Epsilon: General ML, NLP

By Yonatan Feleke <yfeleke@stanford.edu>, Ashok Poothiyot <apoothiy@stanford.edu> and Gurkanwal Brar <gbrar@vmware.com>

1 Abstract

The corporate world deals with task management in a variety of ways with each 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. Project Epsilon explores this area by deploying a deep neural network to predict assignees.

2 Introduction

Web and SaaS companies handle high volumes of tickets in the form of exceptions, support requests, user-reported bugs, and crash reports. Mostly with dedicated teams that work on aggregating, triaging and assigning these tickets to the right individual or team. However, effective automation is essential to improve productivity and obviate the tedious work of manually triaging tickets.

Project Epsilon aims to eliminate this overhead by experimenting with supervised-learning classifiers to intelligently and automatically assign tickets to a developer. With high failure rates in state of the art solutions, we aim to deliver higher accuracy for predicting assignee for a new ticket based on past tickets using deep learning models.

The input to our algorithm is a collection of historic JIRA tickets in JSON format. These tickets are preprocessed and featurized and following which we predict the assignee for new or unassigned tickets using 3 different methods: SVM, Naive Bayesian and Deep Neural Network Classifiers. We then compare the performance for these 3 methods as pertaining to 2 primary input datasets: a public Expium generated dataset as well as a dataset with real Jira tickets from Linkedin.

3 Related Work

Classification on open bug reports with supervised learning strategies has been done in the past. Popular strategies are Naive Bayes and SVM classification on a multinomial event model input featurized as a bag of words. The primary research revolves around interpreting text to predict assignee based on observed history. Deep neural network implementation to ticket categorization seems to be uncommon and implementing dropout and skip layers is an interesting area of research to improve our implementation.

Of the ones detailed in section 10, [2] makes stronger assumptions using implied behavioral patterns and developer dependencies to improve performance. [3], on similar lines, extracts and utilizes intention from the ticket text fields while [4] uses hierarchical attention based contextualization for more robust classification. [1] is most relevant research close to our deep neural network experiments and promises even larger datasets and varied projects.

4 Datasets and Features

The Linkedin dataset's security requirements of keeping data on LinkedIn assets and preventing the external distribution of JIRA tickets has unfortunately split our development and implementation environments. As such, the development of our models, metrics, and experiments was completed on the generated www.jumble.expium.com dataset for developing the algorithm and then applied the architecture to train and test on the private LinkedIn Foundation team support dataset.

A JIRA ticket from our datasets 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"
          }
       ]
    },
]
```


For a deeper dive into the featurization process, the input JSON data is parsed into a bag of words and a multinomial event model is used to vectorize the JIRA ticket. Words in the description, body and comment sections are concatenated and converted to a feature vector via a dictionary mapping of words to indices. We count words that occur more than five times. In our preprocessing step, we remove words that occur in our list of stopwords and remove duplicate JIRA ticket entries (3, 666 or 8.43%), all of which is implemented using a modification of our homework code with a future plan to use word2vec.

The words are assumed to be independent and are expected to have been chosen with separate distributions at create time. We've analyzed and explored cross-entropy error across different parameters of a neural network and compared runs with a support vector machine classifier and a naive Bayes classifier. 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:

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

The labels are developers we have seen before represented as integers, which came to a total of 1403 unique developers. We then create a matrix of a multinomial input vector and an integer class label representing the assignee and go ahead with training and testing. The shape of the Expium datasets after vectorizing was: m x n = <5435, 1970>. The LinkedIn dataset, however, repeatedly demonstrated a wide feature vector (m > n) throughout different slices of the data: m x n:- <559, 2936>, <28200, 46595>, <43483, 65350> demonstrating that even after we remove stopwords there are a lot of unique words per ticket.

Instead of breaking up our dataset to explicit train, cross-validation and test sets, we opted for k-fold cross-validation [10] which creates 80% train and 20% validation groups in k different ways and averages results to report test and train accuracy.

5 Methods

The project aims to evaluate the applicability of deep neural networks to classify ticket assignment or classification using all previously assigned developers as the labels (classes for classification) and previously assigned tickets as training data. The problem is treated as a text classification problem with a novel experiment in using a deep neural network grid search to find optimal architecture and activation functions. The wide format of the feature vectors proves to be a challenge with the highly non-linear nature of deep neural networks which resulted in overfitting on the data. We suspect that much larger datasets will be required to overcome the sheer variance of available words to create tickets.

5.1 Naive Bayes and SVM Classifiers

To give an idea of the performance of the deep neural network, we implemented these traditional methods and report accuracy values when using the currently assigned developer as a label and body of text as the feature vector to evaluate cross-entropy loss on the data set. The implementations still use the same feature vectors.

The Naive Bayes classifier works by calculating the probability of a particular word mapping to a developer and those parameters are used during predict time to select the most probable developer. We use a Support Vector Machine with a linear classifier and the algorithm works by forming an optimal separation between the data points close to the boundary. Using a kernel, the algorithm is able to form highly non-linear classification by operating at higher dimensions.

5.2 Deep Neural Network Classifier

Our Neural network classifier operates by constantly updating weights by back-propagating after observing a cross entropy loss. During each backpropagation round the various layers update their weights to be able to fit the problem better. The activation functions are useful by making the neural network non-linear, i.e by preventing the entire architecture from simplifying to a linear regression with a combination of the weights. The components are the learning rate, hidden layers, height, activator function, loss function, and the final output layer.

We chose a deep neural network because of its capability to fit highly non-linear data sets and our use case benefits heavily from this property due to its freeform nature. We chose a softmax output step and cross entropy loss for our architecture and for the rest we completed a parameter search using the Expium data set focusing on testing 3, 5 and 8 layers, 8,16 and 32 neurons, and relu and tanh activator functions. A 5-fold experiment is used to determine performance and find the optimal architecture with 2000 backpropagation iterations. The architecture is then compared to the other solutions to explore performance in accurately predicting assignee.

A future extension of our research might leverage NLP methods to form better matching among tickets for classification. Especially to reduce the number of features evaluated, if we're able to map similar meaning words into a single feature we expect to reduce our overfitting and also make the features more linearly independent.

6 Experiments

We ran the following experiments based on the http://jumble.expium.com/ generated datasets. A typical JIRA ticket has text body values in summary, description, comments and body sections. The featurization step concatenates all text elements and (unless otherwise specified as in 6.1) builds a multinomial event model with counts tracked for words that appear more than 5 times in a bag of words model assuming words are independently selected.

6.1 DNNs Architecture selection

Selecting the correct DNN marks the first stage of the experiments with the goal of improving predictions. Selecting Neural network parameters is a challenging task so we tried out combinations to achieve better accuracy measures for our test sets. We explored cross-entropy training error across different parameters of the neural network. The following test parameters were experimented with and analyzed.

Table1: Neural network design parameters

Parameter	**Tested Values**
Activator	Reul, tanh
Learning rate (alpha)	0.5, 0.05, 0.005
Depth	3,5,8
Height	8, 16, 31
Backprop Iterations	2000

The focus on this execution is to observe the best minimization strategies during training that would have given lowest final logistic cross-entropy loss. The experiment was run with 2,000 iterations with 5-fold cross validation. The results were a fairly surprising mix of high train and test tradeoffs and some of the algorithms suffered from local optima traps. The experiment accuracy values are listed below as are some of the graphs related to the cross-entropy loss per iteration.

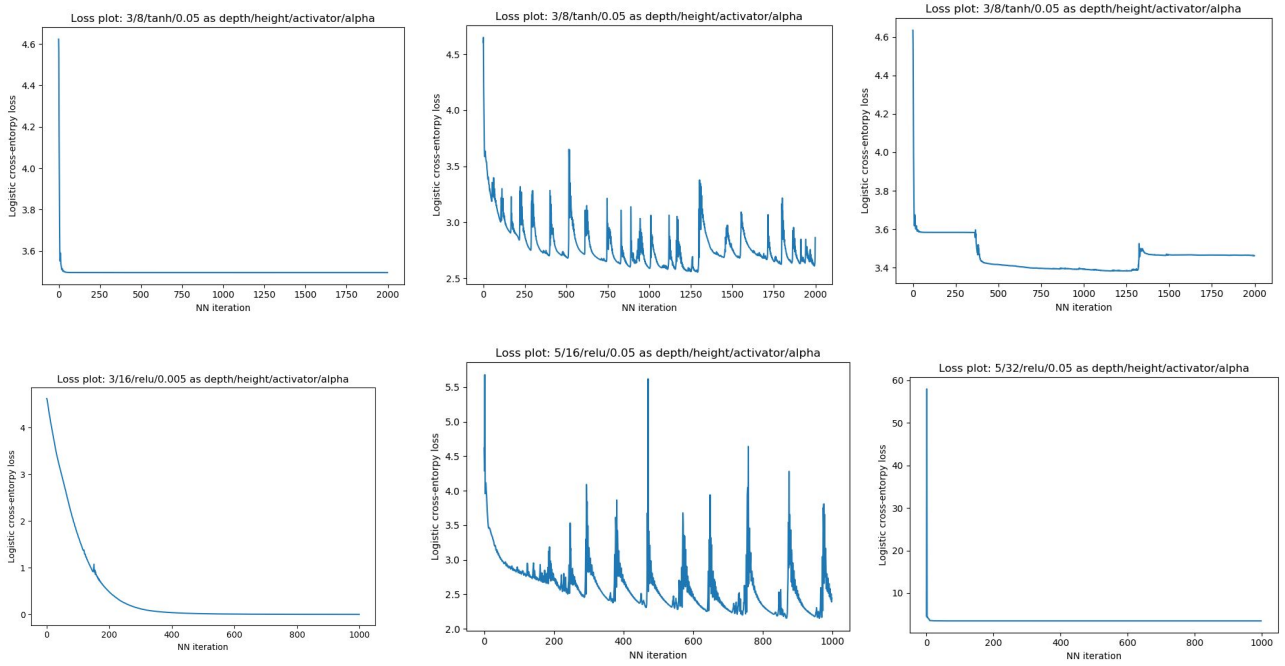
Table 2: 5-Fold mean accuracy values for Expium dataset (5500 tickets) on NN architectures.

Activator	Height	Depth	Alpha	Train Accuracy	Test Accuracy
tanh	8	3	0.5	0.3177196530281229	0.3177196530281229
tanh	8	3	0.05	0.3215842095242583	0.30263503198323694
tanh	8	3	0.005	0.7701970776212672	0.08775843265212382
tanh	8	5	0.5	0.3177196530281229	0.3177196530281229
tanh	8	5	0.05	0.3177196530281229	0.3177196530281229
tanh	8	5	0.005	0.6857608717141656	0.05722404894590326
tanh	16	3	0.5	0.2954524465503901	0.06201693043798307

tanh	16	3	0.05	0.4331042684127383	0.14829360446300147
tanh	16	3	0.005	0.9576768841320951	0.06715592799594194
tanh	16	5	0.5	0.3177196530281229	0.3177196530281229
tanh	16	5	0.05	0.3177196530281229	0.3177196530281229
tanh	16	5	0.005	0.7817730079842313	0.08260751873508931
tanh	32	3	0.05	0.2954524465503901	0.06201693043798307
tanh	32	5	0.05	0.5037704443420721	0.13136939548581236
tanh	32	3	0.05	1.0	0.05832631216040069
tanh	32	3	0.05	0.3177196530281229	0.3177196530281229
tanh	32	3	0.005	0.3177196530281229	0.3177196530281229
tanh	32	5	0.05	0.9836276674025018	0.06384399265193967
ReLu	8	3	0.5	0.3177196530281229	0.3177196530281229
ReLu	8	3	0.05	0.44118471087798383	0.08151019810134484
ReLu	8	3	0.005	0.8311258278145696	0.057406789020591736
ReLu	8	5	0.5	0.3177196530281229	0.3177196530281229
ReLu	8	5	0.05	0.6147251532403153	0.07322947955572565
ReLu	8	5	0.005	0.8657098101155264	0.0579592747595536
ReLu	16	3	0.5	0.3177196530281229	0.3177196530281229
ReLu	16	3	0.05	0.43402440091181815	0.08832303786991832
ReLu	**16**	**3**	**0.005**	**1.0**	**0.0610852878193415**
ReLu	16	5	0.5	0.3177196530281229	0.3177196530281229
ReLu	16	5	0.05	0.3177196530281229	0.3177196530281229
ReLu	16	5	0.005	0.8105224429727741	0.041397905374900845
ReLu	32	3	0.05	0.3177196530281229	0.3177196530281229
ReLu	32	5	0.05	0.6180509007277917	0.10892642271171375
ReLu	32	3	0.05	1.0	0.07525031801783283
ReLu	32	3	0.05	0.3177196530281229	0.3177196530281229
ReLu	32	3	0.005	0.5063439716628981	0.10193118140138008
ReLu	32	5	0.05	1.0	0.06439600444480194

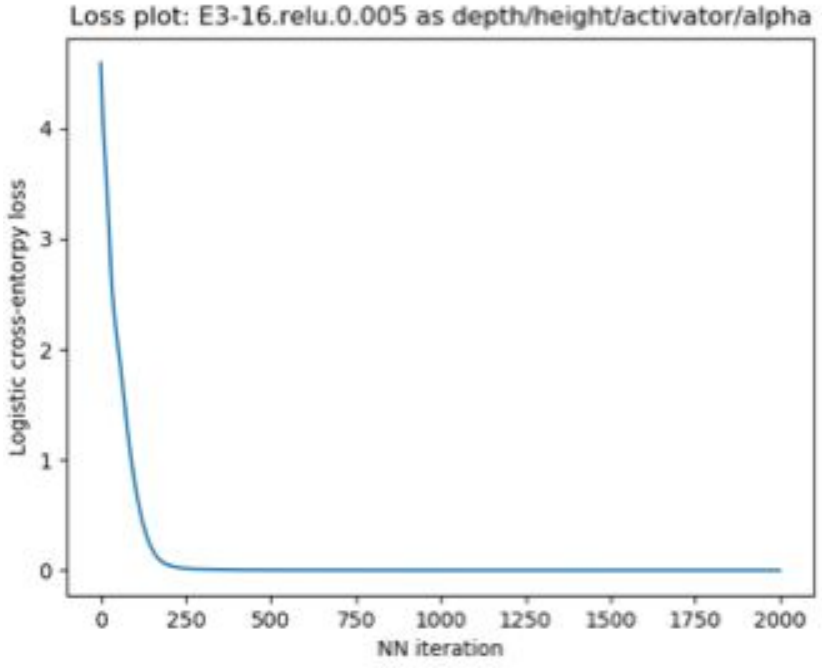
Looking at the results, the choice for an architecture was not obvious because the small data set was usually over fit when we get high train accuracy. The more interesting observations were on the output of graphs compiled with varying the different parameters and looking at the train loss per-iteration. Depending on the parameters, we see wildly varying patterns that need to be tuned for better accuracy and efficient computation. **It was surprising to see that taller and wider nets did not necessarily yield lower training loss.** The full list of graphs or the previous data can be found here.

Figure 1: Training cross entropy loss by iteration for different NN architectures



In the end, we selected the simplest high train accuracy neural network structure with the hopes that the larger dataset will fix the over fitting issue and also by picking the simplest architecture with the smoothest descent we are also attempting to reduce the even more non-linear capabilities of bigger networks.

Figure 2: Final chosen architecture cross entropy loss by iteration



The architecture:

- Width: 3 wide
- Height: 16 high
- Activator:ReLu
- Learning: 0.005
- Iterations: 1,000

6.2 Classification Accuracy

In this section we will compare executions of the above chosen neural network as it compares to Naive Bayes and an SVM classifier with a linear kernel. The different tables represent accuracy comparison on different subsets of the data. Unfortunately, we had computational hurdles when trying to run the SVM classifier on larger values.

Table 3: Expium Generated Dataset of 5500 tickets

Model	Train Accuracy	Test Accuracy
SVM	1.0 w/ 70% train*	0.223 30% test
Naive Bayes	5-Fold: 0.770	5-Fold: 0.208
DNN	5-Fold : 1.0	5-Fold: 0.0719

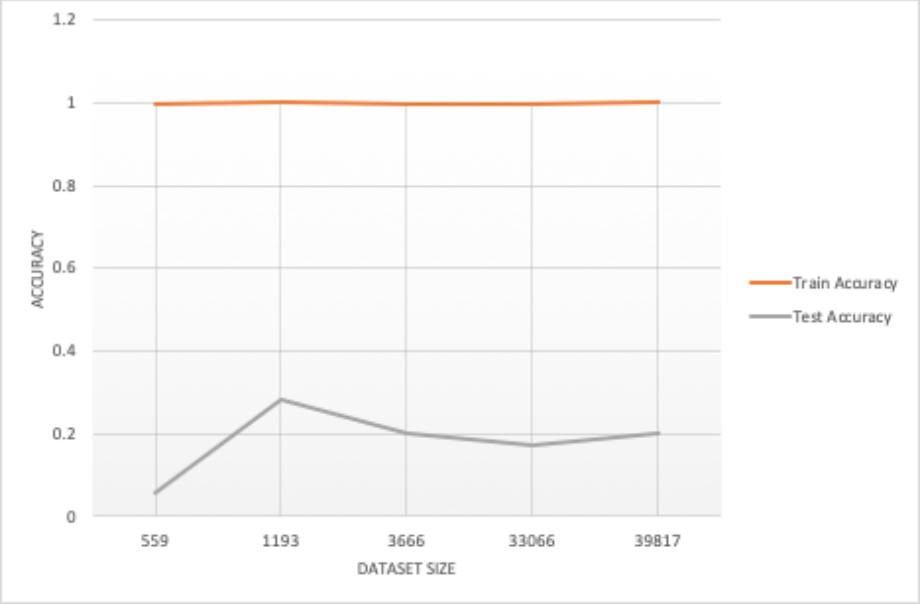
Table 4: LinkedIn Dataset: 559 tickets out of total 43483 cp, arospm

Model	Train Accuracy	Test Accuracy
SVM	1.0 w/ 70% train*	0.291 w/o 30% test
Naive Bayes	5-Fold 0.65251	5-Fold 0.11613
DNN	5Fodl: .996	5Fold: 0.0589

Table 6: LinkedIn Dataset: 39817 tickets

Model	Train Accuracy	Test Accuracy
Naive Bayes	0.3968	0.2255
DNN	0.9997	0.2020

Figure 3: DNN 5 fold cross validation mean train and test accuracy by number of samples



The highly non-linear nature of DNN has allowed almost perfect 5-fold mean accuracy but that same benefit has heavily overfit the data as observed by the low test prediction. The Naive Bayes model doesn't continue to improve that much with more data but we see that our neural network improves, this is due to the higher non-linear fitting capabilities of a deep neural network compared to Naive Bayes probability selection methods. For smaller sample sizes the traditional approaches completely outperform because the deep neural network overfits data.

7 Conclusions

The low accuracy rates in predicting a developer make the currently tested experiments not viable for the primary triaging solution.The results of our experiments resulted in poor accuracy on the test set because the DNN overfit the small dataset. However, the current solution still has a lot of opportunities for improvement. In particular, dropout implementation [8] and even more samples could get the solution to keep improving. As is with the current ~40,000 tickets worth of data, we can not reliably predict the developer but can provide value by predicting top-k developers or whole teams instead of individuals.

8 Future work

The most important work is to implement dropout[8] to reduce our overfit challenges and also take advantage of NLP concepts to improve test accuracy. Primarily if we can reduce the number of features (words > 5 frequency) by merging words that mean the same thing (word embeddings), we may drastically reduce our overfitting issue. Implementing stemming and lemmatization may also help merge features and allow the neural network to have a more linearly independent feature vector.

Additional features from the JIRA ticket fields also provide opportunities for improvement: watchers, labels, reporter, hashed exceptions and so on may provide additional information for classification.

9 Contributions

The contributions section is not included in the 5 page limit. This section should describe what each team member worked on and contributed to the project

1. Yonatan Feleke: Featurization, experiments, algorithm development, writeup and poster.
2. Gurkanwal Brar: Algorithm development and accuracy calculation
3. Ashok Poothiyot: Investigated datasets, Initial data preparation, setup scripts, writeup

Project Code: https://github.com/yfeleke/epsilon

10 References

1. Mani, Senthil, Anush Sankaran, and Rahul Aralikatte. "DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging." *arXiv preprint arXiv:1801.01275* (2018).
2. Xi, Shengqu, et al. "An Effective Approach for Routing the Bug Reports to the Right Fixers." *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*. ACM, 2018.
3. Huai, Beibei, et al. "Mining Intentions to Improve Bug Report Summarization."
4. Lyubinets, Volodymyr, Taras Boiko, and Deon Nicholas. "Automated labeling of bugs and tickets using attention-based mechanisms in recurrent neural networks." *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*. IEEE, 2018.
5. WUYUNTANA, D. and WANG, S. (2018). Distributed Representations of Mongolian Words and Its Efficient Estimation. DEStech Transactions on Computer Science and Engineering, (iceit).
6. – Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S., 2008. Duplicate Bug Reports Considered Harmful. . .Really? In: ICSM.
7. – Domingos, P., Pazzani, M., 1996. Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. In: Machine Learning, Morgan Kaufmann, pp. 105–112.
8. – 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
9. Murphy, G., and D. Cubranic. "Automatic bug triage using text categorization." Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. 2004.
10. Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection." Ijcai. Vol. 14. No. 2. 1995.