Hunt the Headhunter

Team members (last name in alphabetic order):
Yichang Chen, Rui Han, Ziwei Jiang
Varoon Kitayaporn, Chenxuerui Li

Overview

The business of headhunting is very competitive and doing headhunting in the traditional way like skimming through resumes and cold calling to random people is quite inefficient. We would like to provide a tool to increase headhunters' efficiency to find people who want to get a new job, thus reducing cost and increasing the profits.

After data exploration and data preprocessing, we built several machine learning models, which include logistic regression with lasso (ridge), decision tree, random forest and XGBoost Classifier. And then we choose XGBoost Classifier as our optimal model.

Besides that, the most successful part of our project is that we apply this model to solve one profit_optimization problem, which can maximize the profits the headhunter can gain by deciding which certain candidates they need to contact. For some specific position to fill, the headhunter just needs to input the possible candidates and number of candidates that they want to find, and then the model will automatically return the optimal result of who and how many people they need to reach out.

Data Description

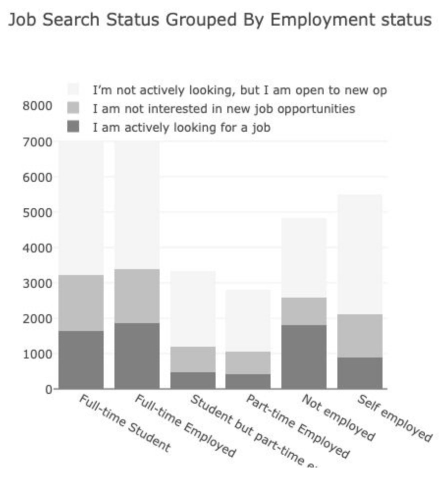
Our model is based on 2018 developer survey data from stackoverflow website. It is generated from Stack Overflow Annual Developer Survey, reflecting all aspects of developers information from their personal information such as residence, age, gender, education to career information such as job satisfaction, job search status, type of developer, programing language they are working with. Besides, it also contains data about how these people react to online advertising but this is not our focus at this moment.

After reading through all the features we have, we found this survey very valuable to the headhunting company, since we can utilize the information that a headhunting company can gather to predict whether someone wants to seek for a job or not. Our model will be a useful tool for headhunters to find the people who want to get a new job efficiently. In this way, we can reduce the people headhunting reach out to, thus increasing their profits.

Data Exploration

At first, we did data visualization to give us an intuition of what the data looks like. Not surprisingly, the majority are software developers and male. Besides, we looked at our independent variables, and found some features relevant with job searching status. Typically, their undergraduate major is computer science and just accepted their first job in the last year.

Then, we picked some important features to give us more information for our next step - data preprocessing - since we need to fill some None values with the most common value for that variable. It seems that most of the developers are in their mid 20s to mid 30s, with no dependents, and their parents' highest education level are bachelor or below. We found that family conditions have a big impact on their willingness to find a new job. In other words, if a developer has no dependents and is still young, also with higher educated parents, probably they are more likely to take the risk of stepping out of the comfort zone and finding a new job.



Data Preprocessing

1. Feature selection and feature engineering

In this part, we merge some classes based on domain knowledge and descriptive statistics.

The raw data has 129 columns, most of which are categorical variables. Since our final clients are headhunting companies, some of these columns are not in our interest. For example, reaction to Adblocker, view on AI, whether you like Stackoverflow or not; these features are not responsible for predicting whether someone will pursue a new position or not. Also, these information are so personal that headhunter can never get such information. Thus, we start our data preprocessing with columns selection. We remove the irrelevant columns as well as the columns with arbitrary/unjustifiable answers. After selection process in columns, we end up with 26 columns which are mostly categorical variable as well as our dependent variable 'JobSearchStatus'.

Apart from feature selection, most of the columns have fragmented categories which can make our model highly biased. For example, in the column "business discipline", we have accounting, finance, marketing and some other categories that are not in the computer science area. Also, the total number of these categories is far less than the computer science category. Thus, we need to group such categories into one to make the distribution more balanced , for example, some company size and age range can be binded into one category. Some of the columns also allow the respondents to choose more than 1 choice so we need to split each answer and label each of them as one categorical variable.

2. Dealing with missing value

First, we remove the columns that has too many null values (>50% of data) and columns that has no structured response.

Moreover, some of the columns are mostly blank responses or has no restriction on the answer so they are difficult to categorize.

After all the preprocessing, there still exist null values in most of the columns. We have dealt with each of them in different ways according to the original intention of the survey form. For example, in the question that provide "None" kind of answer, we treat null value of that columns to has all zero dummy values. But the question that has "Others" kind of answer (the data treat them as null) we try to assign them in to the most common variable. However, in some of the columns that has very high proportion of null value (more than 15% of the remaining rows) we decided to remove those null rows to prevent the bias from null value filling.

Since we've already chosen 'JobSearchStatus' as our dependent variable, we need to drop some rows with None values in that column.

3. Creating dummy variables

Since most of our features are categorical variables, we need to create dummy variables for them, including country, gender, employment status, undergraduate major and other features. For each categorical variable, we dropped the first dummy variable generated from it to eliminate multicollinearity

4. Transform dependent variable

Given 'I'm not actively looking, but I am open to new opportunities' represents neutral opinion, we need strongly opposite attitudes as dependent variable to predict whether this person wants to seek jobs. Thus we dropped some rows with ambiguous answers in this column and finally get a dataset of 22,152 observations and 185 independent variables.

Modeling and Results

Since our dependent variable has some imbalanced issue(0 label is 63% of total data), so we select our model performance matrix to the ROC_AUC value.

1. Logistic Regression with Lasso (Ridge)

In Logistics regression we use 10-fold cross-validation with ridge and lasso penalty to do our features selection. In the process of interpreting the model coefficients, we have found that, for both model, the most important features are the employment status, years of coding, country of living, age, and role of developers. The models suggest that people who never have a job are not likely to find a job, but the self, none or part-time-employed are actively looking for it. Also people who has been coding for more than 6 years, who are in executive levels, and who just changed a new job last year are less likely to seek for a new job.

Interestingly, the position of Data Analyst is likely to find new job opportunity, which may imply high demand in the market for this position.

2. Random Forest

In this part, we turn to random forest to lower variance. For the parameter tuning in random forest, we tune the following four features to improve the predictive power of our model: n_estimators, max_depth, min_samples_split, min_samples_leaflist. After tuning these parameters of the estimator by cross-validated grid-search over a parameter grid, we choose max_depth= 14, min_samples_leaf = 100, min_samples_split= 200, n_estimators=150, which is reasonable in every aspects. Although the variance decreases, our AUC score decreases a lot, which is not our final goal. Thus, we move on to the next part.

3. XGBoosting Classifier

In the last part of machine learning models, we include the XGBoosting to finally optimize our result. The reason why we want to include XGBoosting is that it can (1) help to reduce overfitting; (2) handle missing data; (3) process parallelly; (4) process built-in cross validation; (5) have high flexibility. Sounds great! As usual, we start from parameter tuning.

There are three sets of parameters:

(a)General parameters:

For the booster -- we use the default value gbtree; silent = 1, such that we can see the running process of our model.

(b)Booster parameters:

For the max_depth, min_child_weight, our results are 4 and 3 separately, which make sense since max_depth is between and small min_child_weight is chosen because it is a imbalanced class problem and leaf nodes can have smaller size groups.

For the gamma, we first choose a list of five values and later dig into smaller range of values. At last, we choose 0.2, which quite makes sense because the larger gamma is, the more conservative the algorithm will be.

For the subsample and colsample_bytree, we both start from 0.5 to 1.0. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. colsample_bytree is the subsample ratio of columns when constructing each tree. At last, we choose 0.85 and 0.75 separately.

For the regularization parameter, we decide to leave it out after trying different combinations without affecting the final result.

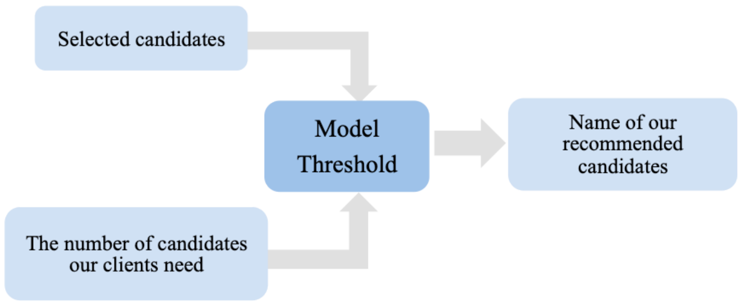
Lastly, we should lower the learning rate and add more trees. Let's use the cv function of XGBoost to do the job again. The final combination shows the highest AUC score.

Model	Train Cross Validation accuracy	Test Accuracy	Train Cross Validation AUC	Test AUC

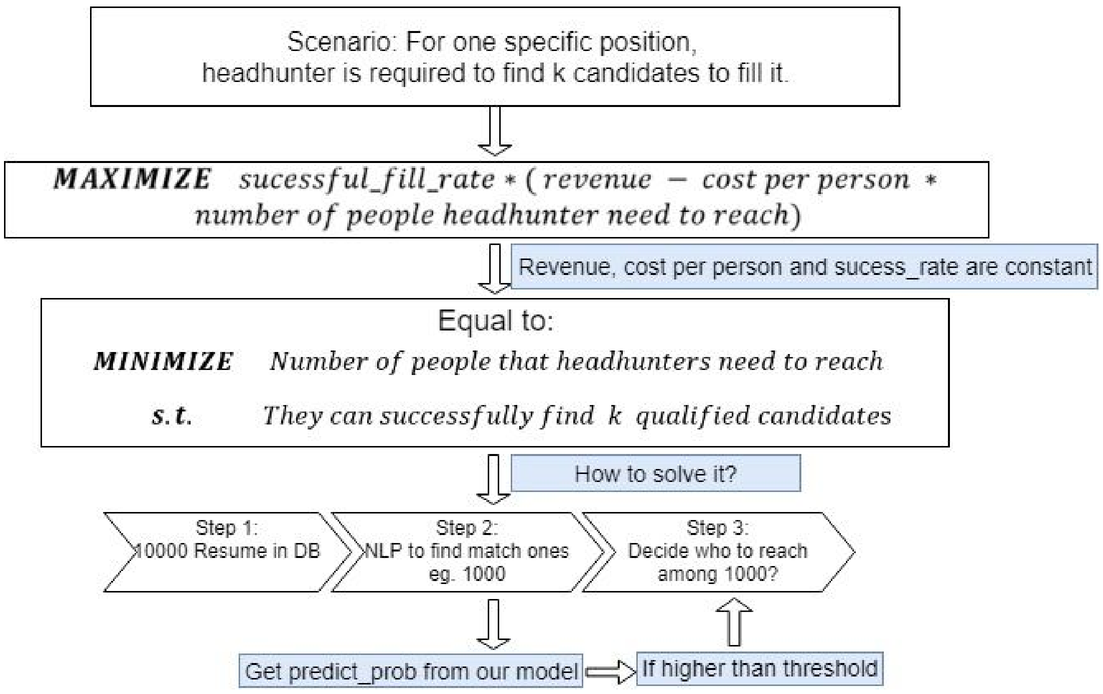
Logistics-Lasso	0.8327	0.8308	0.8985	0.8896
Logistics-Ridge	0.8327	0.8315	0.8988	0.8893
Decision Tree	0.8110	0.8151	0.8714	0.8694
Random Forest	0.7934	0.7940	0.8803	0.8796
XGBoost	0.8401	0.8366	0.9093	0.9054

Application and Business Value

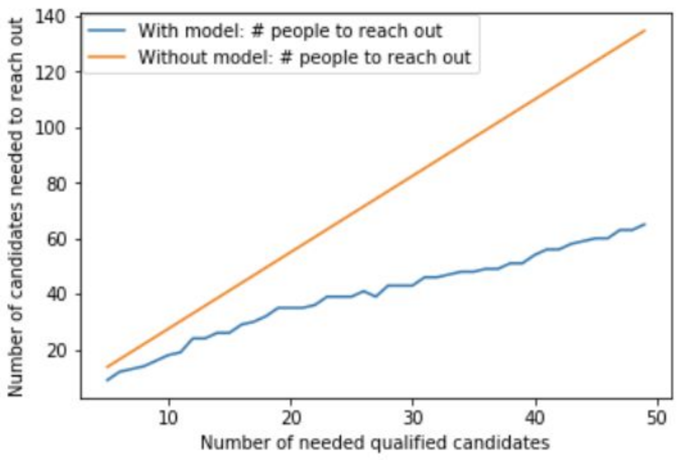
The task of a headhunting company is finding a certain number of candidates (take this number as X) who are a match with the position and also want to find a new job for a certain position. So when they get a new task, they first select the candidates who are a match with the position from available candidates data to get a dataset A. Then they train our machine learning model on dataset A to get the probability of each one wanting to find a job. We assume there is an historical dataset B which contains the candidates information and also if they want to find a new job. Then using our function find_threshold on dataset B, they get the best probability threshold to minimize the people they need to reach to in order to satisfy the number X. After getting the threshold, they apply it on their results from dataset A and return the names of the candidates. They then provide these candidates names and information to the client company.


Selected candidates
Model
Threshold
Name of our recommended candidates
The number of candidates our clients need

Our optimization process is as follows. For the threshold part, we set our threshold based on historical data with label y. Firstly, we bootstrap sample data with the same size as number of selected possible candidates(eg.1000). Then, for each iteration, we find threshold which enables the number of true possible people equal to k and we use the mean of thresholds as our final threshold.


Scenario: For one specific position, headhunter is required to find k candidates to fill it.
MAXIMIZE sucessful_fill_rate * (revenue − cost per person * number of people headhunter need to reach)
Revenue, cost per person and sucess_rate are constant
Equal to:
MINIMIZE Number of people that headhunters need to reach
s.t. They can successfully find k qualified candidates
How to solve it?
Step 1: 10000 Resume in DB
Step 2: NLP to find match ones eg. 1000
Step 3: Decide who to reach among 1000?
Get predict_prob from our model
If higher than threshold

The graph is comparing the number people the headhunters need to reach out (y-axis) in order to get certain number of people they are required to (x-axis) with and without using our model. With our model we can see that the growth rate of people that need to be reached out is significantly flatter than the random selection technique (without model). With this result of reducing the number of people they need to reach to by more than half, it will dramatically reduce the cost and time of the headhunting company and significantly increase total profits.



Summary

Although we've established an excellent model with auc of 0.9093 and successfully discovered its business value, there still exists some space for us to explore. We can go further with more available data, and also more features - for example, the cities our developers live in, the industry our developers contributes to and their salaries. Maybe we can keep our model updated with these further data in the future maintenance. Moreover, to make our model more user-friendly, we can build a mobile application based on it, so non-technical people can use it too.