Critical Analysis Report

BUSINESS ANALYTICS

DATA MINING

Divyank Verma

Table of Contents

PART-1

EXPLANATION OF GRADIENT BOOST CLASSIFICATION MODEL

- In machine learning, boosting is a method in which a model is prepared by us and then based on the errors between the predicted and the observed values subsequent models are created to predict better result.
- In Gradient Boosting the residual error is calculated and based on that error subsequent decision trees are created to predict better result in respect to the target variable. In our report we have used the technique of gradient boosting in regards with the classification problem of Churn Prediction.
- Here we have taken a sample dataset to better explain the concept and working of the gradient boost technique in classification problem. People who likes snacks, there age, favorite color (independent variables) and if or not the person likes the movies (target variable).
- If we implement gradient boost on this sample dataset, we can explain how the residual errors are calculated and how does the gradient boosting classification work with the example.

Likes Snacks	**Age**	**Fav color**	**Likes Movie (Target V.)**	**Residual Value (1)**	**Residual Value (2)**
Yes	13	Blue	YES	0.3	0.1
Yes	86	Green	YES	0.3	0.5
No	45	Blue	NO	-0.7	-0.5
Yes	19	Red	NO	-0.7	-0.1
No	32	Green	YES	0.3	0.1
No	13	Blue	YES	0.3	0.1

- When we use Gradient Boost for classification, the initial prediction for all the individuals is the log(no of positive events/ no of negative events) or log(odds)
- Since here 4 people are liking the movie and 2 are not liking the movie therefore

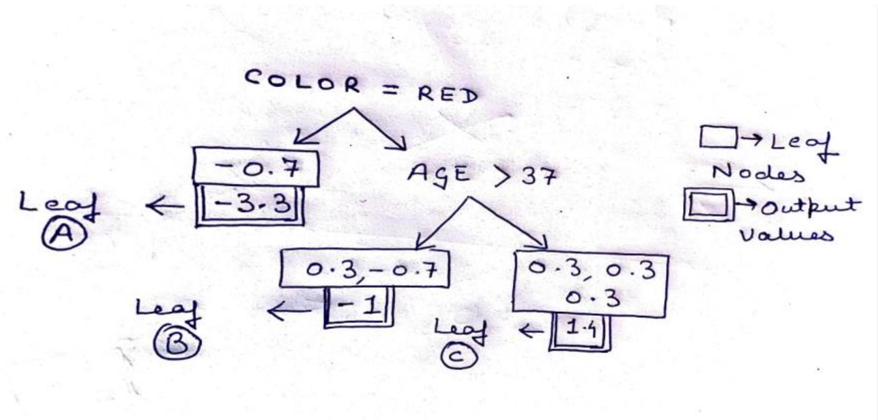
$$\mathbf{\log(4/2) = 0.7}$$

- To use this classification, we now have to convert this into the probability by using the logistic function which is:

$$\textbf{Probability} = \frac{e^{\log(odds)}}{1+e^{\log(odds)}}$$

- By using this function, we now have a probability of 0.7.(Taking 0.5 as the threshold to divide the class of YES(>0.5) and NO(<0.5).
- Once the probability is calculated our model now calculates its first set of residual values (see table column Residual value (1).

Residual Values = Observed – Predicted



Decision Tree 1

- We created our first decision tree with the help of our independent variables and created 3 leaf nodes. Leaf A, B and C.
- In next step we then calculate the output values of each leaf nodes by using the following formulae

$$\frac{\sum \text{Residual}}{\sum[\text{Previous probability x (1- Previous probability)}]}$$

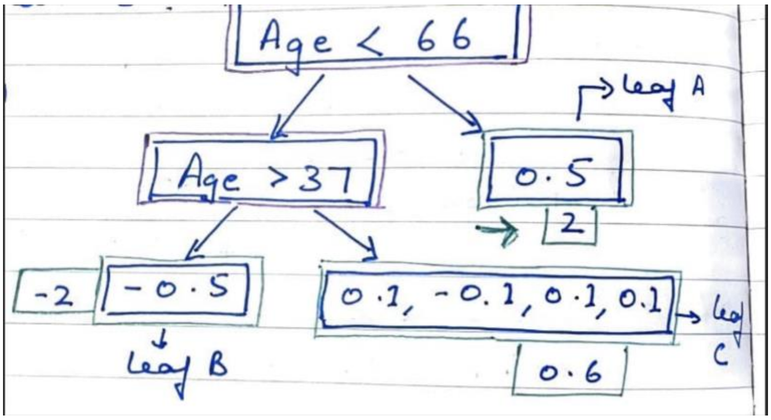
- Next Step is to plug in values of leaf A, B and C in the probability formulae to calculate the probability. For first instance we take 0.7 as our previous prob which was our original prob.
- Prob of liking the movie for Leaf A: -0.7/0.7x(1-0.7) = *-3.3*
- Prob of liking the movie for Leaf B: 0.3+(-0.7)/(0.7x(1-0.7) + (0.7 x (1-0.7)) = *-1*

- Prob of liking the movie for Leaf C: (0.3+0.3+0.3)/(0.7x(1-0.7)) + (07x(1-0.7)) + (0.7x(1-0.7)) = *1.4*
- Now once we have our output values of each leaf nodes we update our predictions by combining the initial leaf with the new tree values along with the learning rate.
- Assumption Learning Rate: 0.8 (In scikit learn default = 0.1)

log(odds) + [L.R. x Tree 1]

- For all the observations new values are obtained upon which new RESIDUAL VALUES (2) will be known to us.
- 1st observation: 0.7+(0.8x(-1))= -0.1 therefore prob = e^-0.1/1+e^-0.1 = 0.5 (we notice that this is worse than before that is the reason we use multiple decision trees to know the best result.
- 2nd observation: 0.7 +(0.8 x (-1)) = -0.1 therefore new prob is **0.5**
- 3rd observation: 0.7 +(0.8 x (-1)) = -0.1 therefore new prob is **0.5**
- 4th observation: 0.7 +(0.8 x (-3.3)) = -1.94 therefore new prob is **0.1**
- 5th observation: 0.7 + (0.8x91.4)) = 1.8 therefore new prob is **0.9**
- 6th observation: 0.7 + (0.8x(1.4)) = 1.8 therefore new prob is **0.9**

- New residual set of values are now calculated with these new predicted probabilities.(In table under the column Residual value (2)
- This now will help us create a new decision tree by using these new residual values.



Decision Tree 2

- On the second iteration new output values are calculated for each leaf node A, B and C as 2,-2 and 0.6 respectively.
- Leaf A: 0.5/(0.5 x (1-0.5)= 2
- Leaf B: -0.5/(0.5x(1-0.5)= -2
- Leaf C: (0.1-0.1+0.1+0.1)/(0.9x(1-0.9)) + (0.1x(1-0.1)) + (0.9 x(1-0.9)) + (0.9x91-0.9))= 0.6
- The process of calculating new values to get the new probability will repeat over again to get new predictions.
- This entire process is how the Gradient Boosting Classification works. This keeps on going until the model reaches to the minima that is very close to 0 but it does not reach exactly to 0 because we are using the Learning Rate which is essential for gradient boost. This process repeats until either the number of decision trees we specified is over or the model reaches the best minima it could according to the dataset and the parameters(along with hyperparameters value).

PARAMETERS WE CHOOSE FOR CHURN PREDICTION

- **Learning Rate:** Learning parameter is a hyperparameter which is added to the previous probabilities along with the previous tree's output values of individual leaf nodes. Lower values are generally taken for better results as lower leaps our model makes to get to the minima. In our case we got the best learning rate of 0.8 after tuning and feature selection.

Log(odds) + [LR x Tree 1]

- **N_estimators:** Number of the subsequent decision trees we are allowing our model to try and reach the best outcome. Using a lot of decision trees may result in a good result but our model could have high variance and overfit hence we tuned it using the GridSearchCV.
- **Max_depth:** Max depth determines how deep our decision trees will be built. Usually the range of max depth by default in scikit learn is 3 but the value can start from 1 and go till infinity. But it is to be noted if we allow our decision tree to grow at a very big depth then it may make leaf nodes too pure and it won't be good for our model.

- Best Parameters after we tuned our model and did the feature selection are as follows:
    - Recall score: **0.68** Precision Score: **0.38**
    - Learning Rate: 0.8
    - max_depth: 1
    - n_estimators: 20
- Once we got our recall and precision scores we started doing our cost benefit analysis based on our company assumptions.

ADVANTAGES OF GRADIENT BOOST

- This algorithm usually creates a model which works well on numerical or categorical features without pre-processing the output.
- It deals well with the missing data
- Forecasting accuracy is quite good as compared to other algorithms.

DIS-ADVANTAGES OF GRADIENT BOOST

- If our dataset has significant outliers then it may tend to focus to minimize the residual error for those observations and it may lead to overfitting as this model reduces inaccuracies and it is very good at it.
- Due to very high flexibility of the algorithm, the variables might tend to interact with each other and could affect the functionality of the model.
- Comparatively costly to run as it require more trees to reduce the errors than other models.

PART-2

COST BENEFIT ANALYSIS

INTRODUCTION

A leading call centre company is facing a hard time due to a large attrition(churn) rate of its employees. The company spends approximately $2000 for hiring a new employee and another $2000 for training each employee. To reduce the churn rate, the company has come up with a churn prevention strategy, where in the company will offer $1000 bonus to all the churning employees, so they stay in the company.

SCOPE

The scope of this report is to create a robust classification model to predict employee churn by implementing Gradient Boost algorithm. We would also be conducting cost benefit analysis, in terms of deploying a churn prediction model, calculate the expected cost per year if no model is deployed, compare it with the cost required to deploy the model and conclude if the company will save any money. For the purpose of this report, we will be considering few assumptions which are as follows:

- Currently, the company has 2000 employees.
- The annual churn rate is 40%.
- Of all the churning employees who are offered $1000, only 30% are expected to accept the bonus and stay.
- All the non-churning employees are expected to accept $1000 bonus, if the bonus is offered to them.
- The hiring cost of a new employee is $2000 and the training cost for a new employee is $2000.

COST BENEFIT ANALYSIS

Based on the assumptions, the company has 2000 employees, , and the annual churn rate is 40%. Considering this data, we will now calculate the cost incurred to the company if no classifier is deployed,

800*4000 = 320,00,00

=$3.2 million/year

We will now calculate the cost which could be incurred if a classifier is deployed.

For the purpose of running the gradient boost algorithm in our python code, we have taken churned employees as a positive label and a non-churned employee as a negative label.

The purpose of deploying a classifier is to simply find out the potential employees who will churn or not. In this process we will get 4 different scenarios (outcomes):

TP: - Churning employee is predicted as a churning employee.

FN: - Churning employee is predicted as a non-churning employee.

TN: - Non-churning employee is predicted as a non-churning employee.

FP: - Non-churning employee is predicted as a churning employee.

Based on the assumptions, we are aware of the fact that, the churning employees are 800 and the actual number of non-churning employees are 1200.

TP + FN = 800

TN + FP = 1200

Combining the recall formula and the recall score which we received, after running the python code, we get the following equation.

$$\frac{TP}{TP + FN} = 0.68$$

$$\frac{TP}{800} = 0.68$$

Therefore **TP = 544**

Since TP + FN = 800

544 + FN = 800

So, **FN = 256**

We also ran the python code to obtain the precision score to calculate the other parameters, so the precision score we got was 0.38.

Based on precision score,

$$\frac{TP}{TP + FP} = 0.38$$

$$\frac{544}{544 + FP} = 0.38$$

$$\frac{544}{0.38} = 544 + FP$$

544+FP = 1432

So, **FP = 888 (887.57 rounded)**

We know that TN + FP = 1200

So **TN = 312**

Actual	Churn (1)	544 (TP)	256 (FN)
	No Churn (0)	888 (FP)	312 (TN)
		Churn (1)	No Churn (0)
		Predicted	

Cost of TP = [544*0.3*1000] (Loss in revenue for employees who took the $1000 bonus) + [544*.70*4000] (Loss in revenue for employees who churn and disregard the bonus hence we had to hire and train new employees) – [544*0.30*1000](Savings as a result of retaining 30% of employees as no hiring and training will be expensed on them although bonus has been given) = **$1.52 million/year.**

Cost of FP = 888*1000 = **$888000**

Cost of FN = 256*4000 = **$1,024,000**

Cost of TN = $0(People who never actually churned and also were never given any bonus as they were not pred. as churning)

So, the total deployment cost is (TP+FN+FP+TN)= **$3,432,000**

CONCLUSION

Therefore, the cost of deploying the gradient boost classifier is **$3,432,000 = $3.4 million/year.** Therefore, if we deploy gradient boost algorithm, the expected savings is $3.2 million - $3.4 million = **-$200,000**. So, it will be a loss to the company if we deploy gradient boost. So, it is advisable to not use this model or we would need a different algorithm for better result.

REFERENCES

- To understand gradient boost classification, parameters used along with their importance.:

  1. https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

  2. https://statquest.gumroad.com/l/wvtmc

- To understand the Cost Benefit Analysis and conduct the calculations.: https://elearning.dbs.ie/mod/book/view.php?id=1281538&chapterid=97219