Allstate Purchase Prediction Challenge

Big Data – Final Project

May 20, 2017

Lan Yan Xin Yu Siqi Ren

Table of Contents

Project Summary

The project predicts the car insurance coverage options purchased by individuals using machine learning. Basically, the whole project consists of two parts. The first part is data modeling. In this part, we trained several models based on the cleaned dataset, using pyspark 1.6.0 as the modeling tool. The performance of different models was evaluated by the accuracy rate. The second part is the construction of the system front end, which should be interactive and user-friendly.

The purpose of this project is to build a prediction system with relative high accuracy to help the firm to get a better understanding of customer transaction. With the prediction system, the firm will be able to get a more precise target segment and offer a different level of product and service to different target customer.

Data Description and Preparation

Data sources

The original dataset, which was named as “Allstate Purchase Prediction Challenge”. was downloaded from Kaggle,

Link: https://www.kaggle.com/c/allstate-purchase-prediction-challenge

Data Preparation

The original dataset consisting of 665,250 entries. It contains transaction history of customers that purchased a policy, total of 26 variables. There are 7 car insurance options (A, B,C,D,E,F,G), each of them has 2 to 4 possible values identified by integer number (Figure 1.0).

Coverage Option:

Option	A	B	C	D	E	F	G
Possible Value	0,1,2	0,1	1,2,3,4	1,2,3	0,1	0,1,2,3	1,2,3,4

Table 2-1

Sample data:

customer_ID	shopping_	record_ty	day	time	state	location	group_siz	homeown	car_age	car_value	risk_facto	age_oldes	age_youn
10000000	1	0	0	08:35	IN	10001	2	0	2	g	3	46	42
10000000	2	0	0	08:38	IN	10001	2	0	2	g	3	46	42
10000000	3	0	0	08:38	IN	10001	2	0	2	g	3	46	42
10000000	4	0	0	08:39	IN	10001	2	0	2	g	3	46	42
10000000	5	0	0	11:55	IN	10001	2	0	2	g	3	46	42
10000000	6	0	0	11:57	IN	10001	2	0	2	g	3	46	42
10000000	7	0	0	11:58	IN	10001	2	0	2	g	3	46	42

married_c	C_previou	duration_	A	B	C	D	E	F	G	cost
1	1	2	1	0	2	2	1	2	2	633
1	1	2	1	0	2	2	1	2	1	630
1	1	2	1	0	2	2	1	2	1	630
1	1	2	1	0	2	2	1	2	1	630
1	1	2	1	0	2	2	1	2	1	630
1	1	2	1	0	2	2	1	2	1	638
1	1	2	1	0	2	2	1	2	1	638

At the beginning of data preparation, we observed that there are three variables contained missing values in the dataset. First is “risk_factor”, which was NA and was 36.1% of the dataset. Since different car have different evaluation on risks, we used a linear regression model and generated values for those missing risk_factor. C_previous and duration_previous also contained missing values. We analyzed that those two records with NA values may represent that customer does not have previous purchasing records, therefore we replaced those missing values with 0.

Then we grouped states based on regions into 5 categories. Then we made some categorized variables based on original data: timeofday, weekend, family, agediff, and individual.

Variables	Value	Meaning
timeofday	evening/day	Time when customer gets quote
weekend	Yes/No	Yes=weekend, No=weekday
family	Yes/No	Customer who is married, has group size >2, and age of youngest < 25
individual	Yes/No	Customer whose group size was 1
agediff	(integer)	Age difference between oldest and youngest in the group

Table 2-2

Then we dummied the variables: car_value, timeofday, weekend, family, individual.

Given that the ‘numClasses’ attribute of both multinomial logistic model and decision tree model contains ‘0’ class by default, while some of the options contain no ‘0’ value. Therefore, we transformed without ‘0’ to (y-1), where ‘y’ indicates the actual value.

Option	A	B	C	D	E	F	G
Possible Value	0,1,2	0,1	0,1,2,3	0,1,2	0,1	0,1,2,3	0,1,2,3

Table 2-3

Methodology

Since the original dataset have 7 coverage options, which means that there are 2,304 possible combinations. We chose not to predict the coverage combination since we find that most combinations will not be used in reality. Also, predicting the combination will not only takes vast amount of machine process, but also result in very low accuracy. Instead, we believe that it is better to predict the 7 individual options for each customer and combine them, rather than predict the entire combination using a single model.

Data Partition:

To evaluate the performance of different models, the dataset was randomly divided into 2 part. The training set contains 80% of the records, which is used to train the model, while the left 20% is marked as testing dataset, which is used to evaluate the performance of different models. Given that the models are used to predict a categorical response, we will use precision rate as an evaluation criteria.

Multinomial Logistic Regression:

As shown in the Data Description part, the 7 options are all categorical variables containing 2 or more possible values. Thus, it is not appropriate to use common logistic model since it can only predict values with 0 and 1. To deal with the multiclass classification problem, the multinomial logistic regression model was constructed. By using this method, we were able to generate the probabilities of different possible outcomes. Hence, multinomial logistic regression can be used in predicting possible values in this project.

Decision Tree:

Decision Tree is a data mining method which can be used to predict the value of a variable based on several input variables. The tree is constructed by splitting the source into several subsets based on specific impurity rule. By specifying the impurity measure, one can determine the best split of the tree. Decision tree can be used to predict both categorical value and numerical value. Thus, we decided to use classification tree to build the model.

Result and Finding

Data Modeling:

Multinomial Logistic Regression:

The multinomial logistic regression model was built using the 'LogisticRegressionWithLBFG' module in pyspark mllib package. Firstly, we tried to model option A with multinomial logistic regression, it came with R^2 of 0.13, which is very low. The accuracy for each class was lower than the baseline derived from random guess. Therefore, the regression model was abandoned.

Decision Tree:

The decision tree model was built using the 'Decision Tree' module in pyspark mllib package. The model contains the following important attributes: numClasses, categoricalFeaturesInfo, impurity, maxDepth and maxBins. While constructing models for 7 options, 3 of them were kept unchanged, which were 'impurity', 'categoricalFeaturesInfo' and 'maxDepth'. The 'numClasses' was differed according to the possible value of the specific option. The attribute 'maxDepth', which indicates the maximum depth of a tree, was manipulated to improve the performance of the model.

For each model, the initial 'maxDepth' was set to 3, and then it was gradually increased to see if the accuracy of prediction in test dataset was improved. Take Option A as an example, we get a precision of over 0.7 for class '0' and class '1' and only 0.35 for class '2'. In spite of the low precision in one class, the overall performance of decision tree was still higher than regression model.

By iterating 'numClasses' and 'maxDepth' for different options, we got the following results for each coverage options (table 4-1, table 4-2).

Options	numClasses	maxDepth
A	3	8
B	2	5
C	4	6
D	4	6
E	2	4
F	4	8
G	4	8

Table 4-1

A	B	C	D	E	F	G
Precision:	**Precision:**	**Precision:**	**Precision:**	**Precision:**	**Precision:**	**Precision:**
0: 0.74	0: 0.60	0: 0.73	0: 0.43	0: 0.67	0: 0.67	0: 0.49
1: 0.75	1: 0.55	1: 0.64	1: 0.48	1: 0.52	1: 0.46	1: 0.48
2: 0.35		2: 0.71	2: 0.68		2: 0.55	2: 0.52
		3: 0.59			3: 1.0	3: 0.44

Table 4-2

The accuracy rate of the model would firstly increase with higher maxDepth. However, when the value of maxDepth increased to a specific point, in would incurred problem of overfitting and caused worse performance.

Frontend Construction:

To create an interactive, real-time and user-friendly frontend, we used Django, a python-based web-frame, to build a web application which can browsed online. AWS RDS MySQL instance is used as the backend database of the transaction information.

The basic idea of our frontend is to not only allow users to view history transaction information and prediction outcome, but to enable them to add new transaction information and get prediction in real-time. Thus, we choose to embed the model in the application.

When it comes to that how to embed our models into the application, we firstly considered to save the models and load them every time when a prediction request is submitted. However, by doing, the following problems appeared:

1. Complicated requirements: the users are required to install a lot of software to browse the application, including Django, MySQL, hadoop and spark.
2. High demand on hardware: since for each coverage option there is an individual model, which means the system has to process 7 model at the same time. Thus, the demand on hardware was very high. Besides, the process was time-consuming as well.

Therefore, we decided to import all the independent variables into database, then define each model as a function in the model.py file. Each time the user add new transaction information, the data will be stored in the database. The system will automatically read the data and return the outcome when the user request the detail page of each transaction.

The web application was deployed to AWS by using Elastic Beanstalk. An URL was assigned to the app, and hence, there is a trouble-free way for users to enter the system without running hadoop and Django at the same time.

Conclusion

Based on the decision tree generated, we noticed that there are several factors affect most on the final option chosen by the customer: cost, risk_factor, duration_previous, c_previous. Therefore, we concluded that the situation of the car, the customer's purchase history with the company and the suggested cost of the coverage option have a great influence in customers' final decision.

Through this project, we learned that how data cleaning and preparation are important to the whole project. Upon enough understanding of the variables, we can deploy a good modeling strategy without wasting large amount of time in running code multiple times. In dealing with such prediction of combination of values, we conclude that predicting single value then combine them provides much better precision than predicting the whole combination.

Appendix

Variable Descriptions

customer_ID - A unique identifier for the customer
shopping_pt - Unique identifier for the shopping point of a given customer
record_type - 0=shopping point, 1=purchase point
day - Day of the week (0-6, 0=Monday)
time - Time of day (HH:MM)
state - State where shopping point occurred
location - Location ID where shopping point occurred
group_size - How many people will be covered under the policy (1, 2, 3 or 4)
homeowner - Whether the customer owns a home or not (0=no, 1=yes)
car_age - Age of the customer's car
car_value - How valuable was the customer's car when new
risk_factor - An ordinal assessment of how risky the customer is (1, 2, 3, 4)
age_oldest - Age of the oldest person in customer's group
age_youngest - Age of the youngest person in customer's group
married_couple - Does the customer group contain a married couple (0=no, 1=yes)
C_previous - What the customer formerly had or currently has for product option C (0=nothing, 1, 2, 3,4)
duration_previous - how long (in years) the customer was covered by their previous issuer
A,B,C,D,E,F,G - the coverage options
cost - cost of the quoted coverage options

Figure 1.0