**Project: Stack Overflow Q&A Analysis**

Thanks to all you real statisticians for sticking with us to the end in these presentations. I'm assuming most of you are already heavy users of Stack Overflow. But for those of you that have little idea of what Stack Overflow is, [picture] "Stack Overflow is like Quora for programmers. It’s the largest online community for programmers to learn, share their knowledge, and advance their careers." As future data scientists, we can make good use of this website whenever we program in languages such as R and Python. Therefore, to help us better understand this community, let's ask the following questions:

* First, what are the differences between R and Python questions, except for R is R and Python is Python?
* Next, how can we ask ‘good’ questions and give ‘good’ answers in order to get a higher score (higher votes)?
* And finally, can we create a system to automatically generate tags in StackOverflow according to the context of the questions?

**(Data source)**

We explored two datasets provided by Stack Overflow on Kaggle. One is for R Q&A and the other is for Python Q&A, which are the top 2 most popular programming languages that data scientists use.

Each dataset is organized as three tables:

1. Questions, each containing the title, body, creation date, score(highest vote), and owner ID..
2. Answers, which contains the body, creation date, score, and owner ID for each of the answers to these questions. Each answer is linked to its parent question in the Questions table by ParentID.
3. Tags, containing the tags of each question.

We only consider data is from Sep 16, 2008 to Oct. 19, 2016.

**(Now let’s look at the first question: Comparison between R & Python Questions, which is also EDA)**

In this part, we try to analyze the differences between R and Python questions.

1. How did Python and R's popularity develope over the last 8 years on stackover flow?

2. What are the hottest topics for the two languages respectively?

3. How quickly are the questions answered for these two languages?

4. Scores you can get as being active on the website?

Question 1: How did Python and R's popularity develop over the last 8 years on StackOverflow?

[picture] This is what the changes for the amount of questions of python and R look like from 2008 to 2016. It shows that both kept growing at a high speed.

Question 2: What’s the hottest topics for the two languages respectively?

From the word cloud, we can see the main areas of focus for R questions are ggplot2, dataframe and shiny. Concentrating on data visualization and report delivery. statistics, regression are also common topics

For Python, django, numpy, pandas, and matplotlib are the hottest.

3: How long does it take until first response appear after a question be posted?

For R, the mean time is 48 days, median time is 0.032 days, which is about 45 minutes. While, for Python, the mean time is 70 days, the median is 0.02 days, which is less than half an hour. Since the distribution of first response time is not normal, we use Wilcoxon rank sum test and the p-value is much small than .01, therefore the time for R and Python are significantly different, which matches with intuition.

This means if you ask a question about Python on this website, you may receive answers sooner and more frequently than asking a question about R.

4: Scores you can get as being active on the website?

On a Q&A website, if you are an expert in programming languages, there might be some tricks for you to get higher scores.

By exploring the data, we found an obvious linear relationship between the question’s score and its corresponding highest rated answer’s score.

We got two linear regression models here.

For R, the expected highest rated answer’s score = 1.01+ 0.93\* question’s score

For Python, highest rated answer’s score= 0.80+1.11\* question’s score

It indicates positive relationship between highest that you can get higher score by answering popular questions. Also, get relatively higher score by answering a question about Python. Since the coefficient for Python is 1.11, higher than that for R.

**(Moving on to the next question: Score & Time Analysis)**

(Background)

When we ask questions on stackover flow, we all hope that our problems can be seen by more people so it can be solved sooner. But have you ever thought about how to make this happen? To make our questions draw more attention?

By looking at the two Stack Overflow datasets, we are interested in two responses: the score of the question and the time that it takes to get the question resolved.

From the histograms, we can see that two variables of interest don’t follow the normal distribution. Both distributions are highly skewed. Based on that, we define that a question gets solved when there is a corresponding answer that’s upvoted by at least 3 people. This is the 3rd quantile of the score distribution. Therefore, we can label each question as 0 or 1; 0 being a question without good answers and 1 being a question with good answers.

Due to the departure from normality, we appeal to non-parametric method, powerful xgboost Model.

First, we extract some features from the raw dataset.

1. tag\_count: the number of tags for each question

2. sum\_tag\_freq: the sum of freq rates of all tags for each question.(occurrence of tag over # of questions)

3. max\_tag\_freq: the maximum freq rate of tags for each question.

4. body\_length, title\_length, code\_length, comment\_length: text length of respectively question part.

5. Title\_word\_count, body\_word\_count: how many words included in respectively question part.

6. code\_blocks\_count: the number of code chucks included

7. code\_comments\_count: the number of lines of code comments included

8. Url\_count: # of web links included

9. img\_count: # of images included

Then, we train the model on these features. And after parameter adjustment, we got a cross-validated MSE for the best model of R Questions as nearly 8.2 and MSE for python questions as 19.1. From the importance matrix, we can see that ,for R questions, sum\_tag\_freq max\_tag\_freq code\_length are most relevant to the question score. And there is a gap of information gains between features in cluster 1 and features in cluster 2. As for Python questions, the most important factors change. Title length is inserted into the front row.

Now, we have a model to predict the question score. However, our ultimate goal is to predict whether we will get good answers. We have to validate the relationship btw question score and its label, whether the question is resolved. Naturally, we applied a logistic regression to predict question’s label. It shows that if the question score gets higher, the question will be more likely to get solved. The logistic regression model has an accuracy rate, about 80%, on the validation set.

Above all, we are confident that the outcome of a question depends on the question’s score, and as we built a model to predict question scores, we can infer that some text features are relevant to the question’s outcome. Our suggestion is to try to tag your questions with some popular and regularized tags and control your codes’ length and title’s length included in your question. Popular tags and brief description will make you a stack Overflow star and you don’t have to worry about your programming homework anymore.

**(Tag Recommendation System)**

In the last part of our project, we asked the question can we create an a system to automatically generate tags in StackOverflow. Because when people ask questions on StackOverflow, they have to create tags by themselves, and sometimes they don’t really know what tags they should choose. So here we develop a tag recommendation system to help them out by implementing LDA topic modeling and calculating weighted probability of tags of the closest training documents within the given topic.

First of all, I will introduce a little bit about LDA topic modeling, which is a Bayesian clustering technique widely used for inferring hidden ‘topics’ in document corpora. The basic idea is that each document can be represented as a mixture of K topics, and every topic has a discrete distribution over words. By fitting LDA model on the body text of 80% R questions as training set, and here we tuned K = 50, we got document distributions of 50 topics and word distribution of each topic. We can get a sense of what these 50 topics look like by looking at this graph. On the left hand side, each circle represents a certain topic. The size of the circle represents the number of documents in that topic. If you click one of the circles, i.e. topics, on the right hand side, you will see the top 30 most prominent terms in that topic, which would give you a sense of what that topic is about. For example, (hover over topic 3) if we click topic 3, we’ll see this topic is mainly about plotting, since we have ‘ggplot’ and all those options here.

After fitting the LDA model, we can predict the topic that a new document belongs to. Let’s take a cleaned question in the test set for example. It is predicted to belong to Topic 15. After we know its topic, we can shrink our search range from all 140K+ documents to an average of 3K documents of the same topic, which would make our recommendation engine less time-costing and memory-consuming for our client. In our example, there are 5K documents in that topic, which is still too many. Therefore, we want to find 20 questions that are the most similar to the untagged question. The similarity of two questions we chose here is Jaccard Index, which is the ratio of the length of their intersection set to the length of their union set. After we pick out the 20 most similar documents to the untagged question, we count their tag frequencies and sort tags by descending order. By doing this, we get a list of tags we have confidence to recommend. But we didn’t stop there. We thought it would make more sense if the recommended tags also appeared in the question body. So we decided to give ‘bonuses’ to the tags that appeared in the question body. In this example, the tags marked blue here appeared in the question body, so we increased their frequencies by 10 which is tuned, and then sorted the tags again. We got the next table, and finally, we recommended the top 6 tags to this question.

We have repeated this process for the whole test set, and the accuracy of recommending correct tags is 71%, which we think is pretty good.

That wraps up our presentation. Thank you for your attention. If you have any questions, please feel free to ask us now.