IÉSEG SCHOOL OF MANAGEMENT

Individual Assignment

Statistical & Machine Learning Approaches for Marketing
Prof. Minh Phan

Name:
POLANCO, Jorge

Index

1. **Page #2**: Business Case
2. **Page #2**: Datamart Preparation
3. **Page #3**: Feature Selection
4. **Page #3**: Choosing the Optimal Model
5. **Page #3**: Modeling Process
    a. **Page #4**: Logistic Classifier Model
    b. **Page #6**: Random Forest Model
    c. **Page #7**: Gradient Boosting Machine Model
    d. **Page #8**: Bayesian Additive Regression Trees Model
    e. **Page #10**: Adabag Boosting Machine Model
6. **Page #12**: Conclusions
7. **Page #13**: Bibliography

Business Case

The Analytics team of an important bank wants to develop an algorithm, which could help them to identify whether or not a client is going to subscribe to a term deposit (pointed out as a dummy, where one is yes and zero no). In order to identify potential subscribers, the analytics team shared with us, the consultancy group, the following inputs:

- **Bank client data**: Here we can find what is known in the industry as ‘application data’, which is a client’s private information.
- **Information of the last contact**: This is related to the last contact that was made to the client.
- **Marketing attributes**: These attributes solve the question: how many times a client was contacted by the bank?
- **Social and economic context attributes**: Also, we have some national economic metrics.

All this information was already merged and separated into two tables, which we are going to use as a training set and a test set.

Datamart Preparation

Considering all these metrics and others related to these ones, we are about to propose to the bank’s executives some models that we consider appropriate for this particular scenario. However, before presenting the models, in the following sections, we are going to explain how we treat the database before the modeling.

It is important to mention that we used R programming language on the Jupyter Notebook interface to manipulate the data. Therefore, in order to prepare our final database, we followed the following steps:

1. **Importing the files**: We imported the csv files and transformed them into Dataframes.
2. **Binning categories**: This is useful to set combinations of qualitative variables or ranges of quantitative variables.
3. **‘Dummification’**: Once we have created the binning categories, we can transform them into dummy variables. It is important to delete one dummy variable in each set of category in order to avoid multicollinearity.
4. **Creating metrics**: From the original features, we can create new ones related to them. In this case, we created statistical metrics (means, max and min by age), squared, cubic and interaction metrics.
5. **Factorizing features**: Most of the libraries request that the dependent variable has to be a factor.

Feature Selection

After the Data Engineering process, we ended up with 88 features. However, not every independent variable is going to contribute to the forecasting power of our models. Therefore, we have estimated the Fisher score of each variable, the scoring tends to reflect the significance of each feature vs the dependent variable.

After comparing the AUC performance – a topic that we are going to discuss in the coming sections -, we decided to keep the TOP 20 features (according to the Fisher scores). In this case, here are the selected independent variables:

1. **euribor3m** – euribor 3 month rate
2. **euribor3m_cons.price.idx** – Combination (multiplication) between these two variables
3. **emp.var.rate_cons.price.idx** – Combination (multiplication) between these two variables
4. **emp.var.rate** – employment variation rate
5. **euribor3m_sqr** – Squared transformation of euribor3m
6. **euribor3m_cubic** – Cubic transformation of euribor3m
7. **emp.var.rate** – employment variation rate
8. **emp.var.rate_cons.price.idx_euribor3m** – Combination among these three variables
9. **euribor3m_emp.var.rate** – Combination (multiplication) between these two variables
10. **emp.var.rate_cubic** – Cubic transformation of consumer confidence index
11. **emp.var.rate_sqr** – Squared transformation of employment variation rate
12. **pdays.999** – Client was not previously contacted
13. **poutcome.success** – success outcome of the previous marketing campaign
14. **pdays.0to10** – Client was contacted between 0 to 10 days
15. **contact.cellular** – contact communication by cellular
16. **cons.price.idx** – consumer price index
17. **cons.price.idx_sqr** – Squared transformation of consumer price index
18. **cons.price.idx_cubic** – Cubic transformation of consumer price index
19. **euribor3m_mean** – mean euribor 3 month rate by age segment
20. **month.may** – last contact month of year

According to these results, we can infer that the economic context and the last contact are the main drivers of our dependent variable. Something interesting here is the shape of the relationship, square and cubic transformation, which are the most relevant metrics.

Choosing the Optimal Model

Traditionally speaking, metrics such as Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC) and AdjustedR^2. However, these metrics are calculated based on the training set, which could underestimate the test error. As an alternative approach, we are going to use the Cross-Validation method, because this will directly estimate the test error dividing in two 'bank_mkt_train' file. By this, we created 'train_processed2' and 'test_holdout_processed' dataframes, which this last is what is also known as 'validation set'.

As previously mentioned, we have followed a Cross-Validation in order to create a set of models in order to minimize the error. Having established our hyper-parameters, we defined how many times this model is going to be iterated, and then automatically we will get the average AUC scores, resulting in an ensemble model.

There is something very important to point out regarding the model selection, as we are about to present, almost all our models presented a test AUC score around 80%. Although in normal circumstances, and considering similar performance, we would have chosen the easiest model to explain to non-technical clients; for this occasion, we will select the absolute best one.

Modeling Process

Considering the characteristics of the problem to resolve, we are dealing with a classification problem. Why? Because the main goal is to identify those potential subscribers from the rest.

Several models were tested before selecting the ones with the highest AUC percentage. However, it is important to mention that most of the cases, these models presented a test AUC around 79%, excluding the winner, which presented a score of around 81% in the Kaggle competence.

The models that we are proposing to the bank's analytics team are the following:

- Logistic Classifier Model
- Random Forest Model
- Latent Dirichlet Allocation (LDA) Classifier Model
- Binomial Classifier Model
- Gradient Boosting Model

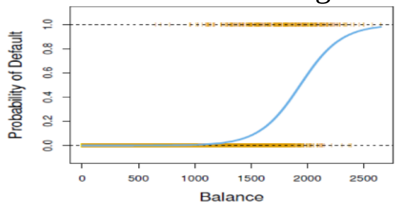
a) Logistic Classifier Model

i. Model Explanation

The Logistic Models part of the Linear Regression approach, but with the Sigmoid transformation to draw a behavior between f(x) = 1 and f(x) = 0. The building of the new function is created through the following transformation:

$$f(x) = \frac{e^x}{1+e^x} \rightarrow p(x) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$

The new function will have a shape similar to the following:



Finally, using the natural logarithm rules, we will end up with this final form, which will help us to understand the effect of each one of the involved features:

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 X$$

Once we are here, the model will be trained estimating the parameters using Maximum Likelihood:

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

In order to find an optimal level (at least a local one) resolving the Gradient Descent Optimization algorithm, here is the function to be optimized:

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i \left(y_i - p(x_i; \beta)\right) = 0$$

Maximizing this function is how we will get our coefficients, which yield a number close to one for all individuals who subscribed, and a zero for all individuals who did not.

ii. Results

After iterating 10 times, the aggregated results, which is the average AUC of all the models, we got a test AUC of 79.60%, which is aligned to the 79.98% AUC training score.

Considering these results, we can assume that there is no overfitting nor overtraining due to the small between the training and test AUC. Normally here, we would retain the best hyper-parameter model, but in this case, we did run the default model.

iii. Why (or why not) this model?

PROs:

- This model was designed for this particular purpose
- It is easy and fast to compute
- The outcomes are probabilities
- Very intuitive
- Tends to have Low variance

CONs:

- We have to work to make it fit when using nonlinear features
- It can suffer from outliers

b) Random Forest Model

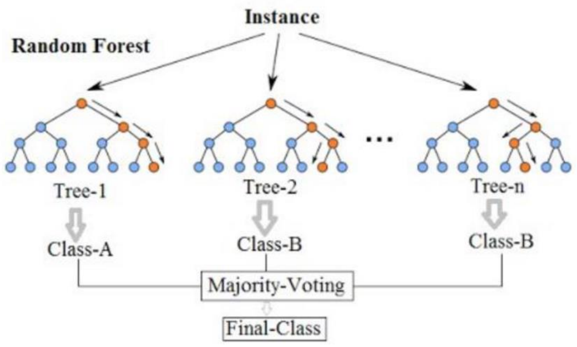
i. Model Explanation

Before talking about Random Forest, we have to explain the Decision Trees models. The idea here is to split in a binary way by some features. Unlike regression models, the Residual Sum of Squares cannot use here as a splitting criterion. There are some alternative methods: Classification Error Rate, Gini Index and Cross-Entropy/Deviance. The last one is the most popular criterion; here we can see its cost function:

$$D = -\sum_{k=1}^{K} \widehat{p_{mk}} \log(\widehat{p_{mk}})$$

After identifying an optimal model, it is very easy to visualize it. However, the Decision Trees models tend to have high variance and low accuracy vs another type of models. There are some ensemble methodologies available to resolve these issues; amongst them, we can find Random Forest models.

Long story short, the idea here is to recreate multiple trees, and then calculate the average of all these previous models. In this way, we can reduce the variance and strength of its accuracy performance.



To make this happened, using bootstrapped training samples (the creation of more observations from the original training set), we can distribute a random sample of predictors across different trees vs our single target variable. As one could see in our script, we tested results with several numbers of variables per tree; however, it is recommended to follow the 'rule of thumb': $m \approx \sqrt{p}$, which is 9 in this case.

ii. Results

For this model, we set the following series of hyper-parameters to be tested:

- **Number of Trees**: [100, 250, 500, 750, 1000]
- **Number of variables for splitting (mtry)**: [1, 2, 5, 9, 18, 36]

According to the algorithm, the best learner was ntree = 500 and mtry = 3. Therefore, taking this model, the test AUC score was 76.73%. Unlike what we did with the previous model, here we are selecting the best model base on its hyper-parameters.

iii. Why (or why not) this model?

PROs:

- It handles missing values
- Easy to understand
- Reduce variance (from Decision Trees)
- Highly reduces the chances to have overfitting

CONs:

- We have to spend too much time in the tuning
- Hard to visualize
- Biased in multiclass problems

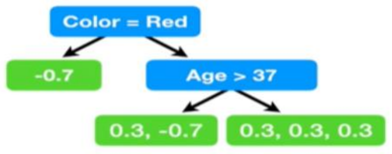
c) Gradient Boosting Machine Model

i. Model Explanation

When we use Gradient Boosting for Classification, in the first stage it takes the 'odds' as an initial prediction for every observation. To calculate these odds, GBM follows a similar path than in Logistic Regressions, the first prediction is calculated as follows:

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

From here, it is common to establish a threshold in order to classify an observation as one or zero; naturally, it could be 0.5, but this could depend on a particular case or in how conservative one wants to be. Once we have the first prediction, then we can estimate the first residual values, which will be the observed vs the probability previously defined for every observation.
These residuals will be plotted in each one of its respective leaves. We would end up plotting something like this:



Now, because our predictions are in terms of log(odds) and the leaves are derived from a probability, we need to transform one of these numbers to make them comparable. When using Gradient Boosting for classification, the most common transformation formula is the following:

$$\frac{\sum Residual_i}{\sum[Previous\ Probability_i(1 - Previous\ Probability_i)]}$$

The output of this formula will substitute the value of each leaf, so now we can multiply this value by the value of the residual. This process is repeated and sum all the tree values. Finally, this odd can be used as input into the prediction formula to have our final output.

ii. Results

In this case, we set a hyper-parametric tuning, in which we test the following attributes:

- **Distributions**: Gaussian, Bernoulli and Poisson.
- **N.trees**: from 100 to 1000
- **Interaction.depth** (maximum nodes per tree): from 5 to 8
- **Shrinkage** (learning rate): 0.01

According to the algorithm, the best learner was a distribution = Gaussian, ntree = 882 and interaction depth = 5. Therefore, taking this model, the test AUC score was 80%. It is important to point out that not every combination was ran, because computationally speaking, it would be very costly. Therefore, using the 'makeTuneControlRandom', we established a max of interactions equal to 5.

iii. Why (or why not) this model?

PROs:

- It handles missing values and transform variables
- Reduce variance (from Decision Trees)
- Highly reduces the chances to have overfitting

CONs:

- We have to spend too much time in the tuning
- Sensitive to outliers and noisy data
- Hard to visualize
- It can overfit if run too many iterations

d) Bayesian Additive Regression Trees Model

i. Model Explanation

This non-parametric model, as presented in the previous ensemble models, put together a bunch of 'weak learners' to provide a clearer picture of the whole. To easily visualize it, this model creates several 'hyper-rectangles' in a plane, so combined, and with many iterations, they could describe the actual data.

As the name suggests, the mere base of this model comes from the classic Bayesian model, which assumes a distribution and a likelihood base on observed data. This last point is represented with the following formula:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

This model consists of two parts: Sum of trees model and a regularization prior on the parameters involved, which consists of the following:

$$P(T_1^M, \dots, T_m^M, \sigma^2) = \left[\prod_t P(M_t|T_t)P(T_t)\right] P(\sigma^2)$$

Where:

$P(T_t)$ is the tree structure

$P(M_t|T_t)$ is the leaf parameters given the tree structure

σ^2 is the error variance, but for classification problems, this is equal to one

like in the other models, the likelihood is normally distributed

The idea here is to make our data flow from one decision tree to another, but only residual data can reach the next tree. What regularization does here is to control the flow of the data, restraining or allowing the pass between the different trees. Doing this, our model tries to prevent overfitting.

The classification version of a Bart Model is extended to the following probit model setup (instead of logit model used for the regression version). A probit model is represented as the following:

$$p(x) = P[Y = 1|x] = \phi[G(x)]$$

Where $G(x) = \sum_{j=1}^{m} g(x; T_j, M_j)$

Φ is the standard normal cumulative distribution function

After replicating these steps with each one of the decision trees, we have to induce a sequence of sum-of-trees functions, which is summarized by this:

$$p^*(x) = \phi\left[\sum_{j=1}^{m} g(x; T_j^*; M_j^*)\right]$$

ii. Results

This model is giving us the second-best results, having 10 iterations in the Cross-Validation Holdout process; the Bart Model is achieving a test AUC score of 80%, this having a number of trees equal to 50.

As presented in previous models, this small gap between the AUC scores tells us that there is a small chance to incur in overfitting. Considering the small difference, either Bart Model or Adabag Model could be a great option to adapt to this particular case.

iii. Why (or why not) this model?

PROs:

- Nowadays, It is considered as a state-of-the-art model
- It works well where the number of variables is very large
- It considers prior knowledge
- Reduce variance (from Decision Trees)

CONs:

- With strong confounding (when an independent variable has an effect over other dependent and independent variable), Bart can exhibit severe bias
- Hard to visualize
- It can be affected by irrelevant variables
- It is hard to tune

e) Adabag Boosting Machine Model

i. Model Explanation

This model has as a starting point the Random Forest model. The main difference is that with Adaboost, the trees are usually just a node and two leaves (also known as 'stumps'). On the other hand, a Random Forest Tree can have any level of depth. In other words, what are we creating here is a 'forest of stumps'.

Another important contrast, unlike Random Forest, not all the trees have the same weight decision. This is linked to the fact that with Adaboost, order matters; the error that the first stump does influence how the second stump is made, and go on.

As in GBM, we are going to start establishing a starting point, which the sample weight (1/number of observations). Nevertheless, after making the first stump, the initial weights will change to guide how the next stump is going to be created. In order to evaluate the first try, the Gini Index is the most popular choice for this model:

$$G = \sum_{k=1}^{K} \widehat{p_{mk}}\,(1 - \widehat{p_{mk}})$$

From the lowest Gini Index, the respective feature is going to be selected for the next stump. As mentioned, not all stumps have the same weight. In order to be every time more accurately, the stump importance is determined on how well it classifies the samples.

This weight will exactly be the stump's total error, which has to be a number between zero and one. The formula to calculate the weight is the following:

$$Sample\ weight = \frac{1}{2}\log(\frac{1 - Total\ Error}{Total\ Error})$$

This process continues through all the variables; once we have read each one, the next step is to do the stumps learn from the previous by increasing or decreasing its sample weight. To achieve this, we have to scale the new sample weight as follows (in case of wanting to decrease the weight, we only have to put a negative sign in 'sample weight'):

$$New\ sample\ weight = \ previous\ sample\ weight * \ e^{Sample\ weight}$$

Once having the new sample weights, we have to normalize them in order to make the sum of all the observations equals to one. Regarding the next selection of stumps, we can use the sample weights to calculate Weighted Gini Indexes to determine which variable should split the next one. Finally, we take all the stumps to see what they selected for a particular target variable, aggregating the values of the 'positives' and the 'negatives', thus we can decide which side is the winner.

ii. Results

The Adabag Boosting Machine Model was the best performer among the presented set of models. In the hyper-parameter tuning, our best model came up having n.trees equals to 882 and an interaction.depth of 5, presenting a test AUC score of 80%. It is important to mention that the results were averaged after 10 iterations.

iii. Why (or why not) this model?

PROs:

- It handles missing values and transform variables
- Reduce variance (from Decision Trees)
- Highly reduces the chances to have overfitting
- It can leverage several weak-learners

CONs:

- We have to spend too much time in the tuning
- Hard to visualize
- Sensible to outliers
- Sensitive to outliers and noisy data

Conclusions

After this experience creating a model from scratch, we have learned the importance of each one of the steps in the funnel of the model creation. Even before writing a single code line, it is important to understand what kind of problem we are dealing with; this is important because it will delimit the list of models we could use.

Once we have selected an approach, now we can treat our data. In this project, we created binned and dummy variables, which helped us to create other variables such as the mean of x variable by a particular segment (age, job, marital status, etc.). Another strategy that helped us to increase our model's performance (yes, we worked in a back and forth way) was the creation of squared, cubic and combination metrics. Through these variables, we could fit our model in a better way. As learned in class, we had to be very careful in not overusing these tactics because it could lead to overfitting.

Regarding the feature selection, we rely on Fisher Scoring methodology in order to identify the most significant relationships between independent variables and the dependent variable. Once again, back and forth, we tested different numbers of independent variables in the model. We found out that in this case, the models showed a better performance when they run with around 20 features, the ones that we describe at the beginning of the report.

Now we have everything ready to start modeling, in order to create the best possible model, we have adapted Cross-Validation Holdout methodology in order to avoid that a particular selection could affect our model's performance. Because some models are computationally expensive, we tended to delimit the number of times we are running these models.

Another good strategy was doing some tuning hyper-parameter in the CV process. This helped us to improve and make even more stable our performance.

Finally, it is true that some models tend to be more accurate versus others; in this sense, generally speaking, ensemble models usually present better AUC scores than single models such as decision trees. However, not because one model is more statistically complex than others, it does not mean this is better. Each particular case could have the best possible fit with any model.

Bibliography

CFI. *CFI*. n.d. https://corporatefinanceinstitute.com/resources/knowledge/other/boosting/. 26 March 2020.

Chipman, Hugh, Edward George and Robert McCulloch. "Bayesian Additive Regression Trees." *Wharton* (2008): 42. http://www-stat.wharton.upenn.edu/~edgeorge/Research_papers/BART%20June%2008.pdf.

James, Gareth, et al. *An Introduction to Statistical Learning with Applications in R*. New York: Springer, 2017.

Kim, Jaehyeon. *Jaehyeon's blog*. n.d. http://jaehyeon-kim.github.io/2015/01/Benchmark-Example-in-MLR-Part-I.html. 26 March 2020.

MLR. n.d. https://mlr.mlr-org.com/articles/tutorial/integrated_learners.html. 26 March 2020.

Mueller, John Paul and Luca Massaron. *Machine Learning for dummies*. New Jersey: John Wiley & Sons, 2016.

UWYO. n.d. https://www.cs.uwyo.edu/~larsko/ml-fac/01-classification-exercises.pdf. 26 March 2020.