Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Statistics
2020 | LIU-IDA/STAT-A--20/025--SE

Automated Bug Classification

Bug Report Routing

Sridhar Adhikarla

Supervisor : Caroline Svahn
Examiner : Krzysztof Bartoszek

External supervisor : Daniel Nilsson

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

With the growing software technologies companies tend to develop automated solutions to save time and money. Automated solutions have seen tremendous growth in the software industry and have benefited from extensive machine learning research. Although extensive research has been done in the area of automated bug classification, with the new data being collected, more precise methods are yet to be developed. An automated bug classifier will process the content of the bug report and assign it to the person or department that would fix the problem.

A bug report typically contains an unstructured text field where the problem is described in detail. A lot of research regarding information extraction from such text fields has been done. This thesis uses a topic modeling technique, Latent Dirichlet Allocation (LDA), and a numerical statistic Term Frequency - Inverse Document Frequency (TF-IDF), to generate two different features from the unstructured text fields of the bug report. A third set of features was created by concatenating the TF-IDF and the LDA features. The class distribution of the data used in this thesis changes over time. To explore if time has an impact on the prediction, the age of the bug report was introduced as a feature. The importance of this feature, when used along with the LDA and TF-IDF features, was also explored in this thesis.

These generated feature vectors were used as predictors to train three different classification models; multinomial logistic regression, dense neural networks, and DO-probit. The prediction of the classifiers, for the correct department to handle a bug, was evaluated on the accuracy and the F1-score of the prediction. For comparison, the predictions from a Support Vector Machine (SVM) using a linear kernel was treated as the baseline.

The best results for the multinomial logistic regression and the dense neural networks classifiers were obtained when the TF-IDF features of the bug reports were used as predictors. Among the three classifiers trained the dense neural network had the best performance, though the classifier was not able to perform better than the SVM baseline. Using age as a feature did not give a significant improvement in the predictive performance of the classifiers, but was able to identify some interesting patterns in the data. Further research on other ways of using the age of the bug reports could be promising.

Acknowledgments

I would like to express my gratitude to my supervisor, Caroline Svahn, for a through supervision. Her feedback and motivation helped me through this thesis.

I would like to thank my supervisor at Ericsson, Daniel Nilsson, for helping me understand the data and for answering my numerous questions. I would also like to thank my line managers at Ericsson, Roland Sevegran and Christer Lindell, for considering me to work on this project.

I would like to acknowledge the employees at Ericsson, Attila Szabo and Bela Bartalos, for taking the time to help me understand the current implementation for the task.

Thanks to my examiner, Krzysztof Bartoszek, for his feedback and suggestions.

Thanks to my opponent Raymond, for having provided me with valuable constructive remarks.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

As a company grows larger, they develop an organized way of handling problems to speed up the process. For software companies, bug reports are raised describing the problem. Bug reports are the medium for communicating a problem to the developers. The reports contain a clear description of the problem and sometimes also the way to reproduce the problem so that the developer gets a better idea of the problem. These bug reports are then assigned to the department that can fix the bug. This tedious process going through the bug reports and assigning them to the concerned department is called bug assignment.

Humans tend to make a lot of mistakes when it comes to tedious and repetitive tasks [1]. this is one of the reasons we start relying on machines for such things. Automating such tasks saves a lot of time and money for the companies, and also reduces manual errors due to mental fatigue.

Bug assignment is a good example of such a tedious and repetitive task. For large software industries, fixing a bug fast is very important. Correctly assigning a bug to the department that is going to handle it plays a key role in speeding up the bug fix. The process usually involves going through the bug report description, which is a combination of free unstructured text, code snippets, and stack traces. Several studies report that doing this manually is tedious and error-prone [2]. Incorrect assignment of bugs leads to delays in the process of fixing due to bug tossing between departments until finally, it reaches the correct one. Supervised machine learning algorithms can be used on the past bug reports data to train an automated system to assign bugs in a fast and efficient manner.

Many of the papers on Automatic Bug classification, [2] [3] [4] [5], use open-source software bug reports and suggest a supervised machine learning approach to solve the problem. The heading and description are the most commonly used fields of a bug report to train the classifier. The authors of [6] claim that most of the studies use the bag of words model for feature generation from the description, which does not consider the syntactic and semantic information available in the bug report description. They go on to propose a bug report representation algorithm, using attention-based deep bidirectional recurrent neural networks (DBRNN-A), that learns syntactic and semantic features from long word sequences in an unsupervised manner. This makes use of the unlabeled data as well which could increase the dataset size by a lot for some companies.

The authors of [2] claim that the ensemble-based learner, Stacked Generalization (that combines several classifiers), can outperform the use of individual classifiers for bug assignment. They also go on to claim that using bug reports that are too old can decrease the prediction accuracy of bug assignments. Their study was conducted on data from five large proprietary development projects where the bugs were assigned to departments instead of individual developers, this also involved the bug reports data from the 4G development at Ericsson.

Other studies have been done on the Ericsson's bug report data to explore different ways that could help build a better classifier for the task. The authors of [7], implemented a new approach for high-dimensional multi-class regression tasks called Diagonal Orthant Latent Dirichlet Allocation (DOLDA) [8] on the bug data. This algorithm combines Latent Dirichlet Allocation [9], an unsupervised topic modeling approach, and Diagonal Orthant Multinomial Probit Models [10]. The topic distributions learned using LDA, from the free unstructured text in the bug report's heading and description, are used as features for the Probit model. Another study by Artchounin [11] suggested using K-Fold cross-validation to train the classifiers. Another study by Svahn [12], explored using the files associated with the bug reports to classify. The major focus of this study was exploring if the alarm logs associated with the bug reports could be useful to classify the bug reports.

1.2 Aim

Most of the studies on Automated bug report assignment used open-source datasets which usually involve the bug report being assigned to one of the hundreds of developers. For large software companies, the bug reports are assigned to a department instead of a developer, which is much fewer compared to the number of developers. This thesis involves bug reports from Ericsson. Some research has already been done on this dataset and Ericsson also has an automated implementation in place. The current implementation uses the Support Vector Machine (SVM) classifier with a linear kernel [13] to solve the task.

A good way for humans to improve at problem-solving is by asking the question, "Can we do better?". This is the aim of this master thesis. The overall accuracy and the F1 score of the current implementation (SVM) will be taken as the baseline for this thesis. The goal of this thesis will be to try and improve on the baseline. There is lot to learn from the previous research being done on this dataset [2][7][11][12]. This thesis tries to build upon some of those ideas to come up with a good implementation for the task.

1.3 Research questions

This thesis will intend to answer the following research questions.

1. Is it possible to improve on the current baseline?
   This is the main aim of the master thesis. This thesis will explore different machine learning algorithms and compare them to the current implementation's overall accuracy and F1 score.

2. Can age of the bug report be used as a feature?
   As claimed in the papers [2] [12], the prediction accuracy decreases as we include bug reports that are too old. This thesis will also explore if age has an impact on this dataset and will try to improve the prediction using the age of the bug report as a feature.

2 Data

This chapter introduces the bug reports dataset used in this thesis. The dataset is a collection of labeled 4G and 5G bug reports, from the telecommunications company Ericsson. The data used contains a mix of bug reports raised internally by Ericsson employees and by external customers. The reports contain a clear description of the problem which helps track the source of the problem. These reports are to be routed to the department that could fix the problem.

2.1 Bug reports

The dataset contains 30,878 bug reports. Each report in the dataset is identified by a unique id and contains a heading, an observation field, the registration date for the bug report, and information regarding which department finally handled the bug report. An example of how the data looks like can be seen in Table 2.1.

Eriref	Heading	Observation	Registration date	MHO
HU65504	goes to network loader mode if trigger.....	trouble summary commercial effect dus	2016-03-09	RCS-DEV
HY14738	mtr ec3 rbsnclm sigsegv cmd proc	description crash observed sites frequency num	2019-11-25	CAT-SW
HW74295	prach preamble faulty detection nb iot cell	trouble summary commercial effect wrong prach	2018-03-26	LTE-BBSW

Table 2.1: Sample of the Data

Each bug report is assigned a unique id called *Eriref* when it is created. The *Heading* and *Observation* are unstructured text fields. *Heading* is a one-line summary of the problem while the *Observation* field contains a clear description of the problem, sometimes even steps to

reproduce the problem to help the developers understand the problem better. The *Observation* field usually contains keywords that could help when assigning the bug report to the correct department. The *Registration date* of the report is the date on which the report was created. *Modification Handling Office (MHO)* is the name of the department that finally handled the problem. *MHO* is the response variable for the task.

There are 20 unique *MHOs* a bug report can be assigned to, however, the data is not evenly distributed among them. The distribution of the data is shown in Figure 2.1.

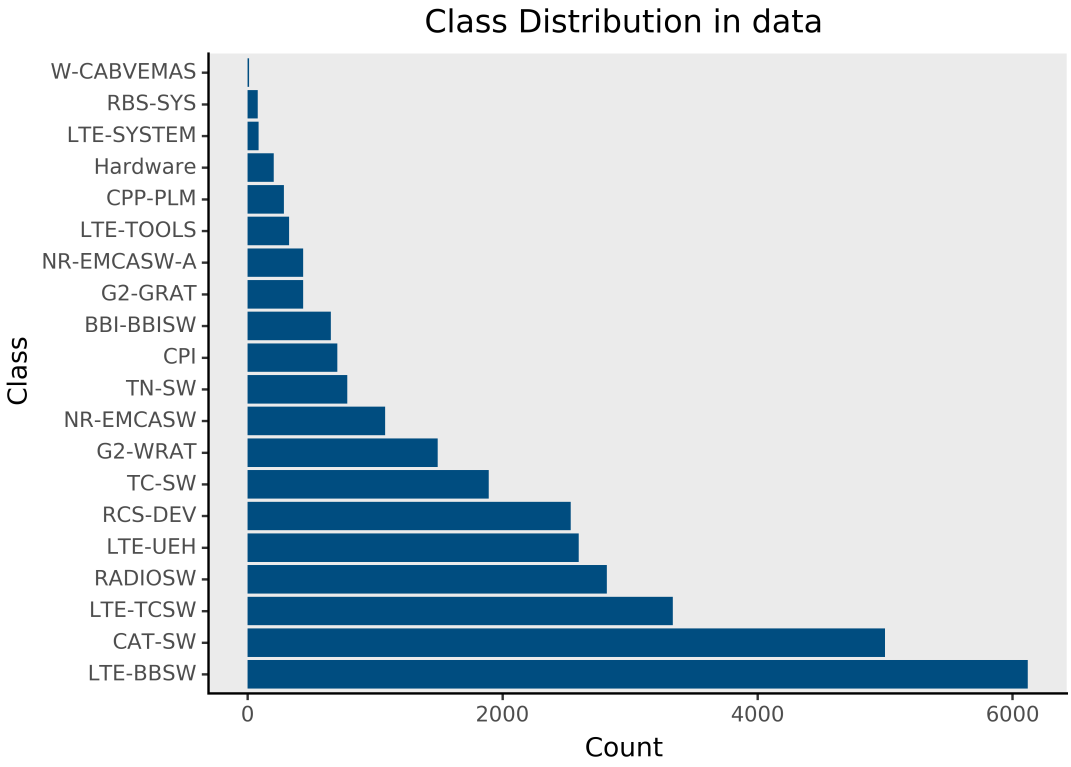


Figure 2.1: Distribution of response variable in data

It can be seen that the *MHOs W-CABVEMAS*, *RBS-SYS*, *LTE-SYSTEM* are highly under-represented in the dataset. Since the bug reports belonging to these minority classes rarely occur, good performance of the classifier on these classes is not of much importance to the company.

Some of these *MHOs* are old and occur rarely in the new incoming bug reports. It can be seen from Appendix A.5 that there were no bug reports from *W-CABVEMAS*, and approximately 20 bug reports from *RBS-SYS*, *LTE-SYSTEM* and *HARDWARE* in 2019. This thesis will discard the bug reports belonging to the MHOs *W-CABVEMAS*, *RBS-SYS*, *LTE-SYSTEM*, *HARDWARE*, from the dataset. The resulting dataset has 30000 bug reports with 16 unique *MHOs*. This data is further split into training, validation, and test in a manner that yields similar class distribution for all three sets. 70% of the data was used for training the classifiers, 15% as validation for tuning the hyper-parameters, and the remaining 15% was used as test data to evaluate the classifiers.

2.2 Distribution of data over time

The dataset contains bug reports from 2015-12-01 to 2020-02-29. The distribution of data changes over time due to increase in use of the 5G network. The distribution of the data over the years can be seen from the following 6 images:

1. Data distribution in 2015 Appendix A.1.
2. Data distribution in 2016 Appendix A.2.
3. Data distribution in 2017 Appendix A.3.
4. Data distribution in 2018 Appendix A.4.
5. Data distribution in 2019 Appendix A.5.
6. Data distribution in 2020 Appendix A.6.

It can be seen that bug reports from *TC-SW* and *TN-SW* increase after 2018. These are some of *MHOs* belonging to the 5G department of Ericsson.

3 Method

In this chapter, the methods being used in the thesis are described. The methods used can be divided into three categories, feature construction, model construction, and model evaluation. The following sections go on to describe each of them.

3.1 Feature Construction

Unstructured text data is valuable, but it has to be cleaned and processed to be used in a machine learning algorithm. The following subsections describe the methods, along with the required background, used to extract features from such unstructured text data.

Beta Distribution

The Beta distribution is a family of continuous probability distributions defined on the interval $[0, 1]$. The distribution is parameterized by two positive shape parameters, denoted by α and β. The Beta distribution, with $0 \leqslant x \leqslant 1$, has the following probability distribution:

$$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \tag{3.1}$$

where

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{3.2}$$

and Γ is the Gamma function.
The Beta density function can take a wide variety of different shapes depending on the parameters α and β. A Beta distribution with shape parameters $\alpha = \beta = 1$ is the same as a uniform distribution. This distribution is commonly used in Bayesian inference as a conjugate prior [14].

Dirichlet Distribution

The Dirichlet distribution is a multivariate generalization of the Beta distribution and is also called the Multivariate Beta Distribution. The probability density function for the Dirichlet distribution, with number of categories $K \geqslant 2$, is of the form:

$$f(x_1, ..., x_k; \alpha_1, ..., \alpha_k) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}, \tag{3.3}$$

where

$$B(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{j=1}^{K} (\alpha_j))}, \tag{3.4}$$

$x_1, ..., x_k$ represent a k dimension unit simplex, $0 \leqslant x_i \leqslant 1$ and $\sum_i^K x_i = 1$,
and $\alpha = (\alpha_1, ..., \alpha_K)$ is the vector of concentration parameters for the Dirichlet distribution, where $\alpha_i > 0$.

The concentration parameter α controls the probabilities of the distribution. For $\alpha_i < 1$ $\forall i$, the probabilities are more scattered towards the corners, while for larger values of α_i the probabilities are more concentrated around a single point. For $\alpha_i = 1$ $\forall i$, the distribution is uniform over the unit simplex [14].

Multinomial Distribution

The Multinomial distribution is a generalization of the Binomial distribution. The probability mass function of the Multinomial distribution is of the form:

$$f(x_1, ..., x_k; n, p_1, ..., p_k) = \frac{\Gamma(\sum_i x_i + 1)}{\prod_i (\Gamma(x_i + 1))} \prod_{i=1}^{k} p_i^{x_i}, \tag{3.5}$$

where n is the number of trials, with $(n > 0)$, and $p_1, ..., p_k$ are event probabilities with $\sum p_i = 1$.
The number of success categories is defined by k. When $k = 2$, the Multinomial distribution is either the Bernoulli distribution for $n = 1$, or Binomial distribution for $n > 1$. For $k > 2$ and $n = 1$, it represents the Categorical distribution. For k>2 and n>1 it is the Multinomial distribution.

The Multinomial and the Dirichlet distribution are very similar, therefore the Dirichlet distribution is a conjugate prior for the Multinomial distribution [14].

Latent Dirichlet Allocation

Jonsson et al. [7] used Diagonal Orthant Latent Dirichlet Allocation (DOLDA) [8] to solve the problem of automatic bug localization. DOLDA [8] was initially proposed by Jonsson et al. as a supervised topic model for high dimensional classification. DOLDA combines Latent Dirichlet Allocation (LDA) [9] with the supervised Diagonal Orthant probit (DO probit) [10] model for bayesian classification. LDA is a topic modeling technique for unstructured text data.

Topic modeling is a type of statistical modeling technique in text classification, used for discovering abstract topics that occur in a collection of documents. It is similar to soft clustering on numerical data, where algorithms try to find some natural groups of items. A topic is defined by a distribution over the words and as in soft clustering, each document can be assigned to multiple topics with different probabilities. LDA is an unsupervised topic modeling technique, which uses a set of unobserved groups or topics to explain the observations or documents.

LDA relies on the Dirichlet and the Multinomial distributions to generate the topics from the documents collection [9]. LDA makes the assumption that the documents collection was

created from a generative process and tries to reverse engineer the process. Random mixtures over latent topics are used to represent documents, where each of the topics are further categorized by a distribution over the words in the corpus.

Assume a document corpus D which contains M documents, each of length N_i where $i = 1, ..., M$. Let the total number of unique words in the document corpus, also known as the vocabulary of the document corpus, be of length V. A generative process with K topics looks like:

1. For each topic $k = 1, ..., K$:
   a) Simulate a distribution over words $\varphi_k \sim Dirichlet_V(\beta)$
2. For each document $d = 1, ..., M$:
   a) Simulate topic proportions $\theta_{d|\alpha} \sim Dirichlet_K(\alpha)$
   b) For $i = 1, ..., N_d$:
      i. Simulate a topic assignment $z_{i,d}|\theta_d \sim Multinomial(\theta_d)$
      ii. Draw a word $w_{i,d}|z_{i,d}, \varphi_{z_{i,d}} \sim Multinomial(\varphi_{z_{i,d}})$

where the model parameters β is the parameter of the Dirichlet prior on the per-topic word distribution, and α is the parameter of the Dirichlet prior on the per-document topic distribution.

The unknowns of the model to infer are φ_k is the vector of word probabilities for each topic k, θ_d is the topic proportions for the document d, and z_d is the topic assignment for the words.

A Dirichlet prior is introduced for all the unknown θ_d, z_d and φ_i to obtain a joint posterior distribution over the variables. As the Dirichlet distribution is conjugate to the Multinomial distribution, the joint posterior distribution follows a Dirichlet distribution:

$$p(\theta_{1:D}, z_{1:D}, \varphi_{1:K}|w_{1:D}) \propto \prod_{i=1}^{K} p(\varphi_i|\beta) \prod_{d=1}^{D} p(\theta_d|\alpha) \left(\prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\varphi_{1:K}, z_{d,n}) \right). \quad (3.6)$$

The authors of LDA [9] used Gibbs Sampling to obtain $z_{1:D}$, $\theta_{1:D}$ and $\varphi_{1:K}$. An implementation of LDA from the python library `Gensim` will be used in this thesis. This LDA implementation in `Gensim` [15] uses Variational Bayesian Sampling to sample from the joint posterior.

Term frequency - Inverse Document Frequency

Term Frequency - Inverse Document Frequency (*TF-IDF*) [16], is a numerical statistic that shows how important a word is to a document in a corpus. The *TF-IDF* value increases proportionally with the number of times a word appears in a document and is offset by the number of documents in the corpus that contain the word.

The number of times a word occurs in a document is called *Term Frequency(tf)*. There are many forms of calculating the *tf* value. The one used in this thesis is called Augmented Term Frequency. It is calculated as the frequency of a term in the document divided by the frequency of the most frequent term in the document. This prevents a bias towards longer documents. The *tf* value is calculated as:

$$tf(w, d) = 0.5 + 0.5 \frac{f_{t,d}}{max\{f_{t',d} : t' \in d\}}, \quad (3.7)$$

where $f_{t,d}$ is the frequency of the term in the document and $max\{f_{t',d} : t' \in d\}$ is the frequency of the most popular term in the document.

The *Inverse Document Frequency(idf)* is a measure of how much information the term provides. It is calculated as:

$$idf(w, D) = log \frac{N}{1 + |\{d \in D : w \in d\}|}, \tag{3.8}$$

where N is the total number of documents in the corpus ($N = |D|$), and $|\{d \in D : w \in d\}|$ is the number of documents where the word w appears (also known as document frequency for the word).
The *TF-IDF* is calculated as a product of the *tf* and the *idf*:

$$tfidf(w, d, D) = tf(w, d) \cdot idf(w, D). \tag{3.9}$$

The words that have high term frequency and low document frequency in the corpus have the highest *tf-idf* value [16]. While calculating the *idf* as the ratio:

$$\frac{N}{1 + |\{d \in D : w \in d\}|} \geqslant 1. \tag{3.10}$$

The value of *idf* and *tf-idf* is always greater than zero.
This thesis uses an implementation of *TF-IDF* from the python library `scikit-learn` [17].

3.2 Model Construction

This section goes on to introduce the classification methods used in this thesis along with the background required.

Multinomial Logistic Regression

Logistic Regression [14], is a statistical method that is used when the dependent variable is nominal with two levels. For problems where the dependent variable is nominal with more than two levels the *Multinomial Logistic Regression* is used. *Multinomial Logistic Regression* is a generalization of the *logistic regression* to multi-class problems.

Similar to logistic regression, the *Multinomial Logistic Regression* uses a linear combination of the independent variables to predict the dependent nominal variable [14]. It trains a different linear combination of the independent variables for each of the categories in the dependent variable. The outputs of the linear combinations are combined to get the probabilities for each class, and to make a prediction the class with the highest probability is selected.

Let a dataset D contain (x, y), where $x \in \mathbb{R}^n$ is the vector of independent variables and y is the dependent categorical variable with K levels. The activation for class j is defined as:

$$a_j = b_j + W_j^T \cdot x, \tag{3.11}$$

where $j = 1, \ldots, K$, $b_j \in \mathbb{R}$ and $W_j \in \mathbb{R}^n$. b_j and W_j are the learned class conditional parameters of the model.
To convert the K linear combinations obtained into probabilities a *softmax* transformation is performed. Softmax is a normalized exponential function which takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponential of the input numbers. Let $\mathcal{K}$ be the response variable with $K > 2$ levels, the softmax transformed probability for class $\mathcal{K} = j$ given x is obtained as:

$$\Pr(\mathcal{K} = j|x) = \frac{\exp(a_j)}{\sum_{k=1}^{K} \exp(a_k)}, \tag{3.12}$$

where $j = 1, \ldots, K$.

Maximum Likelihood Estimation (MLE) is used to obtain the optimal class conditional parameters as:

$$l\left(\{b_j, W_j\}_{j=1}^{K}\right) = \frac{1}{N}\sum_{i=1}^{N}\log \Pr(\mathcal{K} = j|x_i), \tag{3.13}$$

where $j \in \{1, \ldots, K\}$, and $\Pr(\mathcal{K} = j|x_i)$ is the probability of class j given the feature vector x_i for observation i and the parameters $\{b_j, W_j\}_1^K$. Due to non-linearity in the *softmax* transformation, there is no closed form solution for maximizing log likelihood for multinomial logistic regression [14]. Gradient based optimization such as *SAGA* is usually used to find the optimal parameters for the model. SAGA [18] is an incremental gradient algorithms with fast linear convergence rates. This thesis uses an implementation of *Multinomial Logistic Regression* from the python library `scikit-learn` [17].

Elastic net regularization

Least Absolute Shrinkage and Selection Operator (*LASSO*), proposed by Tibshirani [19], is a regression analysis method that performs both variable selection and regularization to increase the prediction accuracy. LASSO is useful for high dimensional data, as it reduces the number of features in the model to predict the target variable. To preform LASSO regularization penalty is added to the loss function:

$$l_1 = -\lambda \sum_{j=1}^{p} |\beta_j|, \tag{3.14}$$

where λ is the regularization coefficient and β is the vector of model coefficients. This is also known as the L1 norm.

Ridge regression also known as L2 norm, proposed by Hoerl and Kennard [20], provides improved efficiency in parameter estimation for models with a large number of features. Ridge does not reduce the number of features but reduces the impact that each feature has on the model by reducing the coefficient value. To perform Ridge regularization penalty is added to the loss function:

$$l_2 = -\lambda \sum_{j=1}^{p} \beta_j^2, \tag{3.15}$$

The authors of [21] proposed *Elastic net*, which is a weighted combination of the Ridge Regression and the LASSO. Elastic net can perform both adjustable variable selection and coefficient shrinkage. The loss function with the elastic net penalty is given by:

$$l_{enet}(\hat{\beta}) = l(\hat{\beta}) + \lambda\left(\frac{1-\alpha}{2}\sum_{j=1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j=1}^{m}|\hat{\beta}_j|\right), \tag{3.16}$$

where m is the number of features in the data, α is the mixing parameter for Ridge and LASSO with $0 \leqslant \alpha \leqslant 1$, β is the vector of model coefficients and λ is the regularization coefficient.

α determines the weights of the *LASSO* and *Ridge* penalties. $\alpha = 0$ indicates the Ridge penalty while $\alpha = 1$ indicates the LASSO penalty [21]. The aim is to minimize the loss function $l_{enet}(\hat{\beta})$. A python implementation for Logistic regression with elastic net penalty from the library `scikit-learn` [17] will be used.

Multinomial Diagonal Orthant Probit

Probit models are most commonly used for Bayesian classification tasks. Standard probit models use the Markov Chain Monte Carlo (MCMC) approach to compute the posterior. Johndrow et al. in their paper [10] talk about the inefficiency of the MCMC approach in practice due to the highly dependent latent variables and parameters, and also with the difficulties in efficiently sampling latent variables when there are more than two categories. To address this inefficiency they proposed the *Multinomial Diagonal Orthant Probit (DO-probit)* model. DO-probit assumes conditional independence between the latent variables, given the parameters, to enhance the performance.

DO-probit models uses a set of binary variables to represent an unordered categorical variable. Let y, an unordered categorical response variable with K levels, be represented by $\gamma_{[1:K]}$, an independent binary variable. If $y = l$, where $l < K$, the unordered categorical variable y can be represented by the vector $\gamma_{[1:K]}$ as:

$$\{\gamma_l = 1\} \cup \{\gamma_k = 0 \ \forall \ k \neq l\}, \tag{3.17}$$

a vector of zeros at all positions except index l. This is also known as a *one-hot* representation for a categorical variable.
Latent variables can be used to represent the binary variables as:

$$z_l \sim f(\mu_l, \sigma), \tag{3.18}$$

where f is the location scale density function, μ_l is the location parameter and σ is the common scale parameter. To ensure all $\gamma_l = 1$ and $\gamma_k = 0 \ \forall \ k \neq l$, the z's are restricted to the set:

$$\Sigma = \bigcup_{j=1}^{K} \left\{ z \in \mathbb{R}^K : z_l > 0, z_k < 0, k \neq l \right\}, \tag{3.19}$$

z_l corresponding to the $\gamma_l = 1$ is greater than zero, and $z_i \forall i \neq l$ is smaller than zero.

Now, if we let the location-scale function f to be a normal probability density function, we get a DO-probit model [10]. The joint probability density function of the latent variables in DO-probit is a K variate normal distribution, with identity and location parameter restricted to one positive value and the rest negative. The name diagonal orthant multinomial model comes from the fact that the marginal distribution of any two latent variables will be restricted to orthants that are diagonally apposed rather than adjacent.
The probability of class l in DO-probit model is:

$$\Pr\left(y_i = l | x_i, \boldsymbol{\beta}_{[1:K]}\right) = \frac{(1 - F(x_i\beta_l)) \cdot \prod_{k \neq l} F(-x_i\beta_k)}{\sum_{s=1}^{K} (1 - F(x_i\beta_s)) \cdot \prod_{k \neq l} F(-x_i\beta_k)}, \tag{3.20}$$

where F is the standard normal cumulative distribution function.

The Horseshoe Estimator

The feature generation methods described in the previous section of this thesis are likely to produce a sparse feature vector. A sparse vector is a vector with a lot of values exactly zero. Suitable regularization methods should be used for the classifier to be able to recover the signal from the noisy feature vector. Carvalho [22] proposed a horseshoe prior to solve the problem. The authors of [10] use the horseshoe prior with the DO-probit model to shrink the model coefficients.
Given a sparse feature vector β_i and the global shrinkage parameter τ, the horseshoe prior for β_i is given by the hierarchical model:

$$y_i|\beta_i \sim N(\beta_i, \sigma^2 I_n), \tag{3.21}$$

$$\beta_i|\lambda_i, \tau \sim N(0, \tau^2\lambda_i^2), \tag{3.22}$$

$$\lambda_i \sim C^+(0,1), \ \ (i = 1, ..., n), \tag{3.23}$$

where $C^+(0,1)$ is a standard half-Cauchy distribution, and τ is assumed to be fixed.
The posterior mean of the model can be shown to be:

$$E(\beta_i|y) = \int_0^1 (1-\kappa_i)\, y_i p(\kappa_i|y)\, d\kappa_i = [1 - E(\kappa_i|y)]\, y_i, \tag{3.24}$$

where:

$$\kappa_i = \frac{1}{1+\tau^2\lambda_i^2}. \tag{3.25}$$

The posterior mean, $E[\beta|y,\tau]$, is referred to as the horseshoe estimator and is denoted by $T_\tau(y)$. The horseshoe prior gets its name from the prior on k_i, which is given by:

$$p_\tau(\kappa_i) = \frac{\tau}{\pi} * \frac{1}{1-(1-\tau^2)\,\kappa_i} (1-\kappa_i)^{-\frac{1}{2}}\, \kappa_i^{-\frac{1}{2}}. \tag{3.26}$$

If $\tau = 1$, this reduces to a $Beta(0.5, 0.5)$ distribution, which looks like a horseshoe. A java implementation for DO-probit with the horseshoe prior by Jonsson et al. [7] will be used in this thesis.

Dense Neural Network

A mathematical model for a biological *Neuron* is called a *Perceptron*. A *Perceptron* is a neural network unit that does some computation on the input data to detect features. A neural network is a network of such *Perceptrons* stacked up in layers. The first and last layers of the network are called input and output layers. The layers in between, called hidden layers, perform most of the computations required by our network [14]. Figure 3.1 is a visualization of a basic Neural network structure.

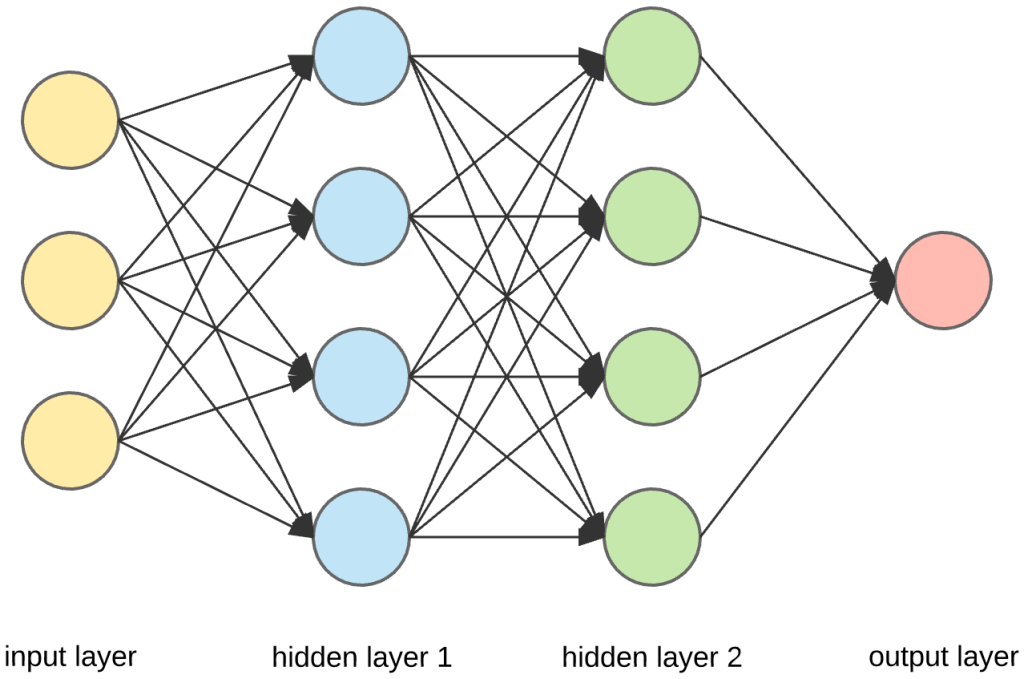


Figure 3.1: Visualization of an example neural network.

Each *Perceptron* takes a linear combination of the inputs from the previous layer and passes it through an activation function to produce the output for the next layer [14]. The computation on each *Perceptron* looks like:

$$a_{i+1} = g(\sum w_{i+1} \cdot a_i), \tag{3.27}$$

where:
a_i : Vector of inputs to the *Perceptron*,
a_{i+1} : Single value output from the *Perceptron*,
w_{i+1} : Weights used for the computations of that *Perceptron*,
g : Activation function used for that layer.
This method of computing outputs from the *Perceptrons* at each layer is called a *feed-forward* step of the neural network training. The weights for the network are learned using the *back-propagation* algorithm. The feed-forward step of the network makes a prediction from the inputs, and the errors are propagated back through the network updating the weights. The matrix notation for a feed-forward step of a neural network is given by:

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \times \mathbf{A}^{[l-1]} + b^{[l]}, \tag{3.28}$$

$$\mathbf{A}^{[l]} = g^{[l]}(\mathbf{Z}^{[l]}), \tag{3.29}$$

where:
$\mathbf{W}^{[l]}$: Weight matrix for layer l,
$\mathbf{Z}^{[l]}$: Weighted sum of activation from previous layer,
$\mathbf{A}^{[l]}$: Activation output from layer l,
$g^{[l]}$: Activation function for layer l,
$b^{[l]}$: Bias vector for layer l.

Matrix notation for the back-propagation step of the neural network is given by:

$$\delta \mathbf{Z}^{[l]} = \delta \mathbf{A}^{[l]} \cdot g^{[l]'}(\mathbf{Z}^{[l]}), \tag{3.30}$$

$$\delta \mathbf{W}^{[l]} = \frac{1}{m}(\mathbf{Z}^{[l]} \times \mathbf{A}^{[l-1]T}), \tag{3.31}$$

$$\delta b^{[l]} = \frac{1}{n}\sum \delta \mathbf{Z}^{[l]}, \tag{3.32}$$

$$\delta A^{[l-1]} = \mathbf{W}^{[l]T} * \delta \mathbf{Z}^{[l-1]}, \tag{3.33}$$

where n is the total number of training examples and δ indicates gradient.

The activation function for a layer must be non-linear, as the purpose of the activation function is to introduce non-linearity into the network. An example for the activation function is the Rectified Linear Unit, also known as *ReLU*:

$$f(x) = max(0, x). \tag{3.34}$$

Similar to *Multinomial Logistic Regression*, for multi-class classification the neural network uses a *Softmax* layer as the output layer for the network.

A common way to regularize neural networks is to use a *Dropout* layer [23]. *Dropout* regularization reduces over-fitting in neural networks by preventing complex co-adaptations on training data. *Dropout* randomly drops or shuts down *perceptrons* of the neural network in each iteration of the training. This prevents the network from relying too much on some particular *perceptrons*, and spreads the weights across the network.
This thesis will use the Tensorflow library [24] to build neural networks.

3.3 Model Evaluation

The methods used in this thesis to evaluate and compare different classifiers are described in this section.

Overall Accuracy

Accuracy gives us a rough idea of how good the classifier is doing on a dataset. It is the percentage of correctly classified observations. For binary classification tasks, the prediction results can be summarized in a 2-dimensional matrix called *confusion matrix*. Figure 3.1 shows an example confusion matrix for binary classification task:

	Predicted Values	
Actual Values	True Positive	False Negative
	False Positive	True Negative

Table 3.1: Confusion matrix for binary classification

The accuracy can be calculated from the confusion matrix as:

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + FalsePositive + TrueNegative + FalseNegative}. \tag{3.35}$$

For a classifier with K classes, the overall accuracy can be calculated from the confusion matrix using the formula:

$$Accuracy = \frac{\sum_{i=1}^{K} C_{i,i}}{\sum_{i=1}^{K} \sum_{j=1}^{K} C_{i,j}}, \tag{3.36}$$

where C is the confusion matrix of dimension K.

F1 score

The *F1* score [14], is a harmonic mean of the *precision* and *recall*, with its value in the range $[0, 1]$. A *F1* score of 1 indicates perfect *precision* and *recall* for the classifier.
Precision, or *specificity*, is the fraction of relevant instances among the retrieved instances. For binary classification it can be calculated from Table 3.1 as:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}. \tag{3.37}$$

Recall, or *sensitivity*, is the percentage of the correctly classified positive cases. For binary classification it can be calculated from Table 3.1 as:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}. \tag{3.38}$$

The *F1 score* is the harmonic mean of the Precision and Recall of the classifier. For binary classification it can be calculated from Table 3.1 as:

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{3.39}$$

The multinomial case will yield one *F1* score per class. An average of all the scores has to be calculated to get the overall *F1* score of the classifier. This thesis uses a *Micro-average F1* score to handle the class imbalance in the dataset. The *Micro-average F1* score takes a weighted average of the F1 scores, weighted by the size of the class. This thesis will be using an implementation from the python library `sklearn` [17] to calculate the *F1* score.

4 Results

This chapter will present the results step-wise as they were produced. The first section presents the results of the feature extraction methods. The second section will motivate the choice of the hyper-parameter values for the classifiers. The third section will show the results for using the age of the bug reports as a feature. The fourth section will present the prediction results from the trained classifiers. The final section will try to visualize the trained features vectors.

As mention in Chapter 2, the dataset was divided into three parts, 70% was used for training, 15% was used for validation and 15% was used for the final tests. Since the feature vectors obtained from TF-IDF were large and compute-heavy, *Cross Validation* was not used to evaluate the classifiers. The training data has been used for training the feature extractors and the classifiers, the validation data has been used to find the optimal hyper-parameters for the classifiers and the test data is used to present and evaluate the final prediction results for the classifiers.

4.1 Feature Construction

The unstructured text fields of the bug reports, Heading and Observation, were merged as one text field. To construct features from this unstructured text field *Feature Extraction* methods, LDA and TF-IDF, were used. The following sections present the results for these methods.

TF-IDF

The heading and the observation fields for the bug reports in the training dataset contained approximately 500,000 unique words. Not all of these words are important when training a classifier, so a subset of words with the highest TF-IDF values was chosen as the features. The motivation behind the choice of the subset size is presented in this section.

The Logistic Regression and the Neural Network classifiers were used to motivate the choice of the max features for TF-IDF. The accuracy of the validation dataset for the classifiers was used to evaluate the choice. The overall validation accuracy for a different number of features used in the classifiers is presented in Figure 4.1.

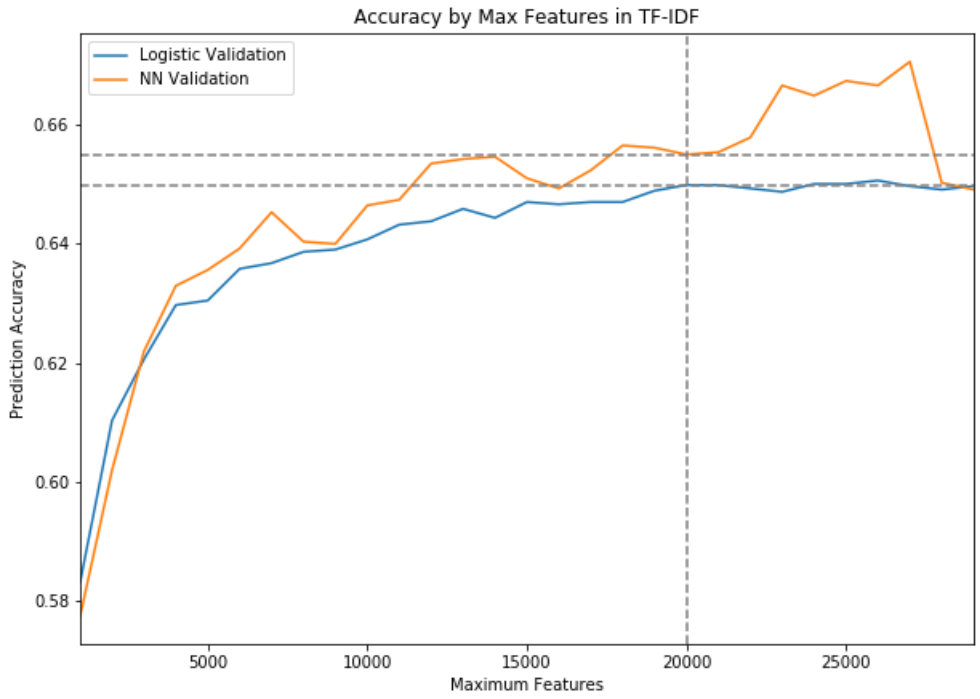


Figure 4.1: Selecting maximum number of features required.

It can be seen from the plot that the validation accuracy for the logistic regression classifier almost flattens for more than 20,000 TF-IDF features. The neural network classifier shows some fluctuations in the validation accuracy. 20,000 is chosen as max features for TF-IDF and will be used to generate the TF-IDF feature vectors. The following sections will refer to the features as TF-IDF (20,000).

For TF-IDF (20,000); logistic regression got an accuracy of 0.649, and neural networks got an accuracy of 0.655. Neural networks got the highest accuracy using TF-IDF (20,000) as predictors.

LDA

This thesis tries different number of topics for LDA to see if that helps improve the performance of the classifier. The motivation behind the choice of the number of topics for LDA is presented in this section.

Logistic regression, Neural Networks, and DO-probit classifiers were used to motivate the choice of the number of topics for LDA. The accuracy of the validation dataset for the classifiers was used to evaluate the choice. The overall validation accuracy for the different number of LDA topics used in the classifiers is presented in Figure 4.2.

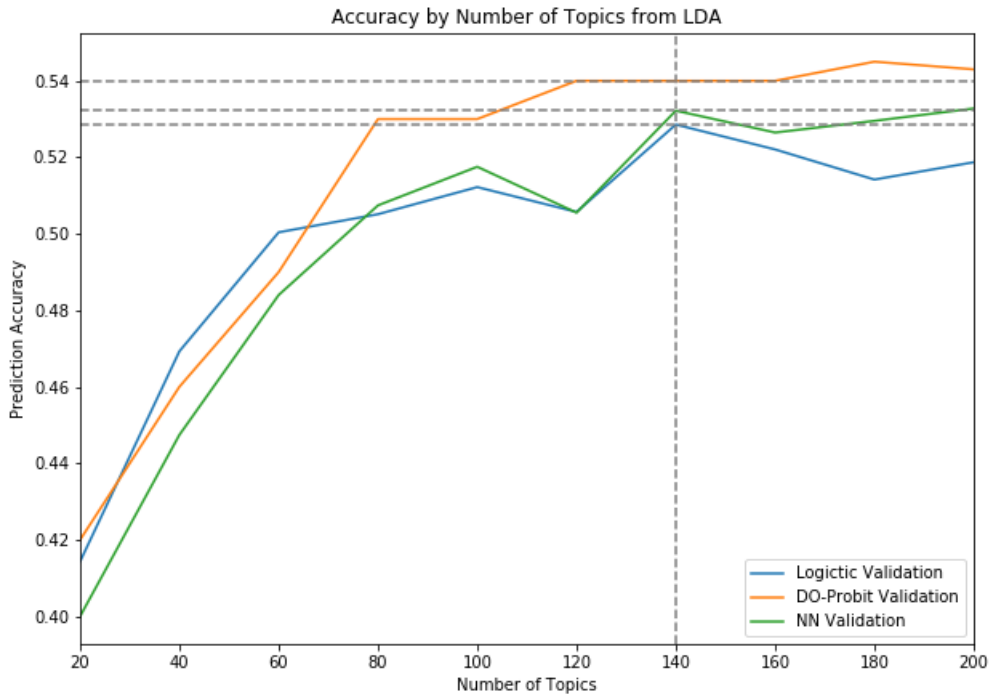


Figure 4.2: Selecting number of Topics required.

It can be seen that the validation accuracy for classifiers either flatten or decrease after 140 topics. 140 is chosen as the best value for the number of topics for LDA. The following sections will refer to it as LDA (140).

For LDA (140); logistic regression got an accuracy of 0.528, neural networks got and accuracy of 0.532, and the DO-probit model got an accuracy of 0.54. DO-probit got the highest accuracy using LDA (140) as predictors.

4.2 Choice of hyper-parameters

This section motivates the choice of the hyper-parameter for the Logistic regression and Neural Networks classifiers.

Multinomial Logistic Regression

The Elastic net regularization used with the multinomial logistic regression classifier is controlled by the hyper-parameters, α, and C. This subsection presents the results motivating the choice of these hyper-parameters. The first part estimates the plausible values for α, and the second part shows the optimal values obtained for the hyper-parameters, using a grid search.

Plausible values for α

The rate of the L1 and L2 regularization is controlled by the α hyper-parameter. The training and validation accuracy for two classifiers, using TF-IDF features, over a grid of values for α is shown is Figure 4.3. The *Logistic V1* classifier used TF-IDF (20,000) as features and the *Logistic V2* classifier used TF-IDF (20,000) + LDA (140) as features.

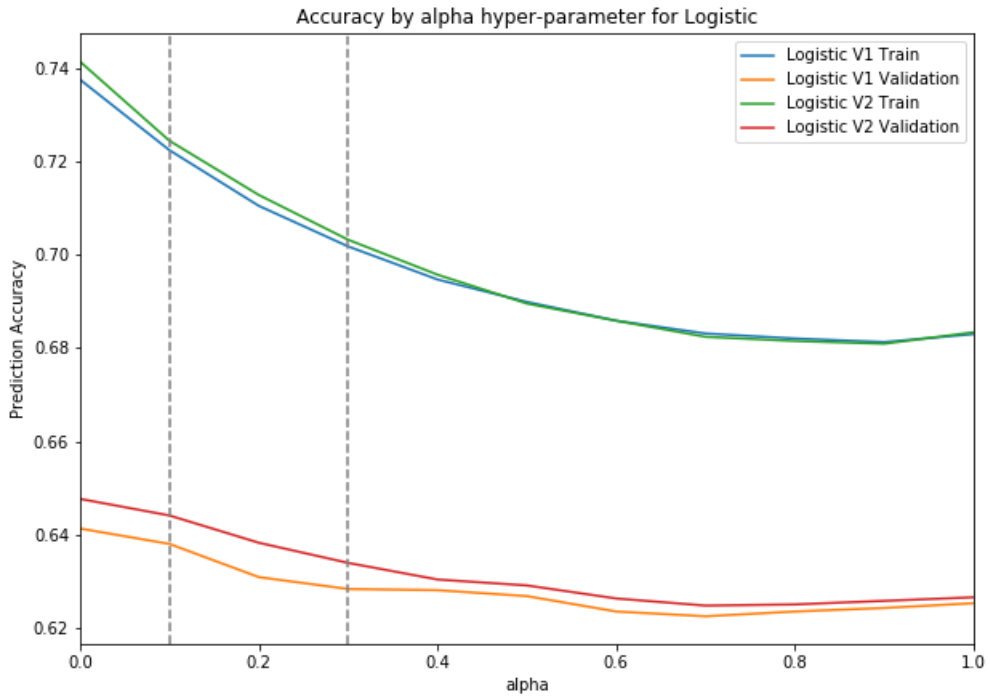


Figure 4.3: Accuracy of Logistic Regression classifiers with TF-IDF features, for different values of α.

It can be seen that for both the classifiers, the train and validation accuracy start going down as α increases. The decline in the training accuracy is steeper than the validation accuracy. The range of values selected for α was $[0.1, 0.3]$. The validation accuracy starts to decrease even more after that.

A separate plot was created for multinomial Logistic Regression with LDA (140) features, as a completely different set of regularization parameters could be useful with the LDA features. The training and validation accuracy for the classifier using LDA (140) features, over a grid of values for α is shown is Figure 4.4.

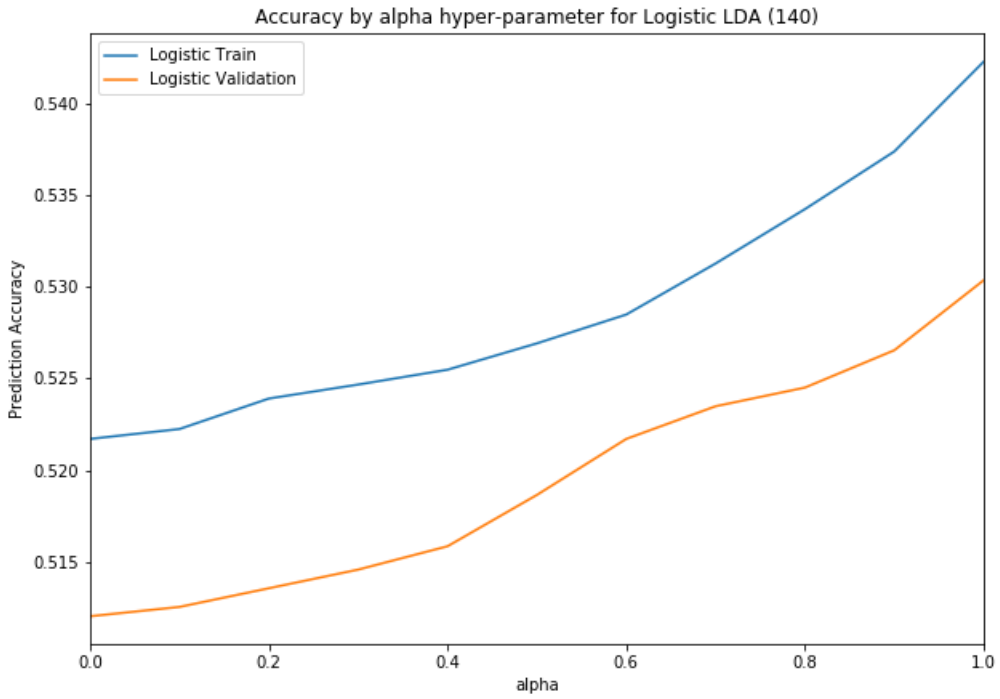


Figure 4.4: Accuracy of Logistic Regression classifier with LDA features, for different values of α.

The train and the validation accuracy for the classifier increase with the increase in the α value. It can be seen that the classifier does not overfit the training data, as both the curves grow consistently and there is not much difference between the training and the validation accuracy. The best validation accuracy of 0.53 was obtained at α=1, indicating L1 regularization is best for this classifier. α=1 was be used for logistic regression with LDA (140).

Grid search to find optimal α and C

The C hyper-parameter is the inverse of regularization strength. Grid search was used to find the optimal values for α and C. The range of α values selected from the previous section was used with a grid of 10 C values in the range $[0, 2]$. Table 4.1 shows the optimal hyper-parameter values for different features used with multinomial logistic regression.

	Hyper-parameters	
Features	α	C
LDA (140)	1	1.2
TF-IDF (20,000)	0.1	1.1
TF-IDF (20,000) + LDA (140)	0.1	1.2

Table 4.1: Optimal values for logistic regression hyper-parameters, found using grid search.

It can be seen that the LDA features prefer L1 regularization while the TF-IDF features prefer a weighted mixture of L1 and L2 regularization. The L1 regularization has a weight of 0.1 and L2 regularization has a weight of 0.9 in the elastic net mixture.

Dropout rate for Neural Networks

Three different variations of neural network classifiers were used in this thesis and their optimal values for the dropout rate is motivated in this section. The classifiers differ in terms of the input features to the neural network. *NN V1* classifier uses TF-IDF (20,000) as features, *NN V2* classifier uses TF-IDF (20,000) + LDA (140) as features and the *NN V3* classifier uses LDA (140) as features.

Multiple architectures of neural network classifiers were tried, and the best results were obtained using a classifier with two hidden layers containing 300 and 100 neurons respectively. Each hidden layer is followed by a dropout layer which helps regularize the network. The training and validation accuracy for different versions of neural networks on a grid of dropout rates can be found in Figure 4.5.

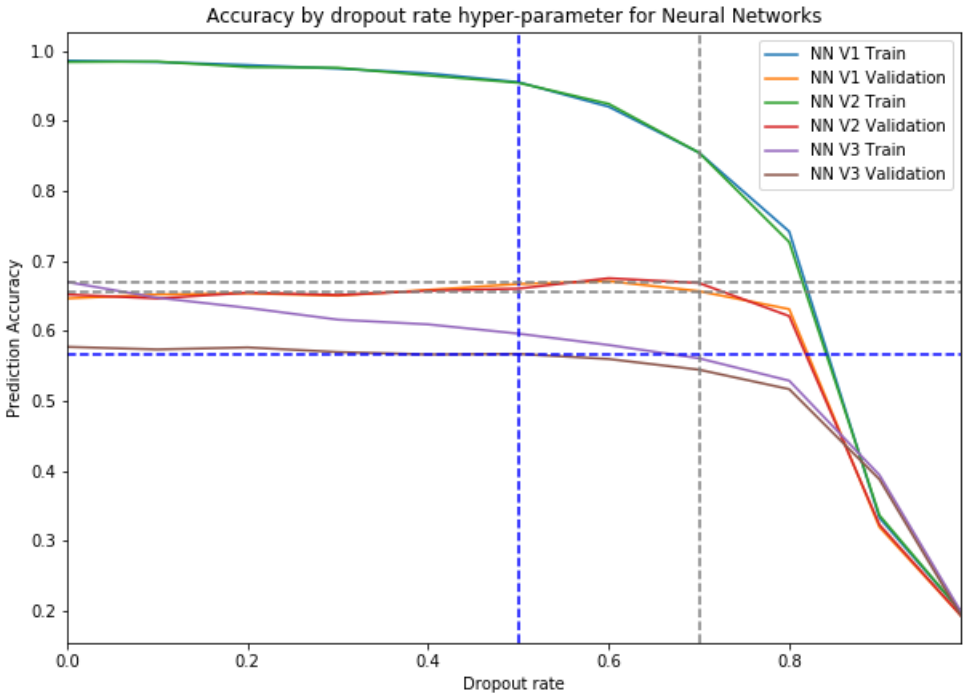


Figure 4.5: Accuracy of Neural network models for different values of dropout rate.

It can be seen that after a dropout rate of 0.7 the validation accuracy for the NN V1 and NN V2 classifiers starts to go down. The validation accuracy for the classifiers at the dropout rate of 0.7 was; 0.6571 for NN V1, and 0.6687 for NN V2. The gray dotted line indicates the value selected (dropout-rate=0.7) for the NN V1 and NN V2 classifiers.

For the NN V3 classifier, after the dropout rate of 0.5 the validation accuracy starts to decrease slowly. The classifier achieved a validation accuracy of 0.5674 at the 0.5 dropout rate. The blue dotted line indicates the value selected (dropout-rate=0.5) for the NN V3 classifier in the plot.

4.3 Using Age to improve prediction

This section investigates the impact of old bug reports and how effective can age of the bug reports be as a feature.

Removing old bug reports

Four subsets of the dataset were created using the date on which the bug report was registered, where each subset discards a part of the old bug reports. The first subset contains the entire data without any filtering, the second subset keeps all the bug reports registered after 2016-01-01, the third subset keeps all the bug reports registered after 2017-01-01 and the final subset keeps all the bug reports registered after 2018-01-01. The prediction of the classifiers on each of these subsets for different features is shown in Table 4.2.

		All data	After 2016	After 2017	After 2018
Logistic	LDA (140)	0.5464	0.5404	0.5355	0.5285
	TF-IDF (20,000)	0.6567	0.6567	0.6484	0.6287
	TF-IDF (20,000)+ LDA (140)	0.6606	0.6597	0.6499	0.6368
Neural Nets	LDA (140)	0.5435	0.5370	0.5363	0.5289
	TF-IDF (20,000)	0.6619	0.6593	0.6532	0.6517
	TF-IDF (20,000)+ LDA (140)	0.6694	0.6550	0.6541	0.6524
DO-probit	LDA (140)	0.5423	0.5400	0.5300	0.5201

Table 4.2: Validation accuracy for classifiers on different subsets of the data.

The validation accuracy for all the classifiers starts to decrease as the old bug reports were discarded. This thesis used the entire dataset to evaluate the classifiers in the following sections.

Using Age as a feature

The dataset contains bug reports from 2015-10-07. This was considered as the point of reference and the age of the bug reports were calculated by comparing it to the report registration date. The new bug reports get a larger value for this feature and the old bug reports get a smaller value. This is introduced as a feature to the multinomial logistic regression classifier and the change in the predictive performance of the classifier on the validation dataset is shown in Table 4.3.

	Predictive Performance		
Features	Without age	With age	Difference
LDA (140)	0.5348	0.5371	+0.0023
TF-IDF (20,000)	0.6593	0.6609	+0.0016
TF-IDF (20,000) + LDA (140)	0.6562	0.6591	+0.0029

Table 4.3: Validation accuracy for multinomial logistic regression classifier on using the age feature.

On using the age feature an $\approx$ 0.2% increase in the validation accuracy of the multinomial logistic regression classifier was observed. The coefficient for each class was visualized to find out which classes found the age feature to be important.

The coefficients for logistic regression with the LDA (140) + Age are visualized in Appendix A.7, A.8, A.9, and A.10. The coefficients for logistic regression with the TF-IDF (20,000) + Age are visualized in Appendix A.11, A.12, A.13, and A.14. The coefficients for logistic regression with the TF-IDF (20,000) + LDA (140) + Age are visualized in Appendix A.15, A.16,

A.17, and A.18. The coefficients for the textual features were represented in blue color and the coefficient for age feature was represented with red color in the visualization.

It can be seen from the plots that the classes *NR-EMCASW*, *NR-EMCASW-A*, and *TC-SW* found the age feature important and had large positive coefficients for age. Classes *G2-GRAT* and *G2-WRAT* also found age as an important feature, but age had a large negative coefficients for these classes.

4.4 Evaluating the Classifiers

In this section, the overall accuracy and the F1 score for each of the classifiers will be presented. Until now the training and the validation datasets were used to choose the hyper-parameters for the classifiers, now the test set which is 15% of the dataset was used to evaluate the classifiers.

The F1 score and accuracy of each class for the three trained classifiers can be found in Table 4.4. The classes in the table are ordered in descending order of the number of observations for them in the entire dataset. The classifiers were trained using only the textual features of the bug report, the age feature was discarded as it did not give much improvement.

The classifiers perform well for the larger classes, but the performance goes down for the smaller classes. Neural network classifiers get good accuracy and F1-score for the larger classes, but it finds it hard to classify the smaller classes like *BBI-BBISW*, *NR-EMCASW-A*, and *CPP-PLM*. The neural network models never got the prediction for the *BBI-BBISW*, and *CPP-PLM* class correct. The accuracy for the larger classes like *LTE-BBSW* and *CAT-SW* remained almost consistent across all the features. These classes had a low F1 score when the LDA (140) features were used and the F1-score improved when the TF-IDF (20,000) features were used.

It can be seen from Table 4.4 that the F1-score and the accuracy for each class improve when the TF-IDF (20,000) features were used. The performance of the logistic regression and the neural network classifiers improve when either TF-IDF (20,000) or a combination of the TF-IDF (20,000) and LDA (140) topics were used. The DO-probit classifier was only trained using the LDA (140) features due to computational limitations. The TF-IDF feature vectors were too large for the classifier to handle.

When using the combination of the TF-IDF (20,000) and the LDA features, the performance of the classifiers for most of the classes like *LTE-BBSW*, *CAT-SW*, *LTE-TCSW*, improves. The performance of some of the minority classes like *NR-EMCASW-A*, *CPP-PLM* goes down. The neural networks classifier improved its performance for the class *NR-EMCASW-A* when the TF-IDF (20,000) features were used, but the performance goes back to 0% accuracy for the class when the features are combined. The neural network classifier found it hard to classify bug reports from the classes *NR-EMCASW-A*, *CPP-PLM*, and *BBI-BBISW*. The classes *CPI*, *G2-GRAT*, and *LTE-TOOLS*, though being small classes, had good performance with the classifier.

Nr	Class	Model	LDA 140		TF-IDF (20,000)		TF-IDF (20,000) + LDA (140)		Baseline Performance	
			F1	Acc	F1	Acc	F1	Acc	F1	Acc
1	LTE-BBSW	Logictic	0.7028	0.8414	0.7335	0.8777	0.7475	0.8821	0.7465	0.8744
		DO-probit	0.7136	0.8491	NA	NA	NA	NA		
		Neural Network	0.7239	0.8766	0.7562	0.8612	0.7594	0.8832		
2	CAT-SW	Logistic	0.5518	0.6525	0.6424	0.6880	0.6472	0.6811	0.6593	0.6948
		DO-probit	0.5395	0.6001	NA	NA	NA	NA		
		Neural Network	0.5658	0.6294	0.6739	0.6730	0.6562	0.6280		
3	LTE-TCSW	Logistic	0.3802	0.3274	0.5015	0.4674	0.5084	0.4753	0.5212	0.4832
		DO-probit	0.3818	0.4012	NA	NA	NA	NA		
		Neural Network	0.4087	0.3688	0.5075	0.4970	0.5094	0.5049		
4	RADIOSW	Logistic	0.5328	0.4919	0.6798	0.6704	0.6867	0.6773	0.6767	0.6659
		DO-probit	0.5175	0.5129	NA	NA	NA	NA		
		Neural Network	0.5731	0.5469	0.6801	0.6933	0.6735	0.6681		
5	LTE-UEH	Logistic	0.4862	0.4341	0.6215	0.5581	0.6250	0.5813	0.6304	0.5555
		DO-probit	0.4679	0.4298	NA	NA	NA	NA		
		Neural Network	0.5178	0.4496	0.6307	0.5891	0.6361	0.5917		
6	RCS-DEV	Logistic	0.5127	0.4975	0.6545	0.6262	0.6487	0.6287	0.6666	0.6435
		DO-probit	0.5153	0.5056	NA	NA	NA	NA		
		Neural Network	0.5492	0.5247	0.6599	0.6460	0.6590	0.6410		
7	TC-SW	Logistic	0.5752	0.5830	0.6678	0.6440	0.6666	0.6542	0.6757	0.6711
		DO-probit	0.5853	0.5901	NA	NA	NA	NA		
		Neural Network	0.5909	0.5728	0.6580	0.6915	0.6491	0.6271		
8	G2-WRAT	Logistic	0.6142	0.5813	0.7800	0.8000	0.7727	0.7906	0.7919	0.8232
		DO-probit	0.6180	0.5673	NA	NA	NA	NA		
		Neural Network	0.6741	0.7023	0.7922	0.8604	0.7983	0.8837		
9	NR-EMCASW	Logistic	0.4709	0.4124	0.6355	0.5762	0.6246	0.5593	0.6483	0.5988
		DO-probit	0.4876	0.4269	NA	NA	NA	NA		
		Neural Network	0.5280	0.5593	0.6239	0.6327	0.5807	0.7005		
10	TN-SW	Logistic	0.4262	0.3679	0.5945	0.5188	0.6021	0.5283	0.6458	0.5849
		DO-probit	0.4157	0.3605	NA	NA	NA	NA		
		Neural Network	0.4444	0.3773	0.6413	0.5566	0.6421	0.5754		
11	CPI	Logistic	0.5164	0.5000	0.6987	0.6170	0.6941	0.6276	0.7485	0.6808
		DO-probit	0.4945	0.5132	NA	NA	NA	NA		
		Neural Network	0.5635	0.5425	0.7356	0.6808	0.7241	0.6702		
12	BBI-BBISW	Logistic	0.0000	0.0000	0.1132	0.0612	0.1296	0.0714	0.0909	0.0510
		DO-probit	0.0000	0.0000	NA	NA	NA	NA		
		Neural Network	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000		
13	G2-GRAT	Logistic	0.4380	0.3593	0.6206	0.5625	0.6101	0.5625	0.7076	0.7187
		DO-probit	0.3870	0.3209	NA	NA	NA	NA		
		Neural Network	0.4912	0.4375	0.6371	0.5625	0.7482	0.8125		
14	NR-EMCASW-A	Logistic	0.2168	0.1323	0.5849	0.4558	0.6000	0.4852	0.7000	0.6176
		DO-probit	0.2117	0.1021	NA	NA	NA	NA		
		Neural Network	0.0000	0.0000	0.4421	0.3088	0.0000	0.0000		
15	LTE-TOOLS	Logistic	0.5070	0.4090	0.6478	0.5227	0.6216	0.5227	0.7500	0.6818
		DO-probit	0.5310	0.4208	NA	NA	NA	NA		
		Neural Network	0.5084	0.3409	0.7435	0.6590	0.7654	0.7045		
16	CPP-PLM	Logistic	0.1754	0.1136	0.3174	0.2272	0.2622	0.1818	0.3076	0.2272
		DO-probit	0.1647	0.1082	NA	NA	NA	NA		
		Neural Network	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000		

Table 4.4: F1-score and accuracy for each class against the baseline performance.

The overall accuracy for the classifiers can be found in Table 4.5.

	LDA (140)		TF-IDF (20,000)		TF-IDF (20,000) + LDA (140)		Baseline performance	
Classifier	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy
Multinomial Logistic Regression	0.5272	0.5464	0.6401	0.6512	0.6437	0.6549	0.6562	0.6667
Neural Networks	0.5498	0.5724	0.6464	0.6612	0.6372	0.6558		

Table 4.5: Overall accuracy and f1-score for different classifiers against the baseline.

It can be seen that the baseline set by the SVM classifier; Accuracy: 0.6667 and F1-score:0.6562, was hard to match by the other classifiers. The neural network classifier using TF-IDF (20,000) features got close to the baseline performance, with an overall accuracy of 66.1% and an F1 score of 0.6464. Neural network classifiers got the best performance on all three sets of features, but there was not much difference when compared to the logistic regression classifiers.

Since the DO-probit classifier did not perform well with the LDA (140) features, and it could not be trained with the TF-IDF (20,000) features, it was discarded from Table 4.5. An increase of $\approx$ 10% in the overall accuracy was noticed when the classifiers started using the TF-IDF (20,000) features.

4.5 Visualization of the learned features

The learned TF-IDF and LDA features are visualized in this section. 100 random bug reports from each class were used to create the visualization. The feature vectors were reduced to 2 dimensions and visualized to see the separation between the classes.
Figure 4.6 shows the visualization of the LDA (140) feature vectors.

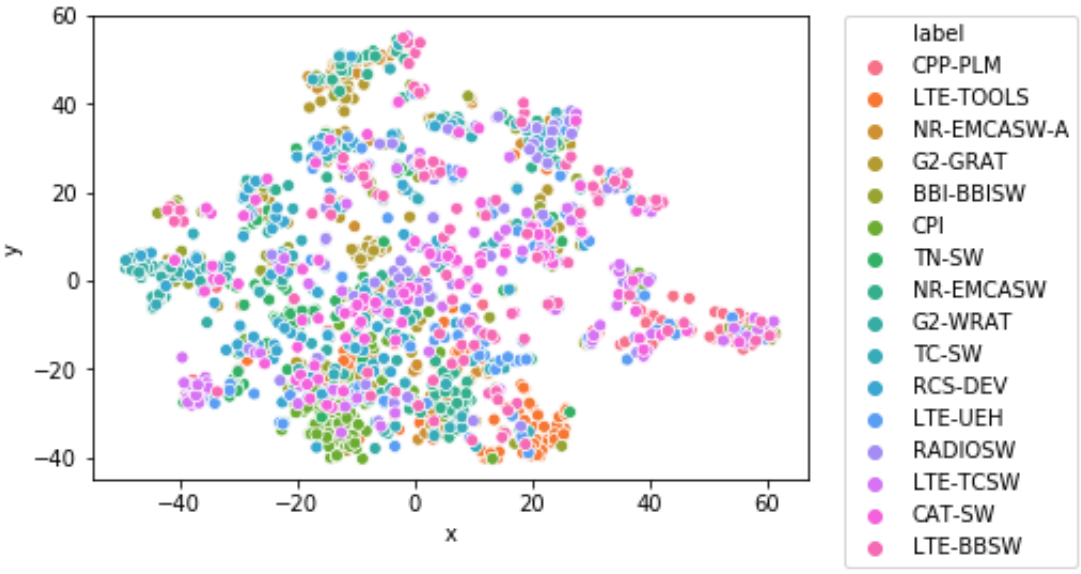


Figure 4.6: Visualization of LDA document vectors with 140 topics, for each class.

Figure 4.7 shows the visualization of the TF-IDF (20,000) feature vectors.

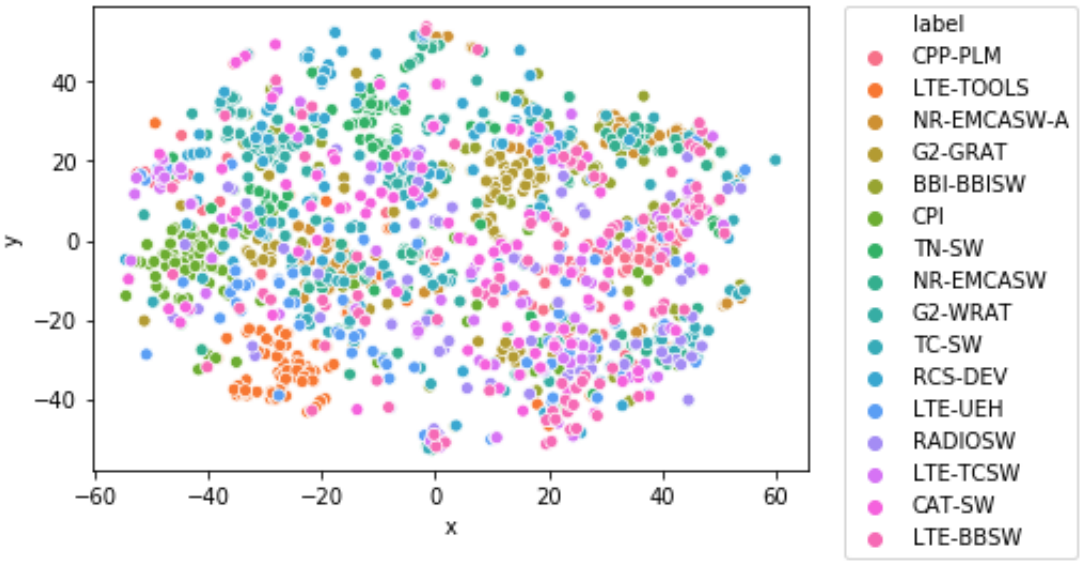


Figure 4.7: Visualization of TF-IDF(20,000) document vectors for each class.

Figure 4.8 shows the visualization of the TF-IDF (20,000) + LDA (140) feature vectors.

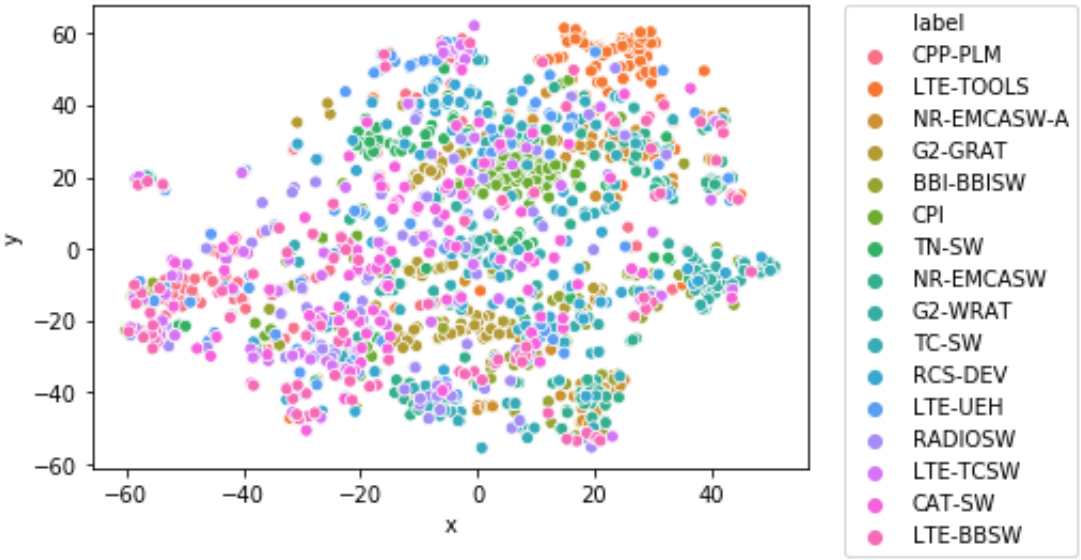


Figure 4.8: Visualization of TF-IDF(20,000) + LDA (140) document vectors for each class.

No clear separation boundaries can be seen from both the visualizations. The LDA vector visualization clearly shows that the larger classes are shadowing the smaller classes. The clusters formed are very compact hard to separate on 2-dimensional space. The visualization for the TF-IDF vectors is much more spread out and the classes can be distinguished better than the LDA visualization. This could be the reason the classifiers perform better when the TF-IDF (20000) features were used. For the concatenated features of TF-IDF (20,000) + LDA (140), the clusters look more compact in this plot, but the majority classes like *LTE-BBSW* and *CAT-SW* are still spread all over the plot.

5 Discussion

The results and the chosen methods are discussed in this chapter.

5.1 Results

Choice of number of topics for LDA

Previous studies [7] [12], suggested 40 or 100 topics for LDA when using DOLDA on the bug reports dataset. A grid of values in the range of 20 to 200 were chosen and the performance of the classifiers on the validation dataset was used to choose the number of topics. It can be seen from Figure 4.2 that the validation accuracy curve for the classifiers almost flattens after 140 topics, which is why 140 was selected as optimal value for the number of topics for LDA.

The validation curve for the DO-probit classifier is almost smooth, but for logistic regression and neural networks there are a lot of fluctuations in the performance. As the number of topics increases the topic probabilities get even more distributed, generating larger sparse vectors. The DO-probit classifier with the horseshoe prior was designed to handle large sparse vectors, which is why the curve looks smooth. The logistic regression and the neural network classifiers were not using any regularization at this point of the experiment, which could have been the reason for the fluctuations.

Choice of max features for TF-IDF

To set a limit on the vocabulary size to be used by TF-IDF, a grid of values in the range of 1000 to 30,000 were tested in the experiment. The results for the experiment shown in Figure 4.1, show rapid growth in the performance of the logistic regression and the neural network classifiers until 6000 max features. The growth slows down after that and almost flattens after 20,000 max features. The results from the neural network model show some fluctuations after 20,000 but crash back down. Therefore, 20,000 was selected as max features for TF-IDF. No regularization was being used by the logistic regression and the neural network classifier during this experiment, which could have been the reason for the fluctuations in the performance.

The DO-probit classifier was not used with the TF-IDF features due to computational limitations. The TF-IDF feature vectors were too large for the model, and limiting the size

of the TF-IDF vectors did not contain enough information for the bug reports. This raises a comparability issue for the classifiers, as DO-probit classifiers were only tested on the LDA topics. The training time for the DO-probit model with the number of topics can be seen in Figure A.19. The model takes approximately 2 hours to train using a vector of length 180. Therefore, the TF-IDF features were not tested on the DO-probit classifier.

Choice of α and C for logistic regression

Grid search was used to find the optimal regularization parameters for the logistic regression classifier. The range of promising values for α was identified first, to avoid the computational limitation, then grid search was used to find the optimal hyper-parameters. The logistic regression classifier using LDA had the best performance when $\alpha = 1$, which is equivalent to using L1 regularization. Since L1 regularization does variable selection and regularization, it is good for sparse vectors like the ones generated by LDA. This would help the classifier identify the topics that are important for a class.

The logistic regression classifiers using the TF-IDF feature vectors found 0.1 as the best value for α. Elastic net with $\alpha = 0.1$ indicates L1 regularization with weight 0.1 and L2 regularization with weight 0.9. L1 regularization is good for large sparse vectors, but for this dataset the classifier indicated a decrease in the validation performance when the weight for the L1 regularization was increased. This could have been because the classes had a lot of common words and the classifier found it hard to shrink the variables. L2 regularization does not reduce the number of variables but reduces the impact of each variable by reducing the coefficients.

Choice of dropout rate for neural networks

It can be seen from Figure 4.5 that without dropout the neural network classifiers, using TF-IDF (20,000) features, were over-fitting the training dataset. There was a difference of $\approx 30\%$ between the training and the validation accuracy. The dropout rate of 0.7 was selected for the TF-IDF features. For the large sparse feature vectors generated by TF-IDF, a large dropout rate is required to ensure the equal distribution of weights across the network and for the generalization of the classifier.

The neural network classifier using just the LDA (140) feature did not have much difference between the training and the validation accuracy. But a dropout rate of 0.5 was selected for the classifier as the validation accuracy started to decrease after that. A high dropout rate helps the network to learn more robust features.

A neural network classifier with 2 hidden layers was used in this thesis. The hidden layer 1 contained 300 neurons and the hidden layer 2 contained 100 neurons. Multiple configurations for the number of hidden units were tried, but increasing the number of hidden units over-fit the training data and did not improve the validation accuracy. Dropout performs model averaging as well, which is equivalent to training different architectures of neural networks and averaging the results.

Using age as a feature

It was observed that the validation accuracy for the classifiers was going down as the old bug reports were removed from the training dataset. This is a contradiction to the claims made in the studies [7] [12]. These studies were done in the year 2016 and 2017 respectively and a lot of new development has been carried out in Ericsson since then. The distribution of the data changes over time, the bug reports coming in for some 4G departments decrease while the bug reports for the 5G departments increase. Removing the old bug reports decreases the amount of training data for the old 4G departments and this could be the reason for the decrease in the performance of the classifiers. Classes like *G2-GRAT* and *G2-WRAT* had most

of the data collected before 2018. This is the reason the prediction accuracy for the classes go down when data before 2018 is discarded from the training dataset.

Using age as a feature gave $\approx$ 0.2% increase in the predictive performance of the classifiers. The coefficients of the logistic regression models were visualized; the classes *NR-EMCASW*, *NR-EMCASW-A*, *TC-SW* were identified to have large positive coefficient for age, and the classes *G2-GRAT*, *G2-WRAT* were identified to have a large negative coefficient for age.

The coefficients did identify an important pattern in the data, but the predictive performance of the classifiers was not improved by this. The visualization of the class distribution in the data over the years can be seen in Appendix A.1, A.2, A.3, A.4, A.5, and A.6. It can be seen from these figures that the classes *NR-EMCASW*, *NR-EMCASW-A*, *TC-SW* started to grow over the years (from 2015 to 2020). This is the reason these classes learned a large positive coefficient for age. The classes *G2-GRAT*, *G2-WRAT* were decreasing over the years, so the model learned a large negative coefficient for age in these classes. The age feature was not used in the final models as it did not give a significant improvement in the performance.

Prediction

The neural network classifier had the best performance among the classifiers trained in this thesis, though it was not able to match the performance of the baseline SVM classifier, the results were close to it. Using the TF-IDF (20,000) features gave an $\approx$ 10% increase in the overall accuracy and F1-score of the logistic regression and neural network classifiers. It could have been because of the better separation boundary between the classes created by the TF-IDF features, as shown in Figure 4.7. Comparing the confusion matrices of the SVM classifier (Appendix B.8), and the best Neural network classifier (Appendix B.6), it can be seen that the results are similar for the larger classes. The neural network classifier gets all predictions for some of the smaller classes like *BBI-BBISW*, and *CPP-PLM*, completely wrong. This could be because these classes are small and overlap a lot with the larger classes like *LTE-BBSW* and *CAT-SW*, and the neural network classifiers need a lot of data to learn a good separation.

The LDA visualization 4.6 shows clearly the domination of the larger classes in the plot, and it is confirmed by the confusion matrices for the classifiers, Appendix B.1, B.5 and B.4. The larger classes *LTE-BBSW* and *CAT-SW* are spread all over the plot, which is why most of the classes are misclassified as them. These are the two largest classes in the dataset which may be misleading the classifier. The visualization for the TF-IDF (20,000) features 4.7 shows that the clusters are not as overlapping as in the LDA (140) plot. This is reflected in the results as other classes see an increase in the accuracy and F1-score when the TF-IDF (20,000) features are used.

The classes *BBI-BBISW*, *NR-EMCASW-A* and *CPP-PLM* were found hard to classify by all the classifiers. It can be seen from the confusion matrices in Appendix B.2, B.6 for the classifiers, each of these classes were closely related to a larger class. The class *BBI-BBISW* had most of its reports misclassified as *LTE-BBSW*. This could have been because both of these classes are related to the *Base-band* department. The class *NR-EMCASW-A* had majority of its reports misclassified as *NR-EMCASW* or *TC-SW*. The class distribution plots for each year A.5 show that these are all new classes. Since there are not many bug reports from the class *NR-EMCASW-A*, the classifier makes a prediction towards the larger classes *TC-SW* or *NR-EMCASW*. A similar situation can be seen between the smaller class *CPP-PLM* and the larger class *CAT-SW*.

The *LTE-TOOLS* class, though being a small class, had a unique set of features, which is why a clear cluster was formed for the class, Figures 4.6, 4.7. This could be the reason all the classifiers were able to perform well in this class, with the neural network classifier having the best performance.

5.2 Method

The implementation of the bug classification task at Ericsson uses an SVM classifier [13], and the performance of the classifier on the test dataset was considered as a baseline for this thesis. The baseline set by the SVM classifier was hard to match for the classifiers trained in this thesis. This could have been the limitation introduced due to the choice of feature extraction methods used on the textual data. The semantic and syntactic features were ignored by the methods used for feature extraction in this thesis. Since, the bug reports description field is written by humans, extracting the semantic and syntactic features could give better separation between the classes.

One of the biggest challenges in this thesis was the unbalanced nature of the dataset. The entire dataset was used to create the training, validation, and test datasets used in this thesis. The larger classes like *LTE-BBSW*, *CAT-SW*, dominate the prediction as most of the other smaller classes are misclassified as them. No clear separation was found for these classes and they shadowed most of the other smaller classes which could be the reason for the misclassification. This problem could be handled by using over-sampling and under-sampling methods to create a somewhat balanced training dataset. An age feature was analyzed in this thesis, which did not prove to be useful in improving the performance of the classifiers, though it discovered some interesting patterns in the data. Other methods could be tried to utilize this feature, over-sampling the bug reports in the training dataset based on the age of the bug reports could help improve the performance of the classifier on the new classes introduced.

The TF-IDF features improved the performance of the classifiers, but the size of the TF-IDF feature vectors was a huge limitation in this thesis. Each bug report was represented as a feature vector of size 20,000 and this required a lot of memory when training. This is the reason cross-validation was not used in this thesis for evaluating the classifiers. Using *K-fold cross-validation* could be a better way to compare the performance of the classifiers, but it was avoided in this thesis due to computational limitations. To prevent the issue of comparability, cross-validation was not used for the classifiers using the LDA topics as well. Finding a more compressed feature representation for the bug reports could prove to be useful for the task.

This thesis tried to maintain a similar class distribution across the training, validation, and test data. Since the distribution of data changes over time, another approach that could have been tried is to create data splits based on time. The most recent bug reports could be used as the test dataset, and the remaining data to train the classifiers. This could give a better estimate of the classifier's future performance.

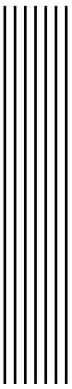
6 Conclusion

Is it possible to improve on the current baseline?

The features and the classifiers used in this thesis could not perform better than the baseline standards set by the SVM classifier. The overall accuracy of the neural networks model using the TF-IDF (20,000) features came closest to the baseline. The LDA topics were not too useful on the bug reports data as reflected by the performance of the classifiers. The TF-IDF features helped the classifiers identify some differentiating features for the classes and helped improve the performance for almost all the classes. A lot of overlap between the classes could still be observed from the visualization of the TF-IDF features. The features learned from the textual fields in this thesis were semantically weak, *DBRNN-A* could be used to learn syntactic and semantic features from long word sequences, which could create a better separation between the classes.

How can the age of a bug report be used to improve the prediction?

A contradiction to the claims about old bug reports was observed in this thesis, which is why no data was discarded based on the age of the bug report. The studies which made the claims were old and a lot of new data was collected since then, which could have been the reason for the contradiction. The age of the bug reports was introduced as a feature to the classifiers. Although the age feature helped the classifiers learn some interesting patterns in the data, it did not improve much on the predictive performance, and was discarded. This thesis tried just one way to utilize the age, other ways like oversampling the training dataset based on the age of the bug reports could be tried in future studies. This could improve the classifier's performance on the newly introduced classes.



Bibliography

[1] J.A. Cheyne, J.S.A. Carriere, D. Smilek. "Absent-mindedness Lapses of conscious awareness and everyday cognitive failures". In: *Consciousness and Cognition, Volume 15, Issue 3, pp. 578-592* (2006).

[2] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, P. Runeson. "Automated Bug Assignment: Ensemble-based Machine Learning in Large Scale Industrial Contexts". In: *Empir Software Eng 21, pp. 1533–1578* (2015).

[3] J. Ferzund, F. Wotawa, S. Ahsan. "Automatic Software Bug Triage System Based on Latent Semantic Indexing and Support Vector Machine". In: *4h International Conference on Software Engineering Advances, pp. 216-221* (2009).

[4] J. Anvik. "Assisting Bug Report Triage through Recommendation". In: *University of British Columbia Press, Volume 7* (2007).

[5] G. Jeong, S Kim, T. Zimmermann. "Improving bug triage with bug tossing graphs". In: *7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, New York, pp. 111-120* (2009).

[6] M. Senthil. "DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging". In: *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, pp. 171–179* (2018).

[7] M. Magunsson, L. Jonsson, M. Villani, D. Broman, K. Sandahl, S. Eldh. "Automatic Localization of Bugs to Faulty Components in Large Scale Software Systems using Bayesian Classification". In: *IEEE International Conference on Software Quality, Reliability and Security (QRS), Vienna, pp. 423-430* (2016).

[8] M. Magunsson, L. Jonsson, M. Villani. "DOLDA - A regularized supervised topic model for high-dimensional multi-class regression". In: *Computational Statistics 35, pp. 175–201* (2016).

[9] D. M. Blei, Andrew Y. Ng., M. I. Jordan. "Latent Dirichlet Allocation". In: *Journal of Machine Learning Research, Issue 3, pp. 993-1022* (2003).

[10] J. E. Johndrow, K. Lum, D. B. Dunson. "Diagonal Orthant Multinomial Probit Models". In: *Journal of Machine Learning Research, Issue 31, pp. 29-38* (2013).

[11] D. Artchounin. "Tuning of machine learning algorithms for automatic bug assignment". In: *Dissertation DIVA, pp. 135-281* (2017).

[12] C. Svahn. "Automated Bug Report Routing". In: *Dissertation DIVA, pp. 49-107* (2017).

[13] V. Apostolidis, K. Lioufi. "SVM classification with linear and RBF kernels". In: *Machine Learning research 2.1, pp. 3351-4083* (2015).

[14] C.M. Bishop. "Pattern recognition and machine learning". In: *12th ed. Springer, pp. 32–33, 197–209, 179-220, 225-284, 685–691* (2006).

[15] R. Rehurek and P. Sojka. "Gensim: Software Framework for Topic Modelling with Large Corpora". In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (2010).

[16] C. Sammut, G.I. Webb. "TF–IDF". In: *Encyclopedia of Machine Learning. Springer, Boston, MA* (2011).

[17] F. Varoquaux, G. Gramfort, A. Michel, V. Thirion, B. Grisel, O. Blondel, M. Prettenhofer, P. Weiss, R. Dubourg, V. Vanderplas, J. Passos, A. Cournapeau, D. Brucher, M. Perrot, M. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research, Volume 12, pp. 2825-2830* (2011).

[18] A. Defazio, F. Bach, S.L. Julien. "SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives". In: *Advances in Neural Information Processing Systems, Volume 2* (2014).

[19] R. Tibshirani. "Regression shrinkage and selection via the Lasso". In: *Journal of the Royal Statistical Society, B 58.1, pp. 267–288* (1996).

[20] A. Hoerland, R.Kennard. "Ridge regression". In: *Encyclopedia of Statistical Sciences 8.1, pp. 129–136* (1988).

[21] H. Zouand, T.Hastie. "Regularization and Variable Selection via the ElasticNet". In: *Journal of the Royal Statistical Society B 67.2, pp. 301–320* (2005).

[22] C. M. Carvalho, N. G. Polson, and J. G. Scott. "The horseshoe estimator for sparse signals". In: *Biometrika 97.2, pp. 465–480* (2010).

[23] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from over-fitting". In: *The Journal of Machine Learning Research 15.56, pp. 19291958* (2014).

[24] M. Abadi, A. Agarwal et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association, pp. 265-283* (2015).

A Figures

Distribution of data over the years

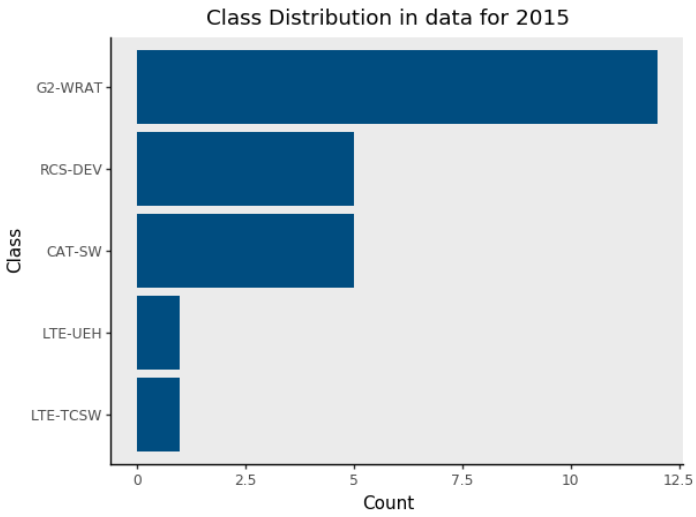


Figure A.1: Distribution of data in 2015

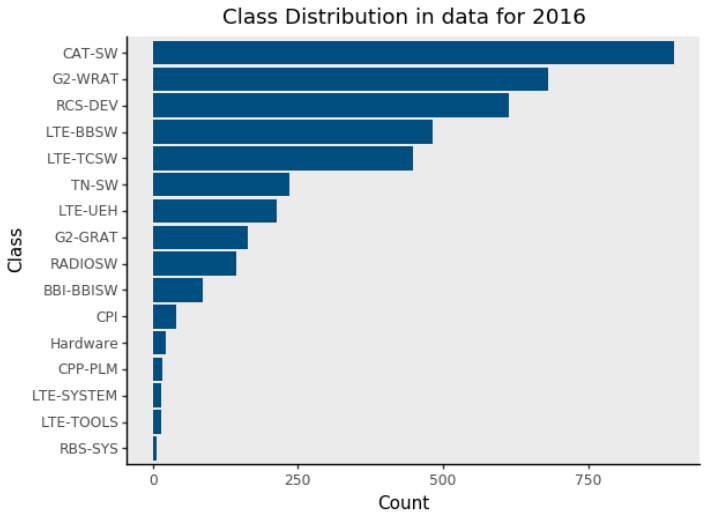


Figure A.2: Distribution of data in 2016

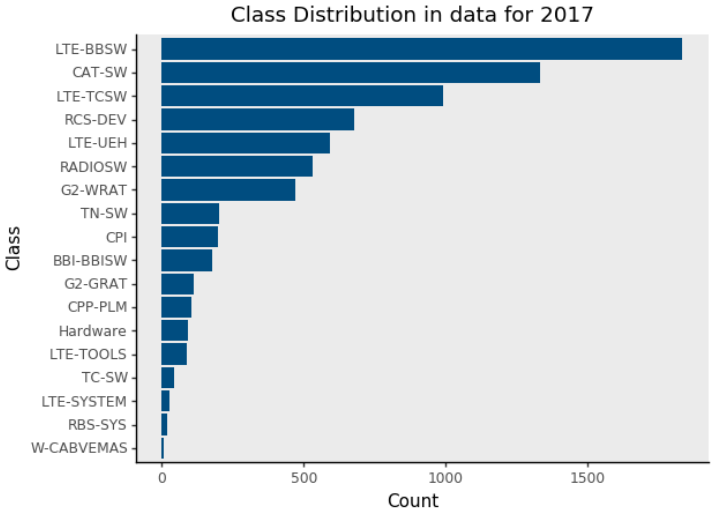


Figure A.3: Distribution of data in 2017

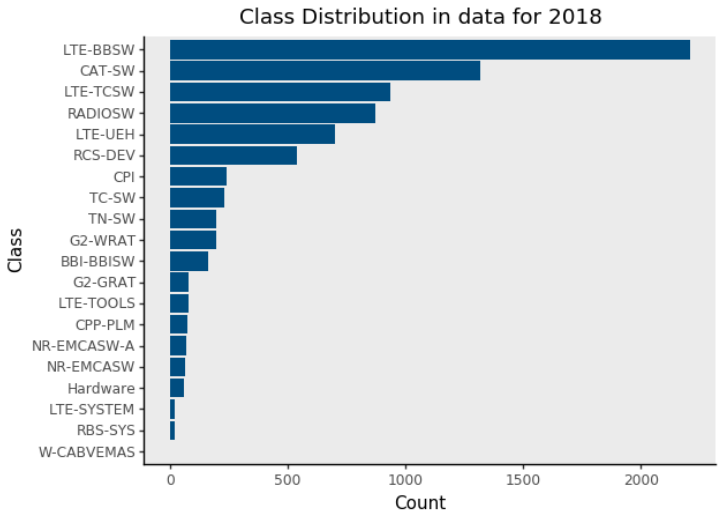


Figure A.4: Distribution of data in 2018

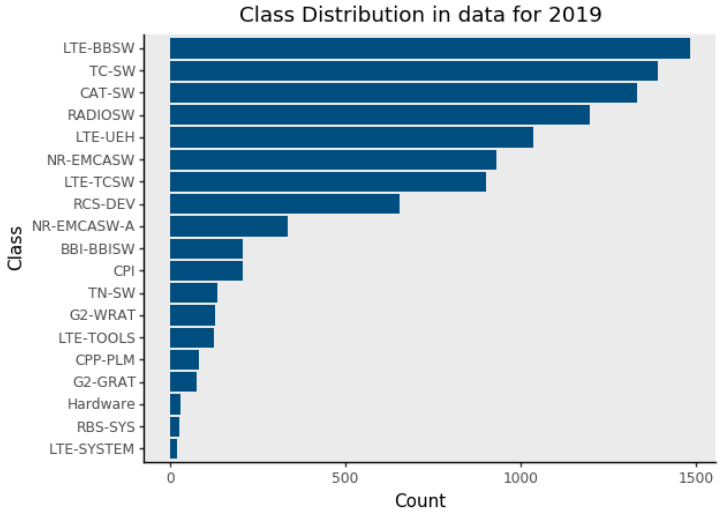


Figure A.5: Distribution of data in 2019

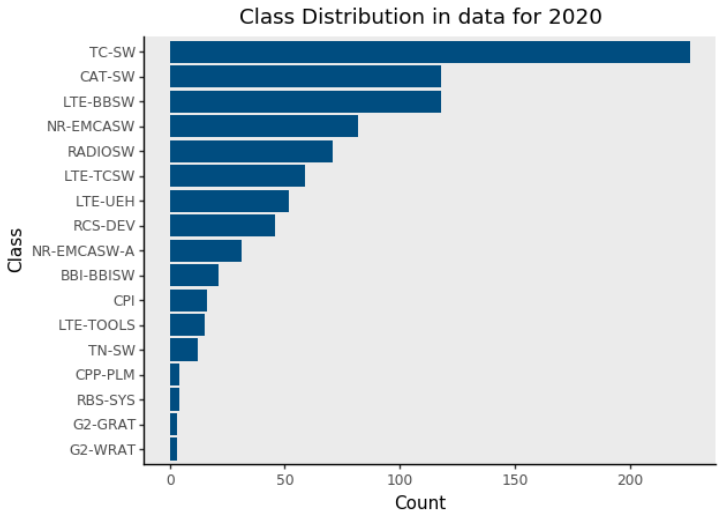


Figure A.6: Distribution of data in 2020

Visualizing logistic regression coefficients to see importance of age feature

LDA (140) + Age

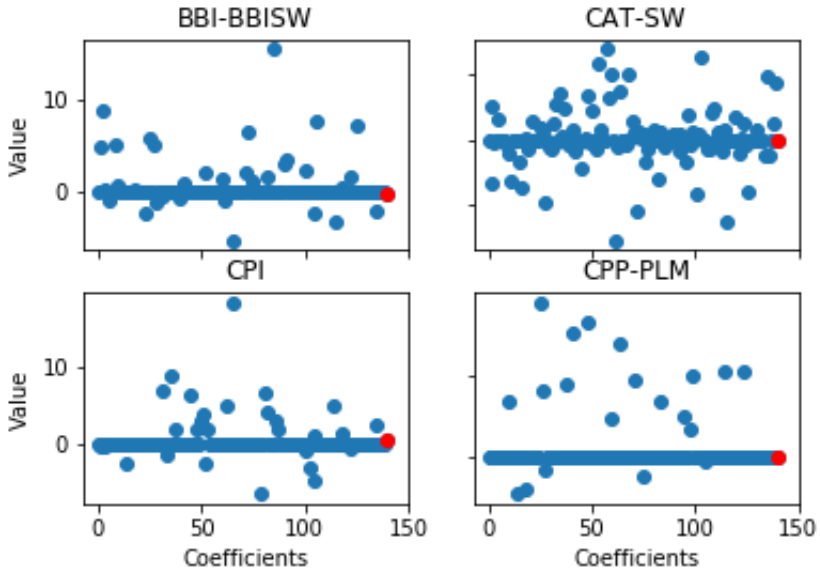


Figure A.7: Visualization of coefficients for logistic regression with LDA (140)+Age

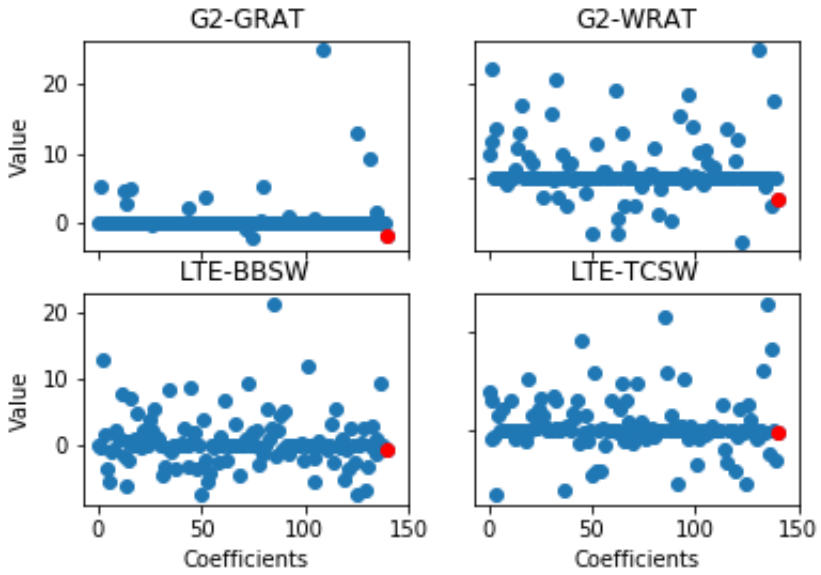


Figure A.8: Visualization of coefficients for logistic regression with LDA (140)+Age

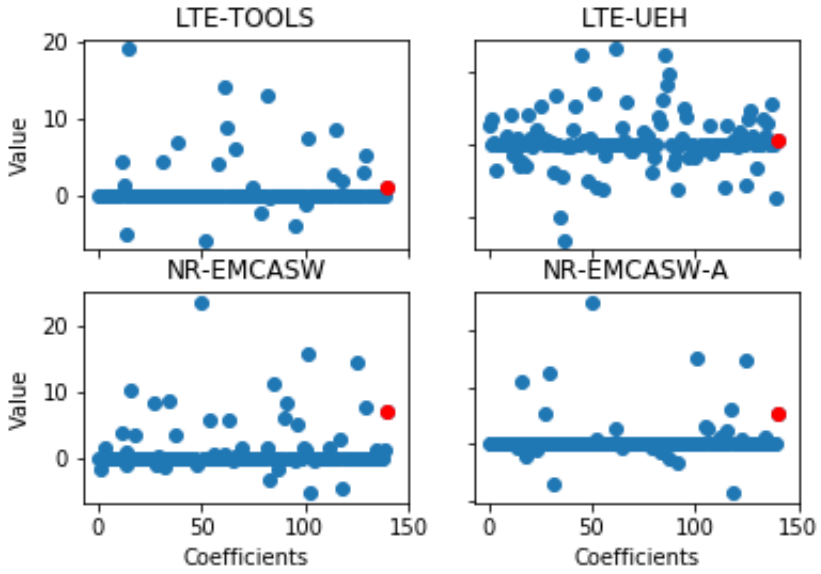


Figure A.9: Visualization of coefficients for logistic regression with LDA (140)+Age

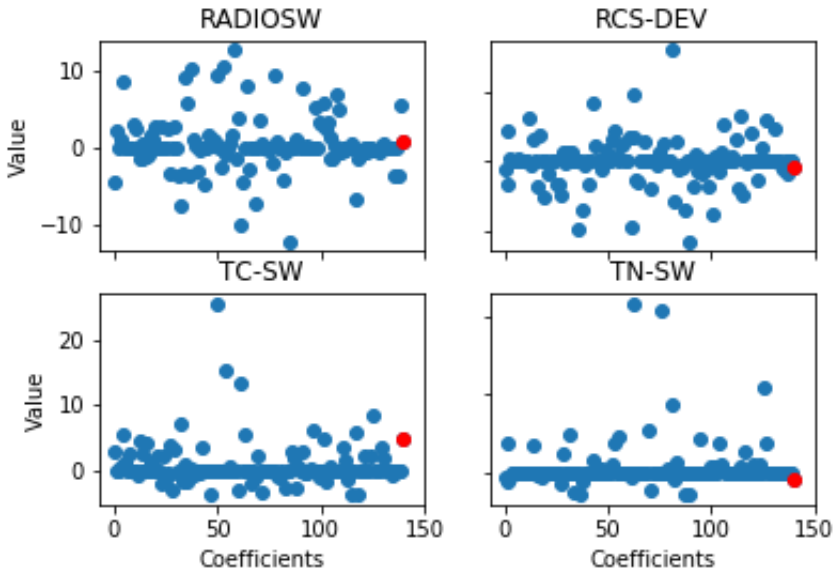


Figure A.10: Visualization of coefficients for logistic regression with LDA (140)+Age

TF-IDF (20,000) + Age

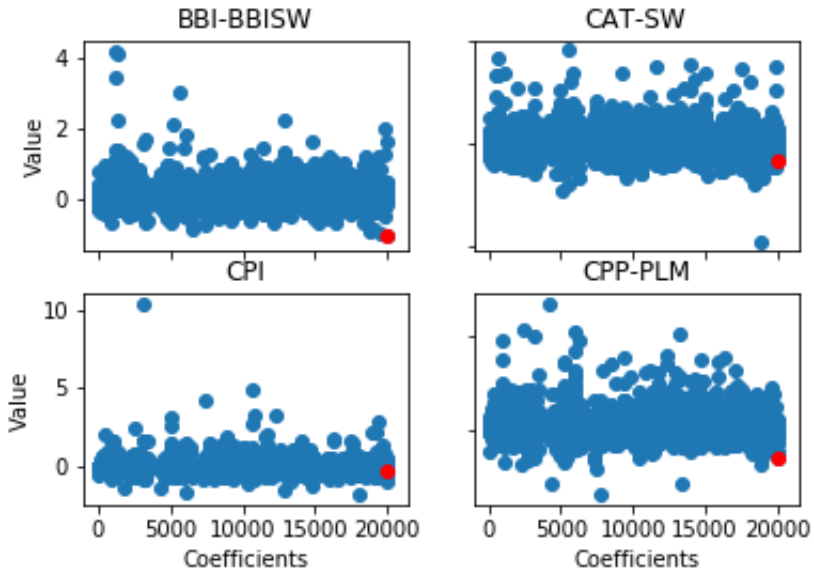


Figure A.11: Visualization of coefficients for logistic regression with TF-IDF (20,000)+Age

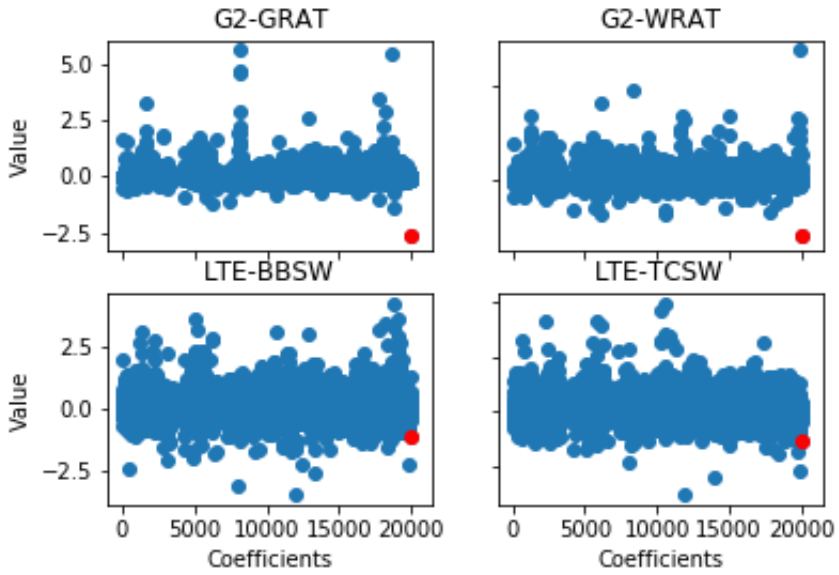


Figure A.12: Visualization of coefficients for logistic regression with TF-IDF (20,000)+Age

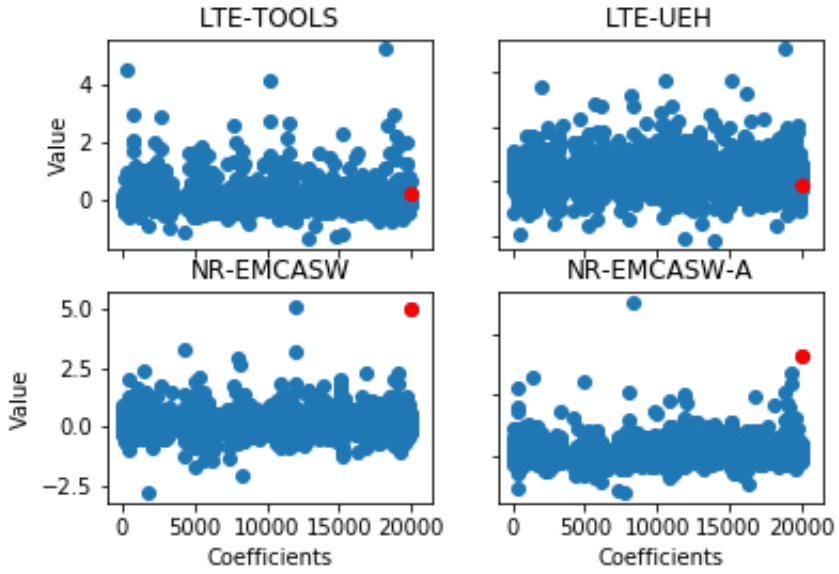


Figure A.13: Visualization of coefficients for logistic regression with TF-IDF (20,000)+Age

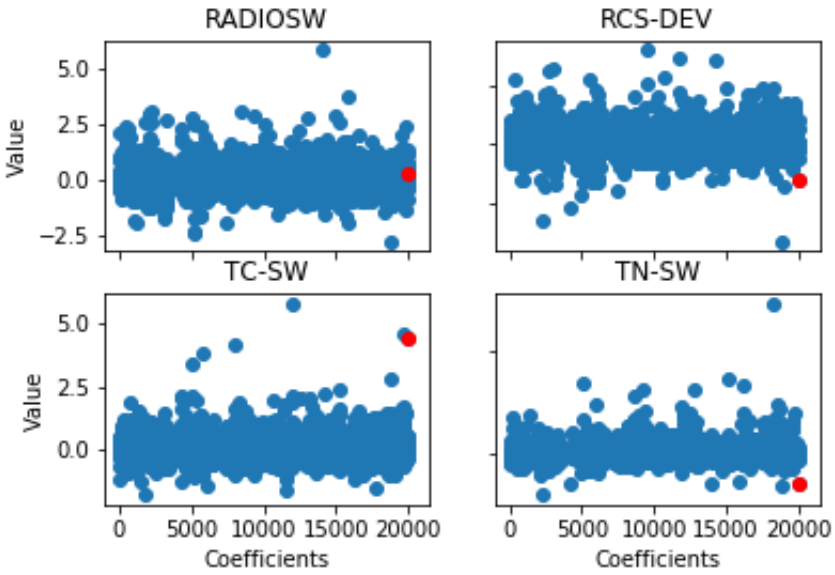


Figure A.14: Visualization of coefficients for logistic regression with TF-IDF (20,000)+Age

TF-IDF (20,000) +LDA (140) + Age

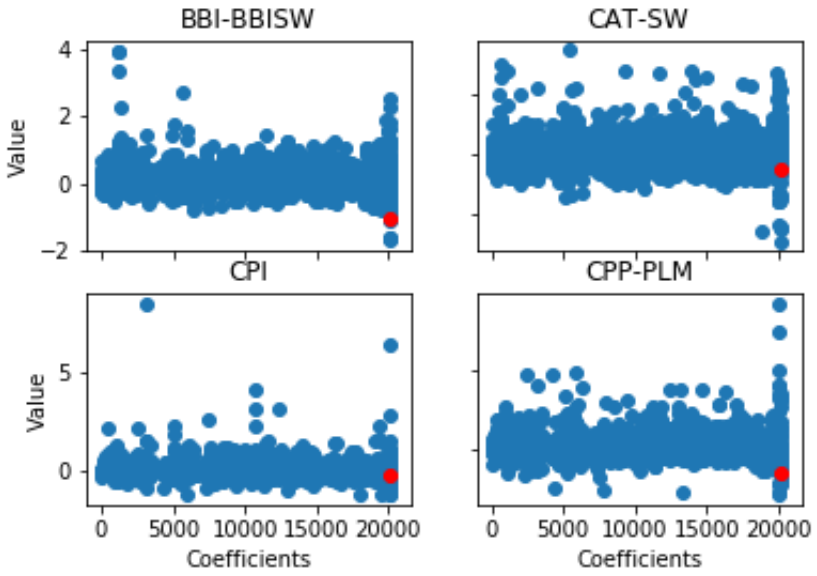


Figure A.15: Visualization of coefficients for logistic regression with TF-IDF (20,000)+LDA (140)+Age

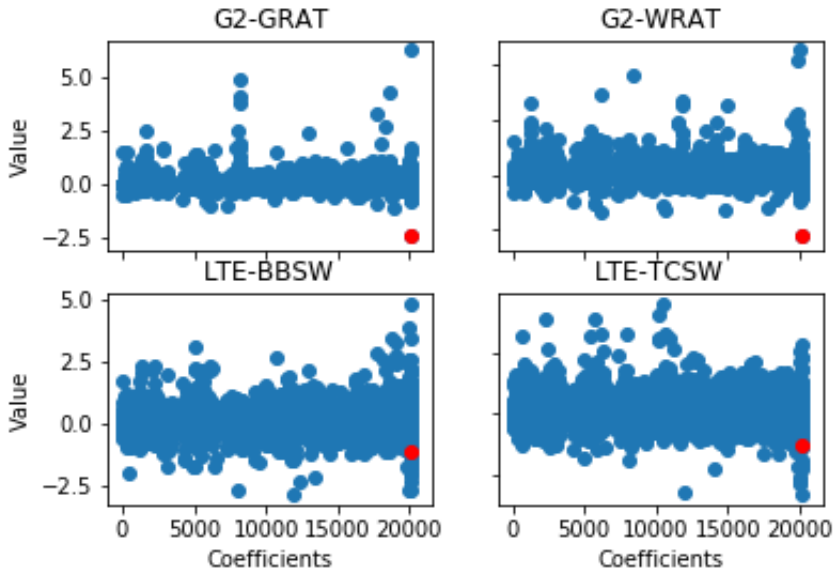


Figure A.16: Visualization of coefficients for logistic regression with TF-IDF (20,000)+LDA (140)+Age

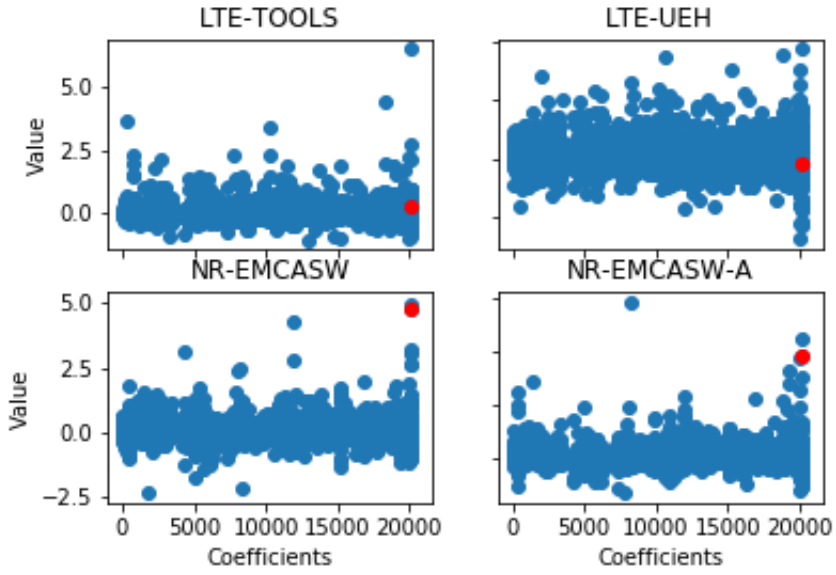


Figure A.17: Visualization of coefficients for logistic regression with TF-IDF (20,000)+LDA (140)+Age

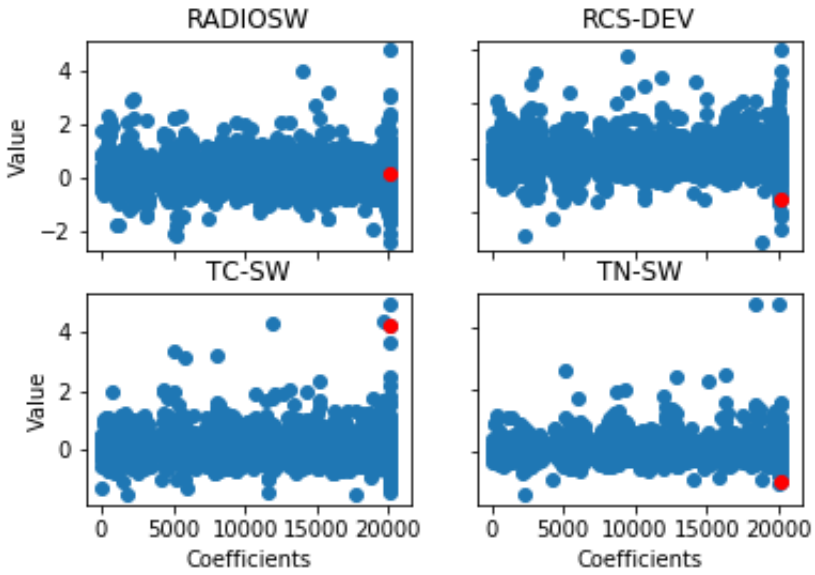


Figure A.18: Visualization of coefficients for logistic regression with TF-IDF (20,000)+LDA (140)+Age

Other figures

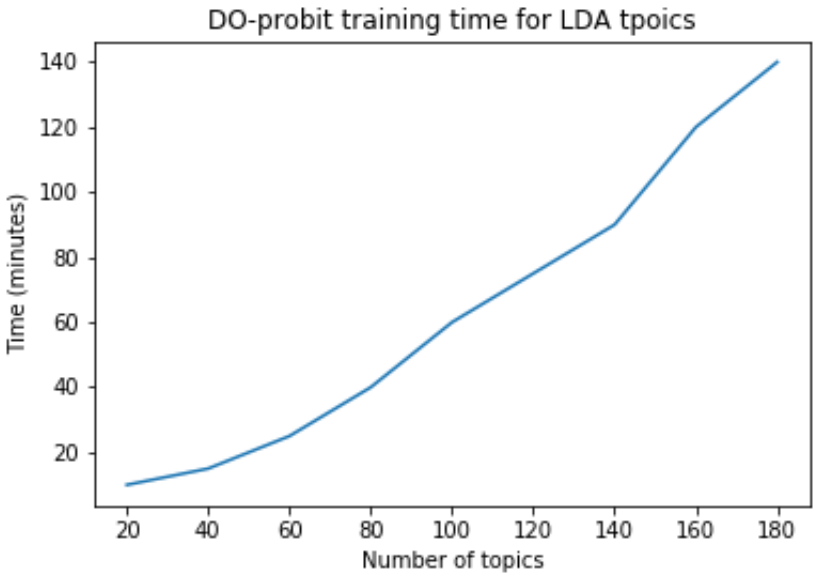


Figure A.19: DO-probit classifier training time with number of topics.

B Confusion matrices

Class Notation:

1. BBI-BBISW
2. CAT-SW
3. CPI
4. CPP-PLM
5. G2-GRAT
6. G2-WRAT
7. LTE-BBSW
8. LTE-TCSW
9. LTE-TOOLS
10. LTE-UEH
11. NR-EMCASW
12. NR-EMCASW-A
13. RADIOSW
14. RCS-DEV
15. TC-SW
16. TN-SW

Logistic

		Predicted Class															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	0	12	0	0	1	5	57	5	0	0	0	0	5	6	4	3
	2	1	479	8	1	6	21	51	37	1	7	1	0	56	46	14	5
	3	0	13	47	0	0	0	11	11	0	6	0	0	1	2	2	1
	4	0	19	0	5	0	0	6	10	0	2	0	0	0	2	0	0
	5	0	16	0	0	23	3	3	3	0	2	2	0	3	9	0	0
	6	1	47	1	0	1	125	12	3	0	6	0	0	3	16	0	0
True	7	0	29	4	1	1	6	764	29	3	42	2	0	16	9	2	0
	8	0	97	12	4	0	2	134	166	3	37	0	0	22	24	1	5
Class	9	0	1	1	0	0	0	16	2	18	3	1	0	0	1	1	0
	10	0	37	6	2	0	4	92	52	1	168	0	0	4	9	11	1
	11	1	2	0	0	0	0	26	0	0	0	73	4	15	5	50	1
	12	0	1	0	0	1	0	11	0	0	0	21	9	2	0	23	0
	13	0	128	1	0	2	4	41	9	0	5	10	0	215	7	14	1
	14	1	82	6	0	6	16	19	21	0	8	0	0	15	201	9	20
	15	1	25	2	0	0	2	18	6	1	15	23	2	12	15	172	1
	16	0	14	0	0	0	4	5	12	0	3	0	0	1	28	0	39

Table B.1: Confusion Matrix for LDA (140) topics, Logistic Regression.

		Predicted Class															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	6	6	0	1	0	7	56	6	0	0	3	0	5	4	2	2
	2	0	505	4	3	6	18	53	35	1	1	0	0	48	41	17	2
	3	0	5	58	0	0	0	11	9	0	4	0	0	2	3	1	1
	4	0	16	1	10	0	0	4	11	0	1	0	0	0	1	0	0
	5	0	16	0	0	36	1	3	5	0	0	0	0	1	2	0	0
	6	1	22	0	0	1	172	6	3	0	1	0	0	1	8	0	0
True	7	1	17	0	1	0	4	797	40	1	30	0	0	11	6	0	0
	8	0	68	3	1	0	1	125	237	0	34	0	0	21	12	2	3
Class	9	0	3	0	0	0	0	11	0	23	3	1	0	1	2	0	0
	10	0	10	2	3	1	0	83	56	1	216	0	0	4	4	7	0
	11	0	4	0	0	0	0	27	1	0	0	102	1	12	2	28	0
	12	0	1	0	0	0	0	10	0	0	0	9	31	3	0	14	0
	13	0	77	0	0	1	3	36	6	0	1	10	2	293	2	6	0
	14	0	61	3	0	7	14	20	16	0	1	0	0	6	253	7	16
	15	0	20	1	0	0	1	15	6	1	13	19	4	17	8	190	0
	16	0	7	0	0	0	5	8	7	0	3	0	0	0	21	0	55

Table B.2: Confusion Matrix for TF-IDF (20,000), Logistic Regression.

		Predicted Class															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	7	7	0	1	0	7	54	7	0	0	4	0	3	4	2	2
	2	0	500	5	3	7	19	47	37	1	4	1	0	48	43	16	3
	3	0	5	59	0	0	0	10	10	0	4	0	0	2	3	1	0
	4	0	19	0	8	0	0	4	10	0	2	0	0	0	1	0	0
	5	0	16	0	0	36	1	2	4	0	0	0	0	1	4	0	0
	6	1	23	0	0	1	170	6	3	0	1	0	0	3	7	0	0
True	7	1	13	1	1	0	4	801	34	3	35	0	0	9	6	0	0
	8	0	59	3	1	0	2	121	241	0	40	0	0	24	12	2	2
Class	9	0	0	0	0	0	0	13	1	23	3	1	0	1	2	0	0
	10	0	9	2	3	1	0	75	56	2	225	0	0	2	5	7	0
	11	0	2	0	0	0	0	21	2	0	0	99	4	11	2	36	0
	12	0	1	0	0	0	0	10	0	0	0	7	33	3	0	14	0
	13	0	75	1	0	1	3	31	7	0	1	8	2	296	5	7	0
	14	1	56	4	0	8	14	17	18	0	2	0	0	7	254	6	17
	15	0	19	1	0	0	1	16	4	1	13	20	3	15	9	193	0
	16	0	7	0	0	0	4	7	7	0	3	0	0	0	22	0	56

Table B.3: Confusion Matrix for TF-IDF (20,000) + LDA (140) topics, Logistic Regression.

DO-probit

		Predicted Class															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	2	11	458	13	20	12	40	25	94	1	31	1	2	124	80	21	16
	3	0	7	51	0	0	1	4	14	1	7	0	0	0	5	2	0
	4	0	3	0	4	0	0	1	4	0	2	0	0	0	0	0	0
	5	0	5	0	0	25	1	1	0	0	0	0	1	3	4	0	0
	6	5	27	0	0	6	141	6	6	0	3	0	0	3	19	1	4
True	7	59	71	12	7	5	10	778	142	14	100	35	14	45	21	20	3
	8	5	23	7	8	2	2	20	155	1	47	0	0	5	15	4	9
Class	9	0	1	0	0	0	0	4	1	20	0	0	0	0	0	0	0
	10	0	7	6	2	2	2	36	38	2	162	0	0	9	6	17	4
	11	3	0	0	0	0	0	2	1	1	0	72	17	6	1	26	0
	12	0	0	0	0	0	0	0	0	1	2	4	10	1	0	3	0
	13	3	56	1	1	2	3	16	19	1	8	12	3	223	19	14	2
	14	6	51	3	2	10	14	9	27	0	9	3	0	6	202	13	23
	15	4	17	1	0	0	1	6	1	1	16	49	21	11	11	174	0
	16	2	6	0	0	0	0	0	4	1	0	1	0	1	21	0	45

Table B.4: Confusion Matrix for 140 topics, DO-probit.

Neural Networks

		Predicted Class															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	0	10	0	0	1	4	57	8	0	0	2	0	5	7	2	2
	2	0	462	7	0	7	28	41	58	0	0	1	0	62	46	15	7
	3	0	12	51	0	0	0	9	10	0	7	0	0	2	2	1	0
	4	0	16	0	0	0	0	6	19	0	1	0	0	0	2	0	0
	5	0	14	0	0	28	5	4	1	0	0	1	0	1	10	0	0
	6	0	34	0	0	2	151	11	3	0	2	0	0	2	10	0	0
True	7	0	23	3	0	1	7	796	23	0	35	4	0	11	4	1	0
	8	0	85	10	0	0	3	138	187	0	42	0	0	19	20	0	3
Class	9	0	1	1	0	0	0	21	1	15	2	1	0	0	0	1	1
	10	0	25	6	0	0	7	99	53	0	174	0	0	3	8	12	0
	11	0	1	0	0	1	0	22	0	0	0	99	0	11	3	40	0
	12	0	0	0	0	0	0	8	0	0	0	41	0	2	2	15	0
	13	0	109	1	0	3	4	41	9	0	3	9	0	239	5	13	1
	14	0	72	5	0	7	18	20	19	0	4	1	0	18	212	8	20
	15	0	17	2	0	0	2	17	5	0	13	39	0	21	10	169	0
	16	0	18	1	0	0	4	1	12	0	2	0	0	1	27	0	40

Table B.5: Confusion Matrix for LDA (140) topics, Neural Networks.

		Predicted Class															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	0	4	0	0	0	9	61	9	0	0	4	0	4	3	3	1
	2	0	494	3	0	4	24	31	48	1	4	0	0	62	45	16	2
	3	0	2	64	0	0	0	8	10	0	4	0	0	2	1	2	1
	4	0	11	1	0	0	0	5	25	0	1	0	0	0	1	0	0
	5	0	12	0	0	36	4	2	0	0	1	0	0	1	8	0	0
	6	0	12	0	0	0	185	4	2	0	0	1	0	2	9	0	0
True	7	0	11	0	0	0	1	782	47	1	40	2	0	16	7	1	0
	8	0	50	3	0	0	3	119	252	0	39	0	0	22	12	5	2
Class	9	0	0	0	0	0	0	8	1	29	2	1	0	1	1	1	0
	10	0	7	2	0	1	1	72	59	2	228	0	0	1	4	10	0
	11	0	1	0	0	0	0	9	0	0	0	112	2	10	1	42	0
	12	0	0	0	0	0	0	2	0	0	0	26	21	1	1	17	0
	13	0	69	1	0	1	3	28	5	0	2	9	0	303	3	13	0
	14	0	46	5	0	7	18	17	16	0	2	1	0	8	261	10	13
	15	0	9	1	0	0	1	10	4	1	9	26	4	20	6	204	0
	16	0	4	0	0	0	3	2	8	0	4	0	0	1	24	1	59

Table B.6: Confusion Matrix for TF-IDF (20000), Neural Networks.

		Predicted Class															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	0	2	0	0	1	9	61	7	0	1	4	0	4	5	3	1
	2	0	461	4	0	7	28	35	58	1	3	0	0	61	56	16	4
	3	0	1	63	0	0	1	8	10	0	5	0	0	2	1	2	1
	4	0	14	1	0	0	0	6	21	0	1	0	0	0	1	0	0
	5	0	2	0	0	52	3	1	1	0	0	0	0	1	4	0	0
	6	0	10	1	0	1	190	4	2	0	0	1	0	1	4	1	0
True	7	0	7	0	0	0	2	802	45	2	31	3	0	11	4	1	0
	8	0	43	3	0	1	1	123	256	0	42	1	0	21	13	1	2
Class	9	0	0	0	0	0	0	8	2	31	1	1	0	1	0	0	0
	10	0	3	2	0	1	0	74	61	2	229	0	0	2	4	9	0
	11	0	0	0	0	0	0	13	0	0	0	124	0	7	1	32	0
	12	0	0	1	0	0	0	0	0	0	0	54	0	4	0	9	0
	13	0	69	0	0	2	3	39	7	0	3	15	0	292	2	5	0
	14	2	41	4	0	10	20	18	17	0	1	2	0	5	259	10	15
	15	0	11	1	0	0	1	9	3	1	13	45	0	18	8	185	0
	16	0	7	0	0	0	3	3	8	0	3	0	0	0	20	1	61

Table B.7: Confusion Matrix for TF-IDF (20,000) + LDA (140) topics, Neural Networks.

Baseline SVM

		Predicted Class															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	5	8	0	1	0	7	54	7	0	0	3	0	3	4	3	3
	2	0	510	4	5	7	19	44	29	1	2	1	0	47	44	17	4
	3	0	4	64	0	0	0	10	8	0	3	0	0	1	3	1	0
	4	0	18	0	10	0	0	4	10	0	1	0	0	0	1	0	0
	5	0	9	0	0	46	1	2	1	0	0	0	0	1	4	0	0
	6	2	22	0	0	1	177	4	2	0	0	0	0	1	6	0	0
True	7	3	16	0	1	0	2	794	36	3	28	1	0	15	8	1	0
	8	0	68	3	3	0	1	124	245	0	28	0	0	18	14	2	1
Class	9	0	0	0	0	0	0	10	0	30	2	1	0	1	0	0	0
	10	0	9	2	1	1	0	85	62	1	215	0	0	2	1	7	1
	11	0	2	0	0	0	0	12	2	0	0	106	4	14	1	36	0
	12	0	0	0	0	0	0	2	0	0	0	6	42	5	0	13	0
	13	0	76	0	0	1	3	36	6	0	1	7	3	291	6	7	0
	14	2	52	3	0	10	16	18	14	0	1	0	0	8	260	5	15
	15	0	13	1	0	0	1	13	5	1	12	25	3	16	7	198	0
	16	0	6	0	0	0	5	7	6	0	2	0	0	0	17	1	62

Table B.8: Confusion Matrix for baseline SVM classifier.