discriminatory (not generative) learners and are iteratively built. They become better at predicting after each iteration.

In a forest, each tree is built (or "fitted") with random samples from a dataset. The, one random sample is selected from our dataset for testing and each tree makes one prediction: 0 or 1 (False or True). The forest majority's prediction is compared to the actual value. Samples that cause the majority to give an incorrect value are *more likely* to be chosen in subsequent iterations.

After a number of iterations, the forest majority becomes better at predicting than a single decision tree.

Each decision tree is built using features (attributes) and a target from the training dataset. Using Python3 with Pandas, we preprocess the training and testing dataset, we remove columns and subcolumns without meaningful information (iterative columns, χ^2 testing), we replace or drop missing values, we threshold continuous variable data. After preprocessing, we model decision trees individually and collectively (forests), we use hyperparameters to optimize model performance, and evaluate our model with accuracy, precision, recall, and F1-score.

This repository has several folders:

1. `python-demos`
   These are intended to be small and simple, showcasing one (or at the most, two) method. These scripts are designed to perform a specific task or set of tasks related to decision trees and random forests.

2. `for-kaggle`
   These are our top kaggle submissions.

3. `notebook`
   This was not in our original submission. It contains Caleb's notebook. He sent me it when we first met as a group and I forgot to include it.

4. `matlab-version`
   This was not in our original submission (although it was written before the four of us began python).

1 Main Folder

`matlab-version`

This folder was not in our first submission. (I included it here because I forgot it the first time. This was my first decision tree. I wrote it after reading Mitchell Chapter 03. In our write up, I discuss its design, more-so than the python version we wrote later. Please ignore if including it would impact anything.)

- **read_data:** Reads data from a CSV file and preprocesses it.
- **k_fold:** Splits data into k folds for cross-validation.
- **id3:** Builds a decision tree using the ID3 algorithm.
- **displayTree:** Displays the decision tree.
- **Gini_Index:** Calculates the Gini index.
- **Misclassification:** Computes the misclassification rate.
- **Entropy:** Calculates the entropy.
- **Gain:** Computes information gain for features.
- **predict:** Predicts the target using a decision tree.

Functions and Descriptions

`demo-basics.py`

Main header. Contains most classes and functions for all python demos.

1. `imbalance_ratio(y:pd.Series)`
   Summary: Calculates the imbalance ratio of the target variable.

2. `score(y, predictions)`
   Summary: Computes the number of correct and wrong predictions.

3. `confusion_matrix(y, predictions)`
   Summary: Generates the confusion matrix based on actual and predicted values.

4. `accuracy(y, predictions)`
   Summary: Calculates the accuracy and weighted accuracy of the predictions.

5. `standard_error(y, predictions)`
   Summary: Computes the standard error and weighted error of the predictions.

6. `remove_iterative(df:pd.DataFrame, columns_to_remove)`
   Summary: Removes iterative columns from the DataFrame.

7. `impute_missing_data(df:pd.DataFrame)`
   Summary: Imputes missing data in the DataFrame.

8. `drop_missing_data(df)`
   Summary: Drops rows with missing data from the DataFrame.

9. `purity(df_test, column_name)`
   Summary: Checks if a column in a DataFrame is pure.

10. `gini_index(y)`
    Summary: Computes the Gini index of a target variable.

11. `calculate_entropy(y)`
    Summary: Calculates the entropy of a target variable.

12. `best_split_threshold(df, column, impurity)`
    Summary: Finds the best split threshold for a numerical feature.

13. `numerical_split(df, column, best_valsmids)`
    Summary: Splits a DataFrame based on a numerical threshold.

14. `categorical_split(df, column)`
    Summary: Splits a DataFrame based on a categorical feature.

15. `calc_information_gain(impurity, column, branches, databelow, dataabove)`
    Summary: Calculates the information gain for a split.

16. `find_column_split_gains(df, impurity_method)`
    Summary: Finds the information gain for each column in the DataFrame.

17. `DecisionNode`
    Summary: Class representing a node in a decision tree.

18. `build_and_predict(df, column)`
    Summary: Builds and predicts using a decision tree.

19. `adjust(n_zeros, n_ones)`
    Summary: Adjusts the imbalance ratio for a given number of zeros and ones.

2 /python-demo

`drop-and-split.py`

Drop "NotFound" and split into train and validating sets

`impute-and-split.py`

Impute missing values and split train and validating sets

`one-tree.py`

Show just one tree train and test

`tree-graph.py`

Show a tree with graphviz

`gains.py`

Displays gains for *some* features

`demo-all-gains.py`

Displays gains for *all* features (iterative included)

`continuous_to_categorical.py`

Utility to convert continuous data to categorical data

`chi_squared_signf.py`

Used to calculate chi-squared and determine significance

3 /for-kaggle

`demo-70-onetree-nodrop-undersample.py`

Does not "NotFound" drop with undersampling: 70% score on leaderboard.

demo-card1.one-tree.no-drop.py

Does not ”NotFound” drop: 55% score on leaderboard.

Usage

To train our model, begin with our basic template:

```
# hello_trees.py

import numpy as np
import pandas as pd
import demo_basics as demo

# Load data
df =  pd.read_csv(os.getcwd() + '/train.csv')

forest = []

for i in demo.categorical:
    one_tree = demo.DecisionNode()
    one_tree.fit(df[i], df["isFraud"])
    forest.append(one_tree)
```

Then execute the following command:

```
python hello_trees.py
```

Usage

1. To fit our model, use *DecisionNode.fit*:

   ```
   import demo_basics as demo

   # load data
   df =  pd.read_csv(os.getcwd() + '/train.csv')

   # build forest
   forest = []
   for i in demo.categorical:
       one_tree = demo.DecisionNode()
       one_tree.fit(df[i], df["isFraud"])
       forest.append(one_tree)
   ```

2. To predict with our model, use *DecisionNode.predict*:

```
none_returned = 0
predictions = []
for (_,row) in X_test.iterrows():

    predict_of = []
    for idx,column in enumerate(demo.categorical):
        value = row[column]
        chance_of_1 = one_tree.predict(value)

        if chance_of_1 == None:
            none_returned += 1
            predict_of.append(np.random.randint(2))    # not found
        elif chance_of_1 < 0.5:
            predict_of.append(0)
        else:
            predict_of.append(1)
    vote = sum(predict_of)/len(demo.categorical)
    if vote < 0.5:
        predictions.append(0)
    else:
        predictions.append(1)

timer_end = time.time()
print(f"time to predict with one-tree node: {timer_end-timer_start}")

p_nones = none_returned/len(X_test)
print("none_returned=%.2f" % p_nones)
p_found = (1.0-p_nones)
print("found_accuracy=%.2f" % p_found)
```

3. How to create a submission for Kaggle

```
if 0:
    % for kaggle submission
    start_at = 472433
    out_index = range(start_at, start_at+len(df_test))
    out_predictions = predictions
    out_pd = pd.DataFrame({"TransactionID":out_index, "isFraud":out_predictions})
    out_pd = out_pd.set_index("TransactionID", drop=True)
    out_pd.to_csv(os.getcwd() + "/out_forest.csv")
end
```

Requirements

Ensure you have the following dependencies installed:

- Python (anaconda) version 3.11.5
- Additional libraries: NumPy, Pandas, ~~Scikit-learn (for comparing with our from-scratch model~~

License

This project is licensed under the MIT License.