Project EPSILON
Automatic JIRA triage

Yonatan Feleke, Gurkanwal Brar & Ashok Poothiyot

Stanford University

PROBLEM STATEMENT

The corporate world deals with task management in a variety of ways all having some form of triaging process to correctly assign tickets to developers. Automation of this task has proven elusive with less than 60% accuracy of latest ML solutions even for Web and SaaS companies that handle high volumes of tickets in the form of exceptions, support requests, user-reported bugs, and crash reports. Effective automation is essential to improve productivity and obviate the tedious work of manually triaging tickets.

Project aims to reduce this overhead by deploying a deep neural network classifier to assign tickets to a developer. The DNN model is trained by using the final assignee listed in the JIRA ticket as the label and predicts a previously seen developer on new tickets.

DATASET & FEATURES

The project utilized the generated jumble.expium.com dataset for developing the algorithm and then applies the architecture to train and test on the private LinkedIn Foundation team support dataset. The implementation uses a bag of words multinomial event model to vectorize the JIRA ticket. The text components of JIRA ticket namely: subject, body and comments, are concatenated and featurized with a frequency threshold of >= 5.

A JIRA ticket has the following JSON structure:


```
{
  "summary": "Update success. 3.0 USB Card",
  "description": "OR memory.dmp folder, ....",
  "priority": "Minor",
  "reporter": "chantal.colman",
  "labels": [
    "Communication"
  ],
  "worklogs": [],
  "status": "Open",
  "issueType": "Epic",
  "created": "2018-09-13T14:22:15-07:00",
  "updated": "2018-11-16T16:00:00-08:00",
  "affectedVersions": [],
  "fixedVersions": [],
  watchers": [
    "kevin.mcwhorter"
  ],
  "components": [],
  "externalId": "OLDMOB-167",
  "comments": [
    {
      "body": "No results. 3. Replacing the next
would like help someone has efficient cooling.
Crashes only hardware problems. Even something or
removing the buzz is supposed to do anything.",
      "author": "sarah.clark",
      "created": "2018-10-11T17:00:00-07:00"
    },
  "customFieldValues": [
    {
      "fieldName": "Epic Name",
      "fieldType": "com.opper.jira:gh-epic-label",
      "value": "Update success. 3.0 USB Card"
    }
  ],
  "history": [
    {
      "author": "chantal.colman",
      "created": "2018-11-06T16:00:00-08:00",
      "items": [
        {
          "fieldType": "jira",
          "field": "status",
          "from": 1,
          "fromString": "Open",
          "to": 3,
          "toString": "In Progress"
        }
      ]
    },
  ]
```


FEATURES, MODELS & RESULTS

Features and Vectorization

The input data described in the data set section was parsed into a bag of words, multinomial event model representation. The words are assumed to be independent and chosen with separate distributions at create time. During the preprocessing step we capture the list of frequent words at a threshold of >= 5 when building multinomial vector. While developing our solution we make the following assumptions:

/> A ticket t_i in a the set of tickets $\{t_1, \dots, t_{|T|}\}$ assigned to a developer $\{d_1, \dots, d_{|D|}\}$ is generated by a unique distribution modelled by θ:

$$P(\mathrm{t}_i|\theta) = \sum_{j=1}^{|\mathrm{d}|} P(\mathrm{t}_i|\mathrm{d}_j, \theta) P(\mathrm{t}_j|\theta)$$

/> Tickets are composed of independently and identically distributed words chosen at random (naive Bayes assumption) and from a multinomial distribution:

$$P(t_i|d_j, \theta) = \prod_{i=1}^{n} p(\mathrm{w}_t|t_j, \theta)^{occurance}$$

The labels are developers we have seen before represented as integers. We then create a matrix a multinomial input vector and an integer class label representing assignee to train and test. NB and SVM are not investigated by this paper but have been added only for reference.

The shape of the datasets after vectorizing were: LinkedIn m x n = <559, 2936> and Expium dataset m x n = <5435, 1970>

Network Model

We utilize a DNN to approximate θ in order to predict a designated assignee. As per the neural network design section, we use a 3x16 tanh neural network with a 0.005 learning rate and 1,000 backprop iterations. The final label (developer) selection is done via *softmax* followed by a *cross entropy loss* for backpropagation. The network is trained and tested using a kFold cross validation similar to the work by Bettenburg et al.

Results

Blow are the results of the experiments conducted on March 2018 tickets, larger compute/optimization is required to run full dataset.

Model	Linkedin Dataset (559/43483)		Expium Generated Dataset (~5500)	
	Train Accuracy	Test Accuracy	Train Accuracy	Test Performance
SVM Classifier (linear kernel, hinge loss)	1.0 w/ 70% train*	0.29166 w/o 30% test	1.0 w/ 70% train*	0.223175 w/ 30% test
Naive Bayes Classifier	5Fold Mean: 0.65251	5Fold Mean: 0.11613	5Fold Mean: 0.77097	5Fold Mean: 0.208463
Deep Neural Network Classifier	5Fold Mean: .996420	5Fold Mean: **0.058976833**	2Fold Mean : 1.0	2Fold Mean: **0.071940617**

Remarks

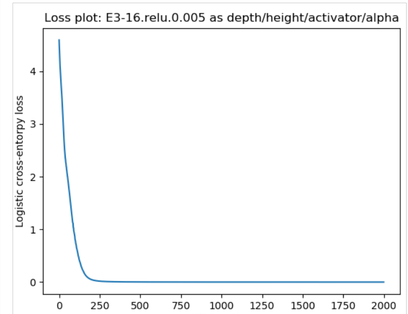
- The highly non-linear nature of DNN has allowed almost perfect 5Fold mean accuracy on over 3,000 LinkedIn tickets and 5,500 generated tickets, that same benefit has heavily overfit the data as observed by the dismal <7% test prediction accuracy.
- Computational challenges with vectorizing text has prevented us from significantly scaling the number of trained samples, working on a batching solution to count words efficiently, store to a datastore and fetch on batched training and prediction stages.
- A developer that has never been assigned a ticket will not be considered during prediction, this is normal in triaging.
- The SVM classifier was only run with splitting up data once instead of the bagging like kFold cross validation completed for the others

Neural Network Design

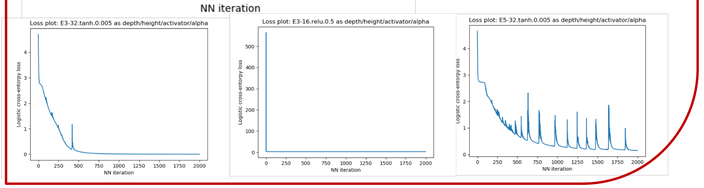
The Neural network parameters were chosen by searching through parameters using the Expium dataset. The dimensions investigated in the images on the left are:

- Activator Functions: [tanh, relu]
- Learning rates: [0.5, 0.05, 0.005]
- Hidden layers: [8, 16, 32]
- Depth : [3, 5]

The architecture chosen is: a 3 wide, 16 high, ReLu activated network with a learning rate of 0.005 and 1,000 backpropagation iterations.



* The architecture is chosen due to the fast and smooth descent and relative simplicity of the network compared to the higher train but lower test accuracy architectures below.



Discussion Remarks

Conclusions: The initial results of our experiments resulted in poor accuracy on the test set. The DNN overfit the small dataset and we hope that after running on full set (~9x more tickets) the test accuracy will dramatically increase, currently experiments have only been applied to a set of 5,000 tickets

Future Work: The most important work is to implement **dropout** to fix overfit. Additional features from JIRA ticket fields leaves a lot of room for improvement

- Take advantage of NLP concepts to improve test accuracy, implementing stemming, lemmatization, Word2Vec and word embeddings.
- Utilize additional features such as: watchers, labels, reporter, hashed exceptions and so on.
- Make the vectorization process cacheable to handle entire dataset.

References

[1] Murphy, G., and D. Cubranic. "Automatic bug triage using text categorization." *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. 2004.

[2] – Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958. APA

[3] – Domingos, P., Pazzani, M., 1996. Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. In: Machine Learning, Morgan Kaufmann, pp. 105–112.

[4] – Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S., 2008. Duplicate Bug Reports Considered Harmful. . .Really? In: ICSM.

[5] – WUYUNTANA, D. and WANG, S. (2018). Distributed Representations of Mongolian Words and Its Efficient Estimation. DEStech Transactions on Computer Science and Engineering, (iceit).