XGBoost

a.k.a. Decision Tree Ensembles

Alistair, Joel, Nate, Peter

How the Algorithm Works: gradient boosting

Gradient Boosting: The objective of any supervised learning algorithm is to define a loss function and minimize it. The same is true for Gradient Boosting algorithm. Here, we have mean squared error (MSE) as loss-function defined as follows -

$$Loss = MSE = \sum (y_i - y_i^p)^2$$

where, y_i = ith target value, y_i^p = ith prediction, $L(y_i, y_i^p)$ is Loss function

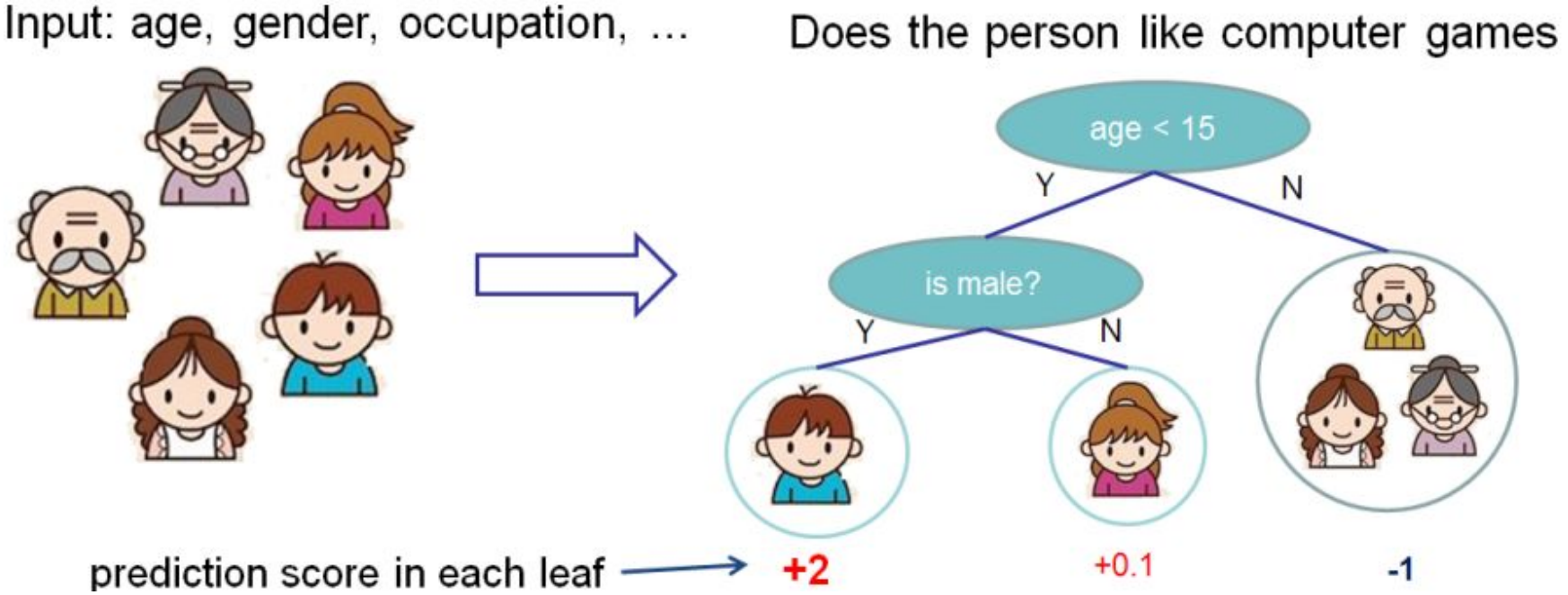
By using gradient descent and updating our predictions based on a learning rate, we can find the values where MSE is minimum.

$$y_i^p = y_i^p + \alpha * \delta \sum (y_i - y_i^p)^2 / \delta y_i^p$$

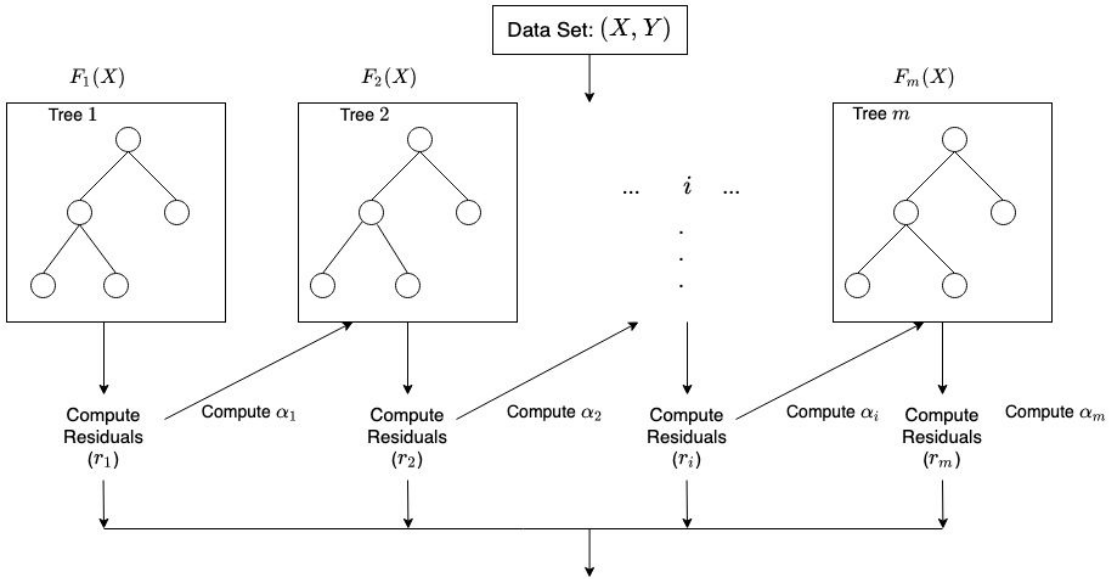
which becomes, $y_i^p = y_i^p - \alpha * 2 * \sum (y_i - y_i^p)$

where, α is learning rate and $\sum (y_i - y_i^p)$ is sum of residuals

How the Algorithm Works: decision trees



How the Algorithm Works: flow chart



$$F_m(X) = F_{m-1}(X) + \alpha_m h_m(X, r_{m-1}),$$

where α_i, and r_i are the regularization parameters and residuals computed with the i^{th} tree respectfully, and h_i is a function that is trained to predict residuals, r_i using X for the i^{th} tree. To compute α_i we use the residuals computed, r_i and compute the following: $arg \min_{\alpha} = \sum_{i=1}^{m} L(Y_i, F_{i-1}(X_i) + \alpha h_i(X_i, r_{i-1}))$ where $L(Y, F(X))$ is a differentiable loss function.

Advantages & Disadvantages

Advantages

- Increases the weight on variables predicted wrong to create a stronger and more precise model.
- Outliers have minimal impact.
- Ability to handle large sized datasets well.
- Premier combination of prediction performance and processing time compared to other algorithms.

Disadvantages

- May perform poorly with data outside the range of the original training data.
- Many hyperparameters which makes tuning more difficult.
- Higher risk of overfitting than the linear and logistic regression models.

Data processing to fit the model: no different from before

- Only numerical input values

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

- String class values must be converted to integer dummy variables

```
from sklearn.preprocessing import LabelEncoder
LabelEncoder()
label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)
```

Data processing to fit the model

- XGBoost may incorrectly assign ordinal relationships to categorical variables when they are actually nominal
- One-hot encoding: design matrix that retains the first factor instead of removing it

```
left-up, left-low, right-up,
right-low, central

1,0,0,0,0
0,1,0,0,0
0,0,1,0,0
0,0,0,1,0
0,0,0,0,1
```

```
# encode string input values as integers
columns = []
for i in range(0, X.shape[1]):
    label_encoder = LabelEncoder() feature
    label_encoder.fit_transform(X[:,i])
    feature = feature.reshape(X.shape[0], 1)
    onehot_encoder = OneHotEncoder(sparse=False)
    feature = onehot_encoder.fit_transform(feature)
    columns.append(feature)
# collapse columns into array
encoded_x = numpy.column_stack(columns)
```

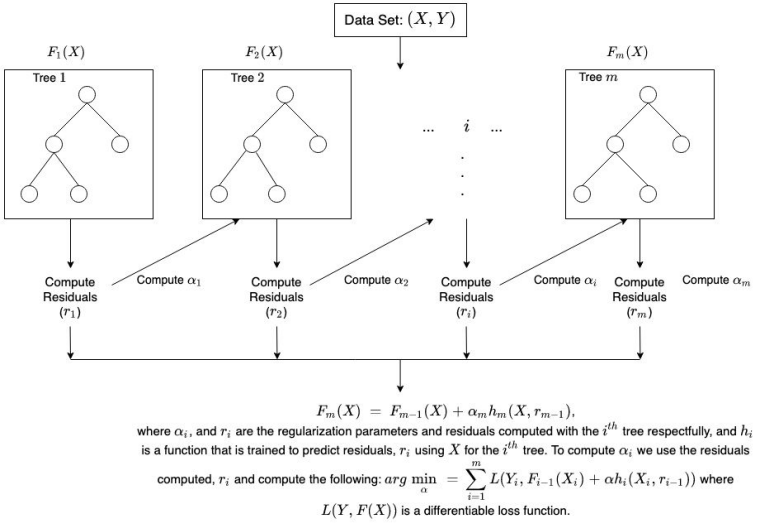
Data processing

- No explicit preprocessing standardization is required for tree-based models, but it can’t hurt to rescale the data, especially for large sets
- XGBoost was developed to handle sparse data (with missing values)
- Missing values can be set to the sparse value 0, imputed as the mean or median, or left as NaN for XGBoost to treat as a distinct value—a default direction is specified at every node of the tree

Hyperparameters & Tuning

- There are a lot of hyperparameters in XGBoost
- These are considered the most important:
  - Maximum tree depth (max_depth; default=6)
  - Learning rate (eta; default=0.3)
  - Minimum child weight (min_child_weight; default=1)
  - Number of sub-trees to train
  - Gamma (gamma; default =0)
  - L1 regularization (alpha; default=0)
  - L2 regularization (lambda; default=1)

Appendix



A Beginner's Guide to XGBoost
https://towardsdatascience.com/a-beginners-guide-to-xgboost-87f5d4c30ed7

XGBoost + k-fold CV + Feature Importance
https://www.kaggle.com/code/prashant111/xgboost-k-fold-cv-feature-importance/notebook

XGBoost with Python: Gradient Boosted Trees with XGBoost and scikit-learn
https://b-ok.cc/book/5005189/f7c6c0

How XGBoost Works
https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html

XGBoost: A Scalable Tree Boosting System
https://arxiv.org/pdf/1603.02754.pdf

Video with a simple explanation of XG Boost:
https://www.youtube.com/watch?v=PxgVFp5a0E4

Scikit Learn Documentation on Decision Trees.
https://scikit-learn.org/stable/modules/tree.html#classification

Mastering XGBoost
https://towardsdatascience.com/mastering-xgboost-2eb6bce6bc76

XGBoost documentation/Notes on Parameter Tuning:
https://xgboost.readthedocs.io/en/latest/tutorials/param_tuning.html

Model Tuning and Feature Engineering
https://towardsdatascience.com/model-tuning-feature-engineering-using-xgboost-ef819bccc82e