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

Thanks to all you real statisticians for sticking with us to the end of these presentations. I'm assuming most of you are already heavy dependent users of Stack Overflow. So are we. For those of you who 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 webiste when programming. Therefore, to help us better understand this community, let's ask the following questions:

* First, what are the differences between R and Python questions in terms of popularity, hot topics, etc, since these two are the top 2 most popular programming languages that data scientists use?
* Second, how do we ask questions that are more likely to receive acceptable answers?
* And finally, can we create a system to automatically generate tags in StackOverflow according to the context of the questions?

**(Data source)**

To answer these questions, we explored two datasets provided by Stack Overflow on Kaggle. One is for R Q&A and the other is for Python Q&A,

Each dataset is organized as three tables:

1. Questions, which contains the title, body, creation date, score, and owner ID.
2. Answers, which contains the body, creation date, score, and owner ID for each of the answers. And each answer is linked to its parent question by ParentID.
3. Tags, which contains the tags of each question.

Notice that The time range of our data is from 2008 to, 2016.

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

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

First let’s look at how R and Python's popularity develop over the last 8 years on StackOverflow?

[picture] This is how the number of questions of R and Python changed from 2008 to 2016, both kept growing at a high speed, but Python questions overall are much more than R questions.

Next, what are the hottest topics for the two languages respectively?

From the word clouds, we can see the main areas of focus for R are ggplot2, dataframe and shiny, concentrating on data visualization and data structure functions.

For Python, Django, Numpy, Pandas,and Matplotlib are the hottest topics.

Then, how long does it take until first answer appear after a question being posted?

For R, the mean time is 48 days; median time is about 45 minutes. While, for Python, the mean time is 70 days, the median is less than half an hour. We use Wilcoxon rank sum test to test their difference, and the p-value is much small than .01, therefore the time for R and Python are significantly different, which matches our 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.

Last but not least, is there any relationship between question’s score and its answer’s score?

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

By fitting robust linear regression, we got two models for R and Python respectively.

The models both indicate you can get higher score by answering popular questions. Also, you can get relatively higher score by answering a question about Python.

**(Moving on to the next question: we want to predict question scores based on some features)**

From part 1’s conclusion, we all hope that our problems can get higher score so that it can be resolved better. But how to make it happen?

We are interested in two response variables: the score of the question and whether a question gets resolved.

From the histograms, we can see that the scores of questions are highly skewed. Based on that, we define that a question gets resolved when there is a corresponding answer with at least 3 scores, which is the 3rd quantile of the answer score distribution. Therefore, we can label each question as 0 or 1; 0 being a question without good asnwers and 1 being a question with good answers.

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.

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

First, we extract some features from the raw dataset.

Then, we train the model on these features. After tuning parameter, 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 the model to predict the question score.

Above all, we are confident that whether a question is solved or not 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 short included in your question. Popular tags and brief description will make you a stack Overflow star and you don’t need to worry about your programming homework anymore.

**(Tag Recommendation System)**

In the last part of our project, we asked the question can we develop 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, which is a Bayesian clustering technique widely used for inferring hidden ‘topics’. 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 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 5K documents of the same topic, which would make our recommendation engine less time-costing and memory-consuming for our users. But 5K documents are 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 was 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 73%, which is pretty good.

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