Bug Prediction in Windows Azure

A machine learning way to compute big data in a cloud platform

By: Wei Liu, Milind

Abstract -- The datasets nowadays are growing in an exponential way. Both the volume and the form of data are so huge that it's hard to use a single client computer processing it. To dealing with this problem, cloud computing support a breakthrough method about processing data in the cloud use high speed supercomputers, which is way much faster than a personal computer. Microsoft Azure is a cloud computing platform and infrastructure created by Microsoft for building, deploying, and managing applications and services through a global network of Microsoft-managed data centers. In this report, we are going to talk about the characteristics about Windows Azure and how we could use it to some hand on big data projects.

Index Terms -- Windows Azure, big data, machine learning, bug prediction.

I. Introduction

Over the decades developing of computer science, the dataset of computer grown from simple numbers in mathematical questions to the music, video, documents files which count billion times bigger. Which more is, for example, the dataset of users in a hospital, insurance records for a company, versions of code in our repository. The data growing in an exponential way. Traditional tools are hard to store, manage, process nor analyze.

The born of Windows Azure, a cloud computing platform, is one good solution to address this problem. It is a model for enabling ubiquitous, on-demand access to a shared pool of configurable computing resources, which can be rapidly provisioned and released with minimal management effort.

We want to try using Windows Azure platform to do some hands on big data project. And the bug prediction is a good choice in this case.

A. Machine Learning

Machine learning uses computers to run predictive models that learn from existing data in order to forecast future behaviors, outcomes, and trends.

These forecasts or predictions from machine learning can make apps and devices smarter. When you shop online, machine learning helps recommend other products you might like based on what you've purchased. When your credit card is swiped, machine learning compares the transaction to a database of transactions and helps the bank do fraud detection.

B. Windows Azure

Azure Machine Learning is a powerful cloud-based predictive analytics service that makes it possible to quickly create and deploy predictive models as analytics solutions.

C. Bug Prediction

Predictive analytics uses various statistical techniques - in this case, machine learning - to analyze collected or current data for patterns or trends in order to forecast future events. Google bug prediction, which we are going to talk about in this paper is one concise and effective way to prevision.

In this report, we are going to talk about the basic characteristics about Windows Azure in part II. And we will going to take a look at the hand on problem we want to address in Window Azure, which is bug prediction method by Google, in part III. We will try to implement the bug prediction method in Windows Azure and get some interesting feature about the results using machine learning in part IV. Then, we will check our experiments results in part V. And finally we will get our conclusion in part VI.

II. Windows Azure

Azure Machine Learning is a powerful cloud-based predictive analytics service that makes it possible to quickly create and deploy predictive models as analytics solutions.

Azure Machine Learning not only provides tools to model predictive analytics, but also provides a fully-managed service you can use to deploy your predictive models as ready-to-consume web services. Azure Machine Learning provides tools for creating complete predictive analytics solutions in the cloud: Quickly create, test, operationalize, and manage predictive models. You do not need to buy any hardware nor manually manage virtual machines.

To develop a predictive analysis model, you typically use data from one or more sources, transform and analyze that data through various data manipulation and statistical functions, and generate a set of results. Developing a model like this is an iterative process. As you modify the various functions and their parameters, your results converge until you are satisfied that you have a trained, effective model.

Azure Machine Learning Studio gives you an interactive, visual workspace to easily build, test, and iterate on a predictive analysis model. You drag-and-drop datasets and analysis modules onto an interactive canvas, connecting them together to form an experiment, which you run in Machine Learning Studio. To iterate on your model design, you edit the experiment, save a copy if desired, and run it again. When you're ready, you can convert your training experiment to a predictive experiment, and then publish it as a web service so that your model can be accessed by others.

There is no programming required, just visually connecting datasets and modules to construct your predictive analysis model.

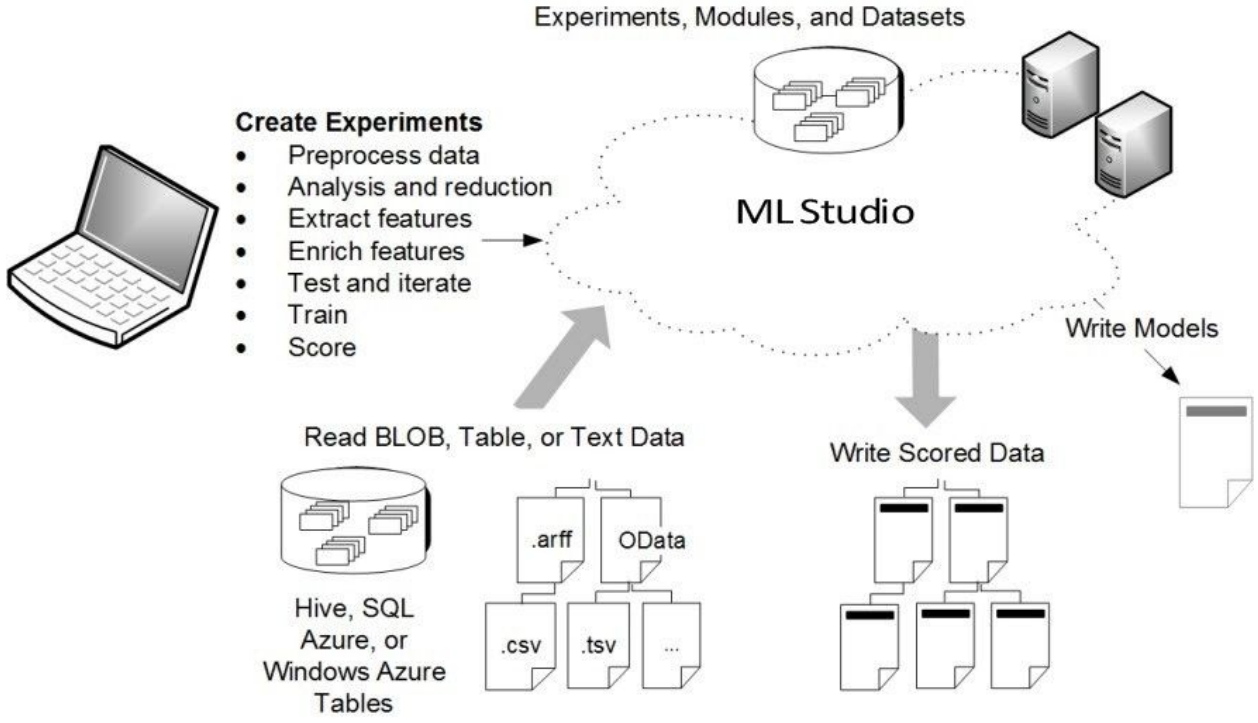


Figure 1. Host Architecture of Windows Azure

Microsoft Azure Machine Learning contains many powerful machine learning and data manipulation modules. Analytics and data manipulation in Azure Machine Learning can be extended by using R. This combination provides the scalability and ease of deployment of Azure Machine Learning with the flexibility and deep analytics of R.

Azure Machine Learning Studio also supports embedding Python scripts into

various parts of a machine learning experiment and also seamlessly publishing them as scalable, operationalized web services on Microsoft Azure.

III. Bug Prediction By Google

A. Problems

No matter in a company as Google or a lab in our school UC Merced, we have a lot of engineers and students working on code every day. In fact, 50% of the code base in Google changes every month. That's a lot of people and a lot of code. Testing and debugging could take innumerable time. Even we work every day on increasingly more complex problems, providing the features and availability that our user depend on. Some of these problems are necessarily difficult to grapple with, leading to find the bug file be unavoidably difficult.

For the sake of bug prediction method by Google, we will call the continuously trouble file "hot spots". And the goal of this method is to find the "hot spots" in a project so we could do some necessary work accordingly, like warning the coder to pay more attention or hand in the code to a superviser. Perhaps a hot spot is resistant to unit testing, or maybe a very specific set of conditions can lead the code to fail. Usually, our diligent, experienced, and fearless code reviewers are able to spot any issues and resolve them.

This bug prediction by Google is a machine-learning and statistical analysis method trying to guess whether a piece of code is potentially buggy or not, said to find whether the code is a "hot spot". Unlike the common bug prediction method which take source-based characteristics, like how many lines of code, how many dependencies are required and whether those dependencies are cyclic, as the metrics. These can work well, but these metrics are going to flag our necessarily difficult, but otherwise innocuous code. However, for our hot spots, we actually have a great, authoritative record of where code has been requiring fixes: our bug tracker and our source control commit log. And the source history from version control system, like Github, SVN, works very well to predicting bugs. So, this method just focus on the commits logs in our version controller.

B. How it works

In this method, Rahman et al. found that a very cheap algorithm actually performs almost as well as some very expensive bug-prediction algorithms. Simply ranking files by the number of times they've been changed with a bug-fixing commit (i.e. a commit which fixes a bug) will find the hot spots in a code base. This matches our

intuition: if a file keeps requiring bug-fixes, it must be a hotspot because developers are clearly struggling with it.

We implemented the Rahman algorithm by filtering out all the commits with bug fixing. Then, for each file in one single commit, we calculate the “hot spot score” for this file and record it. After getting all the scoring done, we accumulate all the scores for a single file. The sum score for that file is the final “hot spot score”. From a sorted file-score table we could easily tell if a file is more intend to be a hotspot because of its higher score.

Is there any improvement to Rahman’s algorithm? Yes. It turns out that while the Rahman algorithm shows us where hot spots are, it doesn't adapt to changes readily. If a development team manages to nail down a hot spot and get it fixed, it'll still appear in the list because of all the bug-fixing commits it created in the past. So, we involved another factor as our key metric for our algorithm: the time ratio. We scale the lifetime of a project as 1, and set the time ratio for a file according to when this commit submitted in the lifetime.

So, the final algorithm for computing the hotspot score of a file is:

$$Score = \sum_{i=0}^{n} \frac{1}{1+e^{-12ti+12}}$$

Figure 1. Hot spot score of a file

Where n is the number of bug-fixing commits, and ti is the timestamp of the bug-fixing commit represented by i.

If we plot our equation, it looks like this:

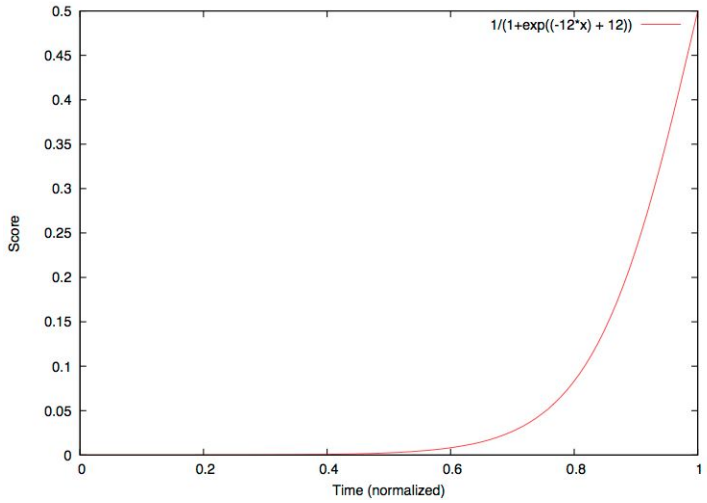


Figure 1. Plot chart of hotspot score algorithm

In this way, we used a machine-learning and statistical analysis to predict the potential chance a certain file will appear bug in the future.

IV. Implement Machine Learning in Windows Azure

It seems Windows Azure could not deal with bug prediction task, because Windows Azure originally doesn't support string matching (which use to find fixed bug related commits) or complex mathematical computation (algorithm to get time ratio and hotspot score). However, this platform support a powerful computation language -- Python. Python is easy use language which has a lot of useful libraries helping implementing variety tasks. Let's take a look at implementing this project about how we use python to get the raw data, how we use the methodology to get hotspots, and more, how we use some machine learning method in WIndows Azure to get some interesting feature from the results from the project.

A. Getting raw database using python

The first step about our project is read the information about commits in a version control system (like github). Thanks for the big volume of hand on libraries in python, we could easily use several current library to read all the content.

Vcs_abstraction is library used to read all the commits information in github, SVN or other version control system. The logic to use is pretty easy: specifics the path of the repository root, read all commits information in the repository, and store into a dictionary in memory.

```
def get_current_vcs(path="."):
    for vcs_type in vcs_abstraction.get_registered_vcs_types():
        vcs = vcs_abstraction.get_vcs(vcs_type)
        if vcs.static_detect_presence(path):
            return vcs(path)
    raise Exception("Not found a valid VCS repository")
```

Table I, get commits' information using Vcs_abstraction

CSV is another useful library which support user to write data table into a csv file, which is the format we could manipulate in machine learning and upload to Windows Azure as raw data. So, after getting the dictionary storing all the raw data, we simply use this library to write the data to a csv file.

```
with open('hotspot_raw.csv', 'w+') as csvfile:
    fieldnames = ['date', 'email', 'message', 'id', 'author', 'file']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    for cmt in cmts:
        for file in cmt['files']:
            writer.writerow({'date': cmt['date'], 'email': cmt['email'].encode('utf8'), \
                'message': cmt['message'].encode('utf8'), 'id': cmt['id'], \
                'author': cmt['author'].encode('utf8'), 'file': file.encode('utf8')})
```

Table II, write commits information into csv file using CSV

B. Steps to get hotspot score

What previous step does is getting the raw dataset in our local device. Then, we could upload the raw data into our Windows Azure platform and do our machine learning methodology to it. In order to get the hot spot score for every file, we follow the four steps in a row in our platform show below:

1. Select Columns

The size of raw data could be tens or hundreds Megabytes and contains much more information, like email, we don't need to use. In order to shorten the runtime in our project, we only selectively use the columns relative to our algorithm. Says the name of the file, the submission date of the commits, and the commit message.

This step could easily implement by nested Windows Azure function module -- Project Columns Selection. We input the raw data which uploaded in section A, specifics the three columns we want to use and get the output is the sub-table we want to compute.

2. Filter Bug Fixing Related Commits

The next step is filtering all the commit message in our dataset. In this project, we use regular expression Regex to match the message which contains a substring "bug", "fixed", "closed". Because a commit message which contain these substring has a high chance being a bug fixing commit. And also, it is a coder customer to include these words in a bug fixing commit.

```
description_regex = re.compile(
    "^.*([B|b]ug)s?|([f|F]ix(es|ed)?|[c|C]lose(s|d)?).*$")
```

Table III, regex used to filtering bug fixing related message

3. Get Time Ratio

After getting the data table purely with bug fixing related data, we need to calculate the time ratio of every file. We take the time scale from the first submission date to the current time as 1, and define the time ratio as the percentage from the first submission time to that submission time of the given file by take second as time resolution. For example, if a project is scale from January to December as right now, and the submission date of a given file is in the middle, says June, we regards its time ratio is 0.5.

Getting time ratio by python is simply apply a mathematical expression:

```
def get_time_ratio(base, now, time):
    return 1 - (now - time) / (now -  base)
```

Table IV, python implementation of getting time ratio

4. Get Hotspot Score

Having the time ratio for all the file, getting the hotspot score is just a piece of cake. For every file in every commit, we get the hotspot score, and sum them up.

```
def get_hotspot(time_ratio):
    return 1/(1+math.exp((-12 * time_ratio) + 12))
```

Table V, python implementation of getting hotspot score

5. Architecture in Windows Azure

Follow by the four steps above, the architecture of workflow chart is shown below. The blocks represent as project columns select, filtering bug fixing relative message, get time ratio, get hotspot score.

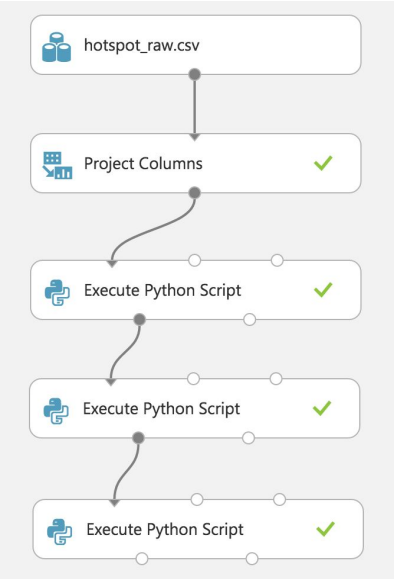


Figure III, architecture of workflow in Windows Azure

C. A little machine learning to get interesting features

If the steps work appropriately, we could get the hotspot score for every file, and the higher score for a file indicate it more potentially have bugs in the future. Projects seems to have a happy ending here. However, we are using a very advanced powerful machine learning platform, we could do much more than just get some statistics we could get just by python.

Talking about machine learning, regression in machine learning in a way to find some relationships between metrics. And right now we have file name, author, commit messages as well as the hotspot score for every file, we could implement some machine learning regression into them and try to find out some relationship between each others.

In this case, we put all the hotspot information back to the original bug fixing related table. And then, we use the nested machine learning API in Windows Azure to train a linear regression model which set L2 regularization weight as 0.001, set Random number seed as 0, train the model taking hotspot as object. Right before input data into the model, we need to split the data into two part, which part one is used for regular machine learning model train while another part is used to evaluate the result. So, model evaluation could be invoked to check how specifically correct the machine learning train is.

By doing this linear regression, we could simply find the most contributed coder by linear regression to hotspot score and commit username; find the most bothering feature in message by linear regression to hotspot score and commit message.

The workflow of whole method in Windows Azure represents as below:

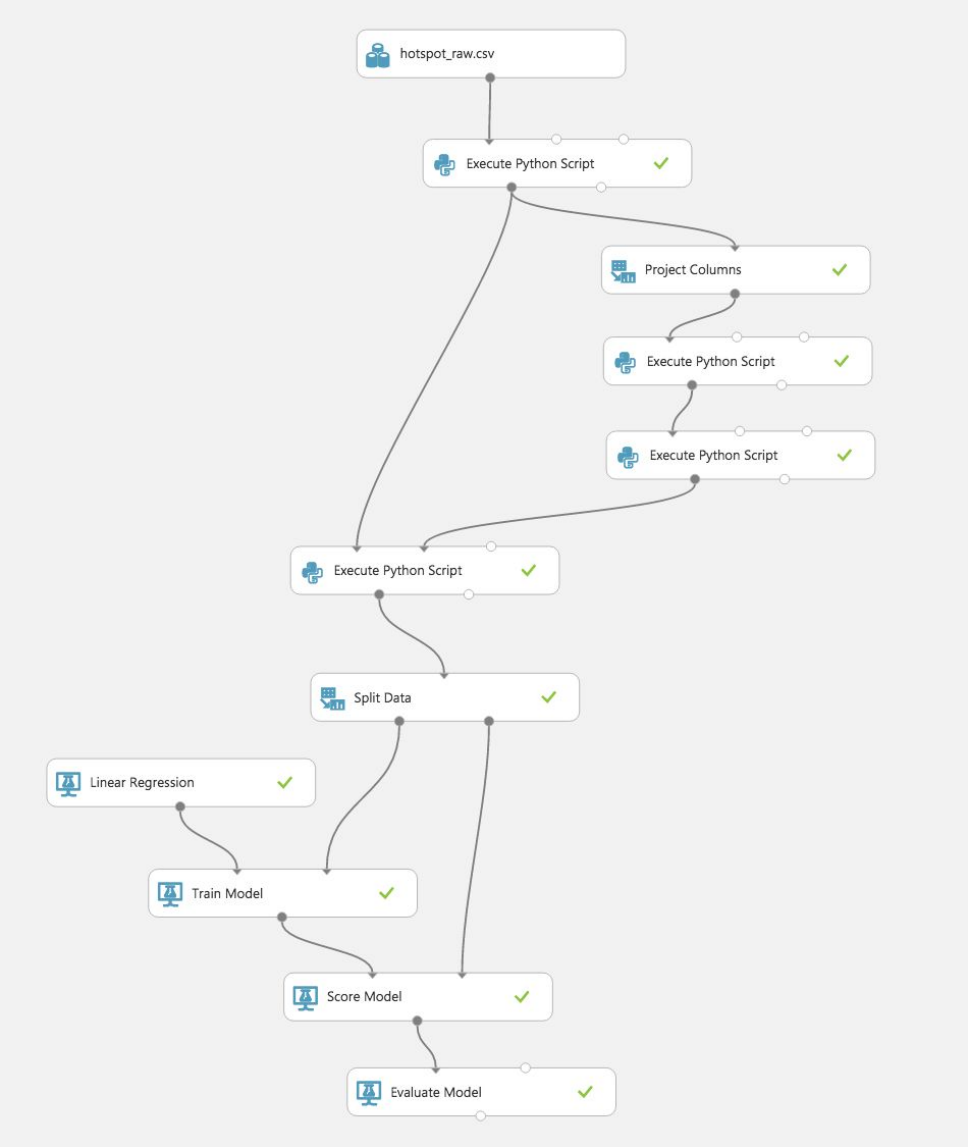


Figure IV, apply linear regression to get the relationship between hotspot score and metrics

D. Parallel computing in big data

Regards to huge projects, such as Linux kernel, which have million commits over years, processing this big data will consume a lot of time in one step. Because Windows Azure run different branches in a project parallelly, it's become a bottleneck for the whole project. In this case, we could split this big data into several small piece so that platform could process them parallelly.

Also, we could upload several different commit raw datasets in one single project and process them parallelly. Then using machine learning methodology to get some relationship between each others.

To implement this idea, we specify different branches in Window Azure to represent different dataset. And Windows Azure take parallel computing to them automatically:

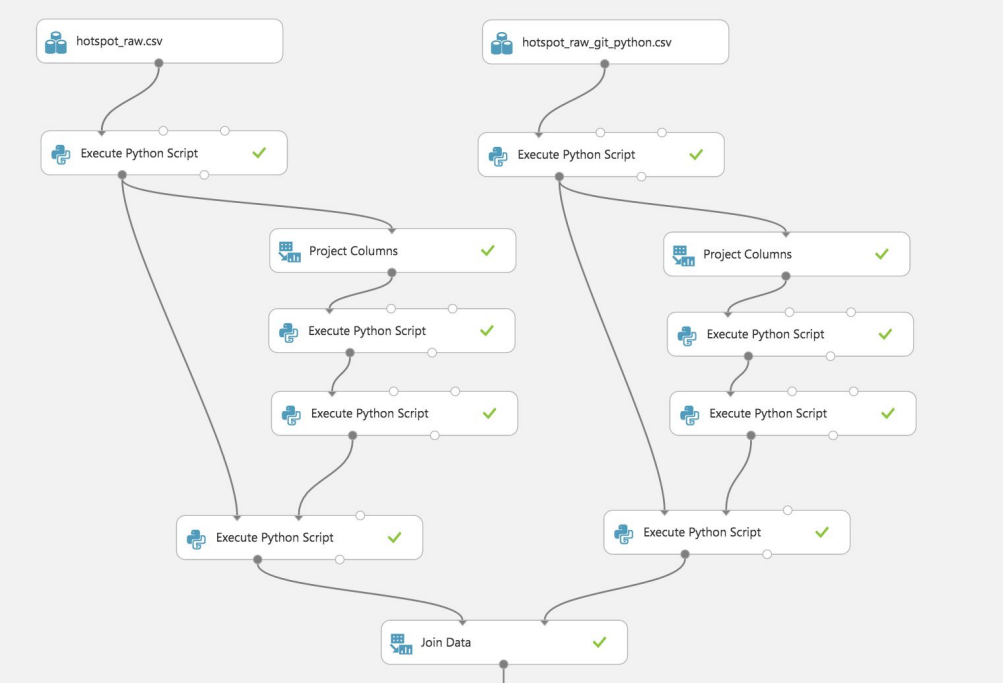


Figure V, use different branches in Windows Azure to process them parallely

E. Critical Checking

Here's a question, now we have all the results from experiment, how accurate is it? Is it really indicate the real buggy file in the future?

In order the check the accuracy, we plan to split the original data file into two part right at a time point defined by user. We go study all the feature we talk about in the paper using the first part of the data, means the data previous the time point, while use the second part of the data, the data after the time point, to check how accurate it is.

But here's a lot of data, manually check them one line by one line would be so time consuming, we need a mathematical way to deal with them. Windows Azure support a model evaluation tool is right to the need. We could input the linear regression result as the sample and use the second part of data to check the accuracy. The result could show the error rate and bias to indict how accurate it is.

V. Experiments Result

For this experiment about bug prediction in Windows Azure, we need to find some really large project which could represent big data concept. However, due to limit of free account and function ability of our experimental computer, we compromised to

choose WordPress, which is a open source web blog building project, as our experimental sample.

A. Get Raw Dataset

The project repository of WordPress size about 180 Megabytes with 27 branches and about 34,170 commits from start till now. Every commits may involve one or more than one relative files. The first step is read all commits infomation out from github database. We use the python code we introduced in part IV section A to digest the information we need and store into a csv data file.

Partial result shows below, which is a table contain 90,666 commits rows with commit date, email, message, id, author and file name. The whole dataset in 20.8 Megabyte.

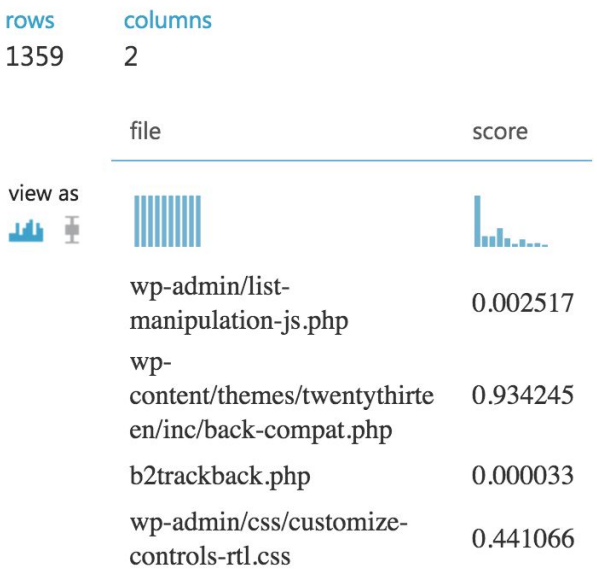
	date	email	message	id	author	file
2	2015-12-18 18:38	boonebgorge	Add `current-cat-ancestor	31db92d83e	Boone Gorge	wp-includes/class-walker-category.php
3	2015-12-18 18:38	boonebgorge	Add `current-cat-ancestor	31db92d83e	Boone Gorge	wp-includes/version.php
4	2015-12-18 18:16	aaron@jorb.i	Remove RDIO from oEmb	0eaedf734fd	Aaron Jorbin	wp-includes/class-oembed.php
5	2015-12-18 18:16	aaron@jorb.i	Remove RDIO from oEmb	0eaedf734fd	Aaron Jorbin	wp-includes/version.php
6	2015-12-18 18:12	boonebgorge	Ensure that `wp_list_cate	b2f4f736338	Boone Gorge	wp-includes/category-template.php
7	2015-12-18 18:12	boonebgorge	Ensure that `wp_list_cate	b2f4f736338	Boone Gorge	wp-includes/version.php
8	2015-12-18 17:44	boonebgorge	Ensure `get_terms()` resul	5d46c03508	Boone Gorge	wp-includes/taxonomy.php
9	2015-12-18 17:44	boonebgorge	Ensure `get_terms()` resul	5d46c03508	Boone Gorge	wp-includes/version.php
10	2015-12-18 17:23	sergeybiryukc	Docs: Fix copy/paste error	3f35196e48f	Sergey Biryul	wp-includes/http.php
11	2015-12-18 17:23	sergeybiryukc	Docs: Fix copy/paste error	3f35196e48f	Sergey Biryul	wp-includes/version.php
12	2015-12-18 12:02	sergeybiryukc	Widgets: Add missing clos	ff36a54824e	Sergey Biryul	wp-admin/widgets.php
13	2015-12-18 12:02	sergeybiryukc	Widgets: Add missing clos	ff36a54824e	Sergey Biryul	wp-includes/version.php
14	2015-12-17 23:44	a.fercia@gma	Accessibility: Remove the	6fdd27a1d6e	Andrea Ferci	wp-admin/css/edit-rtl.css
15	2015-12-17 23:44	a.fercia@gma	Accessibility: Remove the	6fdd27a1d6e	Andrea Ferci	wp-admin/css/edit.css
16	2015-12-17 23:44	a.fercia@gma	Accessibility: Remove the	6fdd27a1d6e	Andrea Ferci	wp-admin/css/wp-admin-rtl.min.css
17	2015-12-17 23:44	a.fercia@gma	Accessibility: Remove the	6fdd27a1d6e	Andrea Ferci	wp-admin/css/wp-admin.min.css
18	2015-12-17 23:44	a.fercia@gma	Accessibility: Remove the	6fdd27a1d6e	Andrea Ferci	wp-admin/includes/post.php
19	2015-12-17 23:44	a.fercia@gma	Accessibility: Remove the	6fdd27a1d6e	Andrea Ferci	wp-includes/version.php
20	2015-12-17 23:06	pascal.birchle	Editor: Correctly indent de	a7e13d0af98	Pascal Birchl	wp-admin/js/editor.js

Figure VI, raw dataset of commit information

B. Get Hotspot Score

Before we could simply run these algorithm steps in Windows Azure, we need to upload the source data file through import function in Windows Azure. After that we could simply drag to use the dataset from our private data container.

Then, implement bug prediction machine learning algorithm one step by one step. To run the whole processes, the time consume about 18 seconds. The result shows below represents hotspot score for every potential bug relative files.



rows	columns
1359	2

view as

file	score
wp-admin/list-manipulation-js.php	0.002517
wp-content/themes/twentythirteen/inc/back-compat.php	0.934245
b2trackback.php	0.000033
wp-admin/css/customize-controls-rtl.css	0.441066

Figure VII, partial result of hotspot score

C. Relation study by linear regression

After we getting the hotspot score for every file, we could put the results back to the original data table and do some linear regression to find some relationships. For example, we hook the score with commit author as so we could find who contribute most in this project.

Settings

Setting	Value
Bias	True
Regularization	0.001
Allow Unknown Levels	True
Random Number Seed	

Feature Weights

Feature	Weight
author_Boone Gorges_6	129.583
author_Jeremy Felt_21	88.2993
author_Rachel Baker_43	87.8051
author_Gary Pendergast_16	61.77
author_Aaron Jorbin_0	51.0224
author_Andrea Fercia_2	50.1434

Figure VIII, linear regression study about who contribute more in bug fixing

D. Critical check

The result from model evaluation shows in Figure IX left, we could see the mean absolute error is 0.260, which is pretty low considering this is a simple function to predict the buggy file in the repository.

To compare with it, we use the error rate of the repository itself. We equally split the data randomly into two parts, and use one part to check the criticality of other part. The result shows as Figure IX right. The mean absolute error is 0.256, which is very similar to the mean absolute error we use time ratio split way. It means, the bug prediction method by Google is pretty accurate.

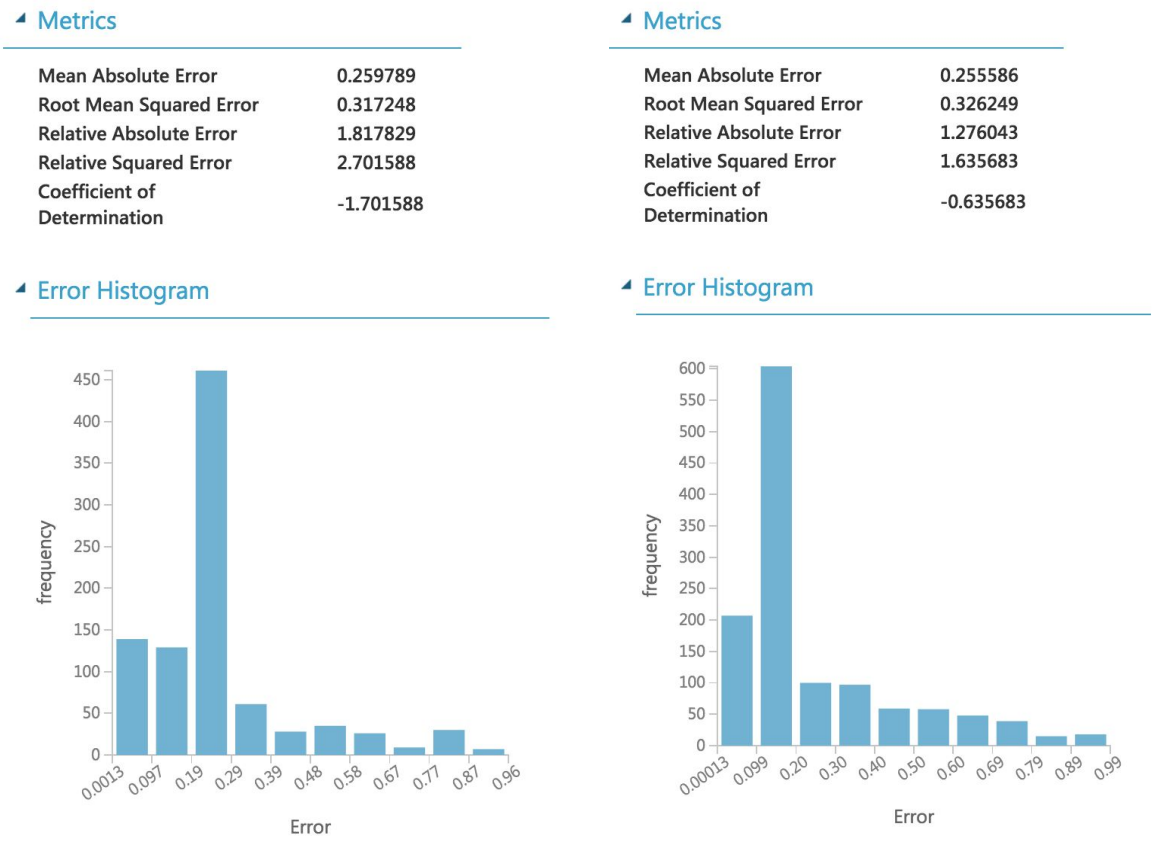


Figure IX, Error rate of split data using time point (left), Error rate of randomly split data (right)

VI. Conclusion

This project is predicting bug by using a simple machine learning algorithm in Window Azure platform. From the result we can see, reading the commit information and applying simple machine learning algorithm to it is very simple and effective way to get hotspot score for every bug fixing relative file. Although, from the critical checking part we can see there is still some improvement space in accuracy, bug prediction is just a tool to give some warning to coder, we don't need to be very accurate in result and use really complex functions and spend much time. This rapid build and run way could give coder a quick hint about which file have potential bug, and then necessary action or extra attention could be payed to that file.