

April 2017

Intelligent Customer Support Case Routing

Joseph William Perez
Worcester Polytechnic Institute

Ziyan Ding
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

Repository Citation

Perez, J. W., & Ding, Z. (2017). *Intelligent Customer Support Case Routing*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/1191>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.



WPI JUNIPER[®]
NETWORKS

Intelligent Customer Support Case Routing

A MAJOR QUALIFYING PROJECT SUBMITTED TO THE FACULTY OF
WORCESTER POLYTECHNIC INSTITUTE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF BACHELOR OF SCIENCE

April 18, 2017

Submitted By:

Ziyan Ding
Joseph W. Perez
Xuanzhe Wang

Submitted To:

Professor Mark Claypool
Worcester Polytechnic Institute

Mr. Steven Tufts
Mr. Aditya Sood
Mr. Amogh Rao
Juniper Networks

This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see <http://www.wpi.edu/Academics/Projects>.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 2 | Background | 4 |
| 2.1 | Scripting | 4 |
| 2.1.1 | Python | 4 |
| 2.1.2 | JavaScript | 4 |
| 2.1.3 | JavaScript NodeJs | 5 |
| 2.2 | Database Technologies | 5 |
| 2.2.1 | MongoDB | 5 |
| 2.2.2 | Hadoop | 6 |
| 2.3 | Machine Learning | 6 |
| 2.3.1 | Categories of Machine Learning | 7 |
| 2.3.2 | Classification Model Evaluation | 7 |
| 2.3.3 | Deep Learning | 9 |
| 2.3.4 | Loss Function | 12 |
| 2.3.5 | Natural Language Processing | 14 |
| 2.3.6 | Feature Encoding | 15 |
| 2.4 | Machine Learning Technologies | 16 |
| 2.4.1 | Python | 16 |
| 2.4.2 | Spark | 17 |
| 2.4.3 | TensorFlow | 17 |
| 2.4.4 | Keras | 18 |
| 2.4.5 | Scikit-Learn | 18 |
| 2.4.6 | Numpy and Scipy | 19 |
| 3 | Methodology and Implementation | 20 |
| 3.1 | Requirements | 20 |
| 3.2 | Workflow Overview | 21 |
| 3.3 | Understanding Previous Intern Work | 22 |
| 3.4 | End-to-End Pipeline Design | 23 |
| 3.5 | Getting Raw Data | 24 |
| 3.5.1 | Robomongo | 24 |
| 3.5.2 | Details | 24 |
| 3.5.3 | Notes | 25 |
| 3.6 | Understanding the Data | 25 |
| 3.6.1 | Categorical Features | 25 |
| 3.6.2 | Visualizing the Categorical Data | 26 |
| 3.7 | Cleaning Raw Details | 26 |
| 3.7.1 | Removing Duplicate Records | 26 |
| 3.7.2 | Flattening Details | 26 |
| 3.7.3 | Flattening Notes | 28 |
| 3.8 | Converting from objects to tables | 30 |
| 3.8.1 | Details | 31 |

| | | |
|----------|---|-----------|
| 3.9 | A Focus on Details | 31 |
| 3.9.1 | Encoding | 32 |
| 3.9.2 | Label Selection | 32 |
| 3.10 | Modeling | 32 |
| 3.10.1 | Javascript Analysis | 33 |
| 3.10.2 | Spark Machine Learning | 33 |
| 3.10.3 | Keras with Tensorflow | 35 |
| 3.11 | Evaluating the Models | 36 |
| 3.11.1 | Data Split | 37 |
| 3.11.2 | Measurements for Keras Model | 39 |
| 3.12 | Webapp Demonstration | 44 |
| 3.12.1 | Backend Server | 44 |
| 3.12.2 | API Endpoints | 45 |
| 3.12.3 | Frontend | 45 |
| 4 | Results | 48 |
| 4.1 | Case Data Analysis | 48 |
| 4.1.1 | Feature Distribution | 48 |
| 4.1.2 | Label Distribution | 50 |
| 4.1.3 | Case Note Insights | 52 |
| 4.2 | Model Performance | 53 |
| 4.2.1 | Keras Models | 53 |
| 4.2.2 | Keras Model unbalanced_groupName | 55 |
| 4.2.3 | Keras Model balanced_groupName | 61 |
| 4.2.4 | Keras Model unbalanced_engineerName | 67 |
| 4.2.5 | Keras Model balanced_engineerName | 72 |
| 4.2.6 | Spark Model | 78 |
| 4.2.7 | Models Summary | 81 |
| 5 | Conclusion | 83 |
| 5.1 | Conclusions | 83 |
| 5.2 | Future Work | 85 |
| 5.2.1 | Goals | 85 |
| 5.2.2 | Suggested Work | 86 |
| | References | 88 |

Abstract

Juniper Networks is an American company selling networking hardware and software. When a customer needs assistance with a product, a customer support case is opened. The case can go through many stages before being resolved. The route a case takes to get to the resolver is currently human-controlled. This project aimed to assist case routing through the use of machine learning. We developed a system that can use a pipeline of data including previously resolved cases to model the relationship between a case and the engineer responsible for solving the problem. We used a deep learning neural network to create a system that could predict both the resolving group and engineer achieving up to 96% accuracy when predicting the 10 most likely groups, and 42% when predicting the 10 most likely engineers. We built a web application for future demonstration of this system.

1 Introduction

Juniper Networks is an international company based in Sunnyvale, CA, providing networking hardware and software to customers around the world (Juniper Networks, 2017). Juniper Networks provides the services for other enterprises that make up the Internet and with a large amount of offerings and strong customer base, Juniper Networks, provides a variety of customer support. Customer support at Juniper involves a range of services, including training, maintenance, and troubleshooting. When a customer seeks help, a new customer support case is opened. By opening a case, a tracked issue is established and will receive attention in order to be resolved.

At Juniper Networks, there are many engineers available to troubleshoot customer needs. Currently, there are often many steps between a case being opened and eventually being resolved. The case could go to any number of employees trying to gather information, presenting possible solutions, or finding others who may be able to solve the issue. This flow of a case, or case route, is controlled by humans with little to no automation. However the structure of the route can lend itself well to automation with the help of machine learning models. Machine learning is a growing field of techniques used to enable computers to find patterns where they are not explicitly programmed to look (Sas.com, 2016). Basically, using machine learning, a problem with defined input can be modeled and understood without a human programmer knowing the explicit relationship ahead of time.

This paper details our project aiming to understand and address human-based customer support case routing. The goal of this project was to analyze and model the current system in such a way that a prototype application could make suggestions for what engineer or group could solve a given case. We explain background context, our

approach to the problem, measurements of our success, and how future exploration could be done. We used a deep learning neural network to train a model with 8 months of data. The trained model was able to predict 1 group out of 124 with 51% accuracy, and 1 engineer out of 941 with 21% accuracy. Further we also evaluated prediction accuracy for the top 3, 5, 7 and 10 predictions. Predicting the top 10 groups out of 146 yielded a 96% accuracy and the top 10 engineers out of 941 yielded a 42% accuracy.

The rest of this report is organized as follows. Section 2 details all of the technologies, techniques and assessments used in our approach. Next, in Section 3 we describe exactly what we did in order to achieve our goal. In Section 4 we present our results, and what implications they had for measuring our success. Lastly, we summarize the project and detail the next steps that can be taken to improve or expand our work.

2 Background

This section is a collection of all the information necessary to understand the context of this project. This provides context to the task presented, the solution proposed and the results of the project. Topics in this section include discussion of scripting technologies, database technologies machine learning, and machine learning related technologies.

2.1 Scripting

To manage large data sets manually would be difficult and inefficient. Given that, we have explored some scripting languages that can enable us to work on different parts of the project. These languages and tools are useful for managing data, in addition to powering a tool for viewing and analyzing data from a front end focused application.

2.1.1 Python

Python is an interpreted scripting language which is designed to be easy to learn and use. Python has many data science and numerical computing toolkits available. Therefore, Python is a good choice for data science related projects. Additionally, python has been used in the construction of web applications on massive scales, like YouTube (Continuum.io, 2017). On top of that many motion picture companies like Sony Dreamworks and Disney have used python to manage clusters of computers for image generation.

2.1.2 JavaScript

JavaScript is another interpreted language. JavaScript uses "just-in-time" compilation technology to make it fast at run-time. Mostly, JavaScript is used with browsers as a web

front end scripting language. JavaScript can also be used for server-side scripting with frameworks like NodeJs (Rauschmayer, 2014). Given this, JavaScript is a good language to create some data visualization tools for internal use, likely as a web based application. JavaScript-based web applications require no installation or configuration for the end user.

2.1.3 JavaScript NodeJs

NodeJs is a JavaScript runtime used to run JavaScript outside of a browser (NodeJs Foundation, 2017). It provides a means of running JavaScript locally, through the use a command line tool.

2.2 Database Technologies

Because the project involves working with large sets of data it is important to understand some of the tools involved to retrieve, manage, and store such data.

2.2.1 MongoDB

MongoDB is a NoSQL database, with no strictly defined data structure for each entity (i.e. Schema-less) (Peterse, 2015). Each entity's structure can be changed after it is created. This makes MongoDB easy to use, and makes the structure malleable. MongoDB officially provides drivers for compatibility with most popular programming languages on Linux, Windows, and MacOS. One important MongoDB feature is load shedding. This means that when MongoDB deals with a massive amount of data, it might drop some records randomly in exchange for speed. Given this, MongoDB is not recommended for data where strong reliability is a priority. Some developers have gone so far as to suggest that MongoDB should only be used if every piece of data is equally

unimportant. Interestingly, this issue with MongoDB makes it an appropriate tool for data science experiments. In machine learning, losing several pieces of data should not influence the whole model to a significant degree.(Sirer, 2013).

2.2.2 Hadoop

Hadoop is an open source, Java based framework designed for the storage and processing of extremely large data sets over a distributed set of computers (Rouse, 2016). Hadoop is part of the Apache project by the Apache Software Foundation. This framework uses mapReduce to process these large data sets very quickly. In simple terms, rather than trying to move data over the network to be processed, mapReduce is more akin to bringing the processing to the data. The mapReduce functionality works by breaking everything up into smaller parts. Hadoop by itself is complex and generally difficult to set up, but there are a growing number of consumer tools to ease setup. Hadoop uses its own file system called HDFS (Hadoop Distributed File System). HDFS is a distributed file system that enables high performance access to data across a cluster of machines (Rouse, 2013). This is particularly useful for extremely large data sets that cannot fit on a single machine.

2.3 Machine Learning

Machine learning is a means of analyzing data sets to automate building analytic models. Using algorithms that can learn from data provided to it, machine learning can enable computers to discern new insights without already being programmed where to find them (Sas.com, 2016).

2.3.1 Categories of Machine Learning

There are 4 popular forms of general machine learning: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

Supervised learning is done by giving the model a set of pre-labeled data, or data that has a known association to the goal of the model. The algorithm will take in the labeled input, then compare its generated output, and change the model accordingly (Sas.com, 2016).

Unsupervised Learning uses only unlabeled data which has no known connection to the goal. The algorithm is not aware of a correct output, so it is mostly trying to find some structure in the data (Sas.com, 2016).

Semi-supervised Learning is done by using a small set of labeled data to start the training, and then a large set of unlabeled data afterwards. The unlabeled data is much less expensive, computationally and takes much less effort to acquire (Sas.com, 2016).

Reinforcement Learning is when the given algorithm learns through trial and error which actions have the best results. This is done with 3 primary components: the agent (the learner), the environment that the agent acts in, and the agent's possible actions. The agent will perform all its actions and record its results with respect to a given goal (Sas.com, 2016).

2.3.2 Classification Model Evaluation

Classification and Regression Supervised Learning problems are further divided into Classification and Regression problems. Classification predicts categories. Regression

predicts numeric values.

Accuracy Accuracy is a ratio of correct predictions to the number of total predictions made. For example, imagine a model is trying to classify 100 fruits, and we know that there are only 90 apples and 10 oranges. The model then predicts that all 100 of the fruits are apples. The model has a 90% accuracy.

Recall High accuracy alone is not a strong indicator of a good model. This is evident because a model can be biased toward one category. Recall is the measurement used to identify that bias. Recall is measured for each value in the true labels, and is a ratio how many times it predicted that value correctly, compared to how many times it predicted that value total. In the previous example with apples and oranges, the recall for apples is 100%. This is because out of the 90 actual apples, the model correctly identified all of them. Contrastingly, the recall for oranges is 0%. This is because of the 10 actual oranges, none of them were identified correctly.

Precision Precision is most easily understood as the ratio of correct predictions for a given label to total number of predictions given this label. For example, imagine there is a set of fruits, and it is known that set consists of 50 apples, and 50 oranges. A classifier is asked to predict each fruit. The classifier then predicts that there are 75 apples and 50 of them are actually apples, and it predicts there are 25 oranges and 25 of them are actually oranges. Given this, the precision for apples is 67% because the classifier predicted 75 apples, but only 50 of them are actually apples. The precision for oranges is 100% because of the 25 predictions for oranges, all of them are actually oranges.

Confusion Matrix A confusion matrix is a way of displaying the label and prediction for each values, and can help derive precision and recall visually. Consider a new

example with apples and oranges:

| Value | Predicted Apple | Predicted Orange |
|--------------|-----------------|------------------|
| Label Apple | 90 | 20 |
| Label Orange | 10 | 50 |

Table 1: an example confusion matrix

The above confusion matrix (Table 1) tells that there are $(90 + 20 = 110)$ apples and $(10 + 50 = 60)$ oranges. The model predicts 90 out of 110 apples are apples and 10 out of 60 oranges are apples. The model predicts 20 out of 110 apples are oranges and 50 of 60 oranges are oranges. Given this, it can be seen that the precision of apple predictions is $90/(90+10) = 90\%$. Also, the precision of orange predictions is $50/(50+20) = 5/7$. The recall for the apple true label is $90/(90+20) = 9/11$. The recall for the orange true label is $50/(50+10) = 5/6$.

A confusion matrix can be used to evaluate whether a model has bias towards any categories and also the performance on each individual categories because it makes it easier to see precision and recall.

2.3.3 Deep Learning

Deep Learning is an approach done by using neural networks as computational architecture to achieve machine learning goals. Essentially, machine learning is about function approximation, and deep neural networking is just a powerful approximation tool.

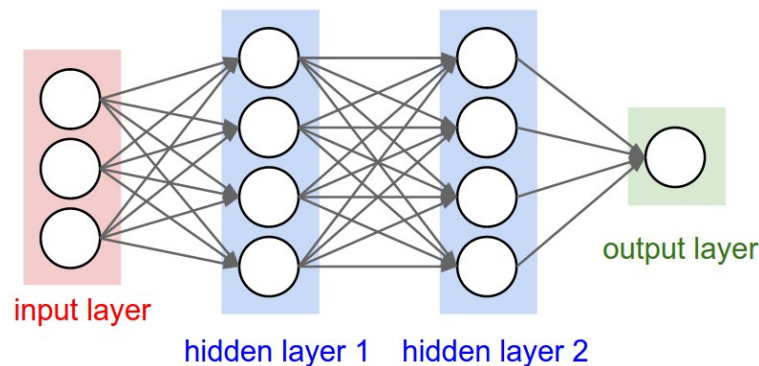


Figure 1: A abstract view of a neural network
(Karpathy, 2015)

Figure 1 demonstrates a deep neural network with 4 total layers. A layer contains multiple nodes that each perform a slightly different operation. This example network takes 3 numbers as input and produces only one number as an output.

Fully Connected Neural Networks Figure 1 shows a fully connected neural network, which is also referred to as a densely connected network. In a fully connected neural network, every node in each layer is connected to all the nodes in all adjacent layers. As figure 1 demonstrates, every node in hidden layer 1 is connected to each node in the input layer as well as each node in hidden layer 2.

Activation Function An activation function is simply a function that takes its input and computes an output. An activation function can be differentiable or non-differentiable, and is not limited to any specific functions. Additionally, the activation function of each node can be different from the activation function of every other node. The input of any given node is just the dot product of all values and weights of all inward edges plus the bias of a node.

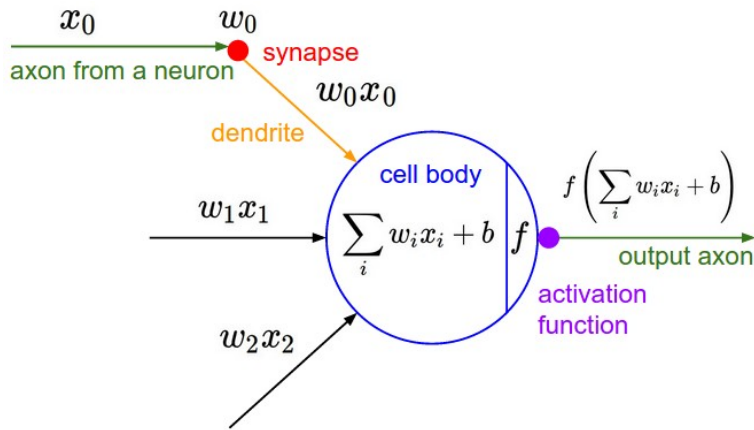


Figure 2: The structure of a neuron node (Karpathy, 2015)

Figure 2 shows the structure of a given node and how an activation function is used to manipulate input to produce a single output. For a conventional fully connected layer, nodes in the same layer have the same activation function.

Three popular activation functions are **sigmoid** (Figure 3), **tanh** (Figure 4), and **ReLU** (Figure 5).

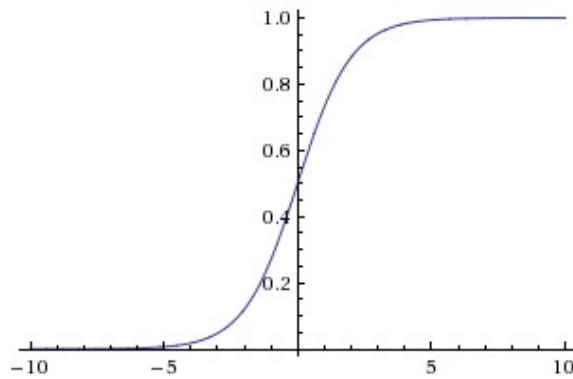


Figure 3: Sigmoid (Karpathy, 2015)

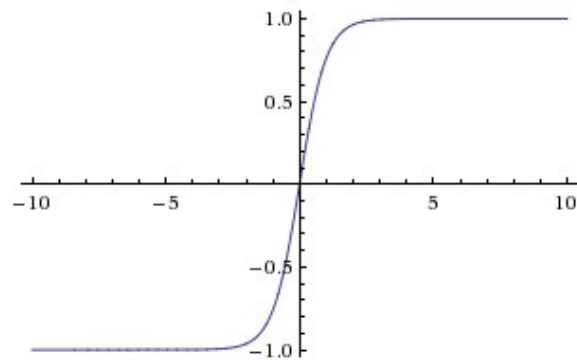


Figure 4: Tanh
(Karpathy, 2015)

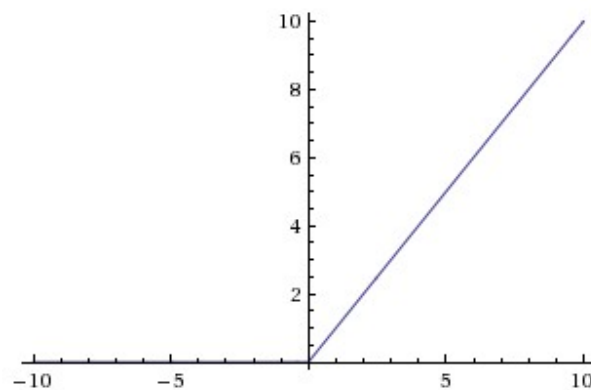


Figure 5: Rectified Linear Unit
(Karpathy, 2015)

Because ReLU has shown a significant improvement over other 2 activation functions, ReLU is the default choice (Krizhevsky, Sutskever, & Hinton, 2012).

2.3.4 Loss Function

A machine learning problem is basically a numerical optimization problem. All numerical optimization problems aim to minimize the output of a loss function. Loss functions are also sometimes referred to as error functions.

Loss functions in a regression problem In a regression problem, a loss function computes the distance between a predicted value and targeted value. For example, if in one prediction the true targeted value is 1, and the predicted value is 10. We can use 2 different loss functions. The first function is called L1 loss, which is the "absolute difference function", or "absolute error function". It is in the form:

$$\text{Loss} = |\text{true value} - \text{predicted value}|$$

The second function is called L2 loss, which is the "difference squared function" or "squared error function". It is in the form:

$$\text{Loss} = (\text{true value} - \text{predicted value})^2$$

In this example, L1 loss is calculated to be 9 and L2 loss is 81. A difference based loss function gives the model a sense of how far it is from the optimal solution.

Loss functions in classification problems A conventional classification problem will usually use accuracy, precision, recall and an F1-score as metrics, but a machine learning algorithm knows nothing inherently about these metrics. Therefore, a good loss function should have a high negative correlation to these metrics.

Cross Entropy Loss Function This function evaluates the difference between two distributions.

$$L_i = -\log(e^{f_{y_i}} / \sum_j (e^{f_j}))$$

Or equivalently,

$$L_i = -\log\left(\sum_j (e^{f_i})\right)$$

A distribution can be represented as a vector, where all values of the vector sum to one. For example if we have (0.2, 0.7, 0.1) and (0.1, 0.7, 0.2), the function can compute the difference between the two distributions. (Karpathy, 2016)

Softmax Classifier A softmax classifier is a fully connected neural network of which the activation function of the last layer is the softmax function.

A Softmax function converts a K-dimensional vector \mathbf{z} of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range (0, 1) that add up to 1. The function is given by:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

Figure 6: Softmax Function

A Softmax classifier uses cross entropy loss function.

2.3.5 Natural Language Processing

Natural Language Processing (NLP), is the means of a computer program understanding human speech as it is spoken (Rouse, 2011). NLP is often modeled as a semi-supervised learning problem due to the kinds of data it can be given. A Recurrent Neural Network (RNN) is basically a neural network with a form of memory; It focuses on sequential data and uses previous data to help understand the current data being processed (wildml.com, 2015). An RNN is a good architecture for natural language

processing because of how it uses sequential data. For instance, knowing the previous words in a sentence can help to predict and understand the next word.

Word2Vec Word2Vec is a unsupervised learning method that maps words in natural languages to a vector space where the vector representations of words reserve their semantics.(Mikolov, Sutskever, Chen, Corrado, & Dean, 2013)

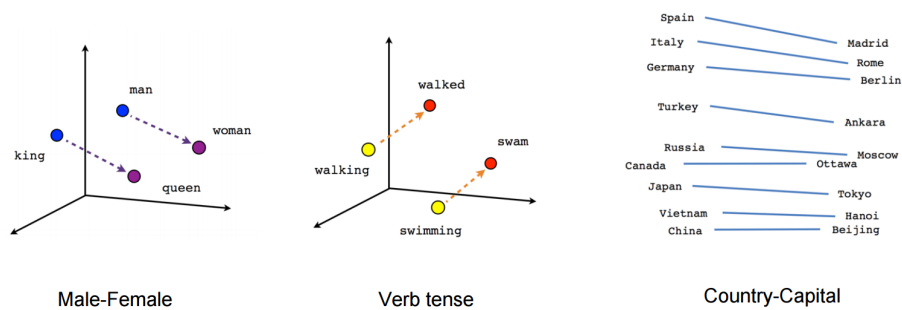


Figure 7: Linear relationships of semantic meaning in word
(Tensorflow.org, 2016)

Figure 7 show some examples of how semantic relationships are represented in a multidimensional vector space. What this means is that through learning, the algorithm derives semantic meaning of a word, so words with close semantic meaning, or words with similar meaning should be closer in the vector space. It also means the relationship between 2 words can be captured by a vector ($\text{word}_2 - \text{word}_1$). In figure 7 it is seen that the words "king" and "queen" are both somewhat near "man" and woman" and the difference between the two words of each set is very similar in both length and direction.

2.3.6 Feature Encoding

Because machine learning algorithms are based heavily in numerical analysis, and a lot of features are categorical in nature, it is important to create a meaningful numerical

representation of the important features. This is done by creating a mapping with which to encode a set of data.

One Hot Encoding One hot encoding is a method of encoding categorical information such that each value is represented by a vector of 0's and 1's that is as long as all the possible values of the set (Strand, 2016). For instance, if one wanted to encode the set of the primary colors {Blue, Green, Red}, all values would be encoded with a vector of length 3. In this example Blue would be (1,0,0) Green would be (0,1,0), and Red would be (0,0,1). The index is indicative of where in the set of data that each value resides. Another example could be that when encoding a data set with 900 unique values, each vector would be of length 900 and a 1 in the vector would be set at the corresponding location of the value in the set.

2.4 Machine Learning Technologies

Machine learning is not a single technique or technology, but rather a field of computational sciences that incorporates a wide range of technologies used to create a model and make predictions. With the rapid development of machine learning and data mining techniques, there is an increasing number of frameworks, libraries and toolkits that can be used for scientific computation and deep learning. Each machine learning tool has their own strengths and weaknesses. Therefore, choosing the right tools for different problem is crucial for achieving the goal.

2.4.1 Python

One common work-flow is to train a model using Python and a data toolkit, export the model into a standard format, and run the model in a production environment with a

language and toolkit of choice. Python only acts like a scripting language when it is being used to prototype. Additionally, the runtime bottleneck will usually be determined based on the computing power of the given hardware and the algorithms and architectures being used. With that, being able to setup and configure an AWS (Amazon Web Services) GPU (Graphics Processing Unit) computing service is an example of this workflow. Specifically, a GPU is important, because a CPU is not as effective for large amounts of data processing (Krewell, 2009).

2.4.2 Spark

Spark is an open source cluster computing framework. It is used for large scale data processing and supports high level APIs in Scala, Java, Python and R. It is powered by a stack of libraries including Spark SQL, Spark Streaming, Machine Learning, and GraphX. It supports three different deployment modes on a cluster: Standalone, Apache Mesos, and YARN. It can access the data with HDFS, Cassandra, Hive, HBase, Tachyon and any other Hadoop data source. Spark can be used to build parallel applications and it has higher performance compared to Hadoop's MapReduce in both memory and disk utility (The Apache Spark Foundation, 2016).

2.4.3 TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. All the computations are represented as directed graphs, where nodes represent the mathematical operations and the edges represent the paths where the data flows from node to node. The data communicated between nodes are called tensors, which are multidimensional data arrays of real values. The flexible architecture allows the computation to be deployed to one or more CPUs or GPUs in multiple platforms with a single API (Google, 2017).

TensorFlow was originally developed by the Google Brain team within Google's Machine Intelligence research organization for conducting machine learning and deep neural networks research, but the system is general enough to be applicable in various other domains as well, and was released under the Apache 2.0 open source license on November 9, 2015 (Metz, 2015).

2.4.4 Keras

Keras is an open source neural network library written in Python and is capable of running on top of TensorFlow and Theano. It is designed to enable easy and fast prototyping with deep neural networks through total modularity, minimalism and extensibility (Chollet, 2017c).

Chollet, the author of Keras, said that Keras was conceived to be an interface rather than an end-to-end machine-learning framework (Chollet, 2017b). It presents a high-level object oriented API which provides a more intuitive set of abstractions and makes it easy to configure the neural network regardless the backend scientific computing library (Chollet, 2017a).

2.4.5 Scikit-Learn

Scikit-learn is an open source machine-learning library for Python that is used mainly for data mining and analysis. It implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface, and is designed to work with Python numerical and plotting libraries like Numpy, SciPy and matplotlib (Scikit-learn.org, n.d.).

2.4.6 Numpy and Scipy

NumPy and SciPy are open source python libraries that are often used for scientific and technical computing. The core functionality of NumPy is its n-dimensional array data structure, along with a large library of basic mathematical functions to operate on said arrays (Numpy.org, n.d.). The SciPy package extends the functionality of NumPy with other numerical routines such as integration and optimization (Scipy.org, n.d.).

3 Methodology and Implementation

This section outlines the requirements of the project, and the strategy we used to meet them. Our strategy was heavily influenced by the availability of data, and how it was structured. This section includes both a high level description of each step, and a detailed description of how it serves the overall solution.

3.1 Requirements

We implemented an end to end pipeline that automates the process of assigning a customer support case to an engineer. We needed a work flow that promoted strong performance, (i.e high accuracy and recall), fast turnaround, user friendly interaction, and extendibility. The requirements include:

- **Performance** Our system should be producing useful results, and correctly classifying cases. This is exemplified by a system with high accuracy and recall.
- **Speed** Training and validation of our model should be fast, minimizing the time it takes to set up and start predicting with good results.
- **Usability** The work flow we create should be usable by others, and understood at a high level without a strong background in the area.
- **Extendibility** The system should be well documented and easily added to by those seeking to continue the work.

3.2 Workflow Overview

| Steps | Main Steps | Minor Steps | Technologies |
|-----------------|--------------------------------------|-----------------------------|---|
| Pre Preparation | Understanding Previous Intern's work | | Git |
| Step1 | Getting Data | Details | mongoexport, Robomongo, JavaScript |
| | | Notes | |
| Step2 | Understanding Data | Features Visualization | JavaScript |
| | | Analysis | |
| Step3 | Cleaning Data | Removing Duplicates Records | Python, JavaScript |
| | | Flattening Details | |
| | | Flattening Notes | |
| Step4 | Preprocess Data for Model | Encoding | Python |
| | | Label Selection | |
| Step5 | Create Models | JavaScript Model | JavaScript, Spark, Python, Keras, TensorFlow, AWS |
| | | Spark Model | |
| | | Keras with TensorFlow | |
| Step6 | Evaluating Models | Measurements | Bokeh, Python, Numpy |
| | | Visualization | |
| Demonstration | Webapp | Backend Server | Node.js, JavaScript |
| | | API Endpoints | |
| | | Frontend | |

Figure 8: The development pipeline

Figure 8 shows an overview of the work flow for the project. This serves as an outline for the steps taken to meet the requirements described earlier. As can be seen, there is a significant number of pieces in each step of the pipeline design. The pipeline ultimately composed of four major themes. Each one has its own sub steps to achieve its purpose. These abstractions are Data Preprocessing, Model Training, Model Evaluation, and Interactive Demonstration.

- **Data Preprocessing** Taking the raw data and manipulating it for later use.
- **Model Training** Encoding the data, and using it to train the Machine Learning model.
- **Model Evaluation** Testing the model, and measuring it's ability to predict labels.
- **Interactive Demonstration** Using the trained models to work with an actual human user and produce results that are readable by a human.

3.3 Understanding Previous Intern Work

One of our first steps was accessing work done on a project with similar goals by interns prior to us . We used this work to gain insights into how the other group had approached our problem. With some high level idea in mind of our own pipeline, this step allowed us to think critically about some of the low level details. Unfortunately, the system was designed using IPython notebooks and it proved difficult to discern details for some of the implementation and the reasoning behind it. (Jain, Gohil, Chun, & Zhou, 2017)

3.4 End-to-End Pipeline Design

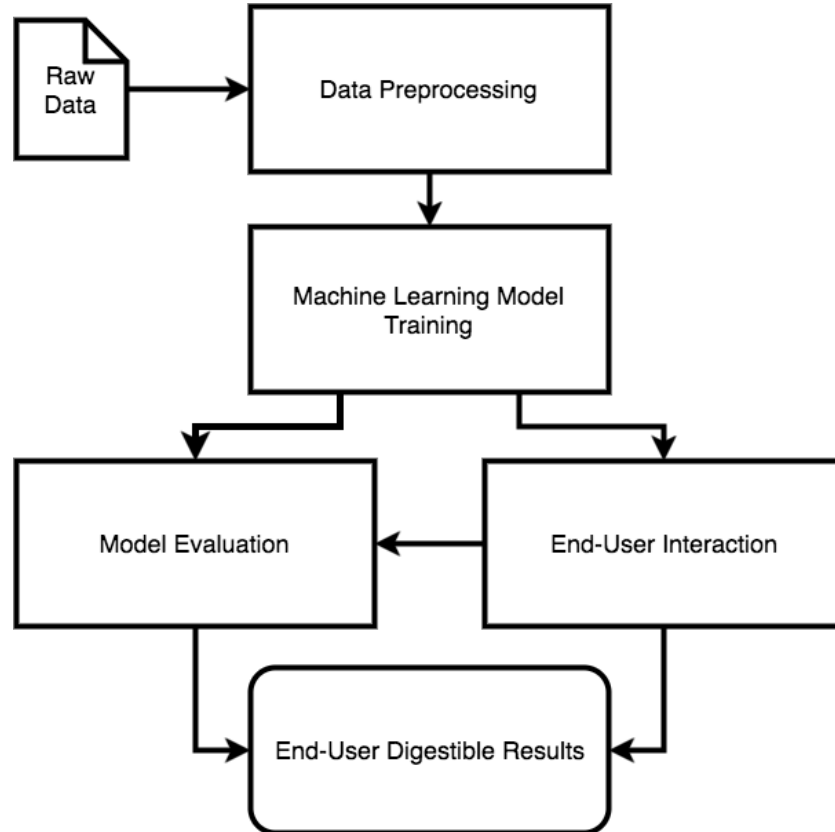


Figure 9: The end to end flow of data for an Intelligent Case Router

Figure 9 shows the overall flow of data in our proposed system. This represents an abstraction of what happens when the system is given data, and how it generates an output. There are multiple layers of abstraction here, and many minor interactions that happen within each box. Each step has multiple instances of data, and different ways of storing and manipulating it. The data preprocessing section turns raw customer support cases into data that is ready to be used in model training. The model is then trained with this data, helping it to recognize patterns. Multiple measurements are used to evaluate the model's performance, indicating success in predicting the correct route for a customer support case. Lastly, the demonstration is the link between an end user using

this system and the predictive ability of the model. This is used in a prototype that shows a user easily digestible results.

3.5 Getting Raw Data

The first step in our preprocessing is to get the data from the live system. To ensure we had a good sample set, we chose to take all cases from 2016, both "Closed" and "Resolved". This meant that the case was solved by a customer support engineer and it should have the all the relevant pieces of information for the given case. The raw data is stored in a MongoDB instance. All the data are stored in 2 MongoDB collections.

3.5.1 Robomongo

In order to get access to the data at the beginning we used some probing queries in a MongoDB client called Robomongo. Robomongo allowed us to see the structure of the data and to practice building complex queries that would serve our purposes in the pipeline. Because it is a simple tool for getting queries interactively it was not useful to use this tool in the pipeline.

3.5.2 Details

After getting access to the data, we designed a means of getting data from the mongoDB server in a usable format for later in the pipeline. We started with the "case details" records, which is the data structure holding the categorical information for any given case.

To get the cases we crafted a query that would retrieve all cases that were with in a given time, and had were considered to be closed cases that were fully resolved. We were

able to query cases based on a given time by using the case IDs, because the IDs were based off of the day that case had been created. This data was retrieved using a command line tool called mongoexport which retrieved the data and stored it in a JSON array on disk.

3.5.3 Notes

In addition to some categorical information about a case, there is a set of notes associated with each case. This is the conversation between all parties involved, including the people like customer support associates, engineers, and the customer. A "notes" record is best understood as a JSON object which is a collection of note objects. This record is the set of all notes relating to the given case. A "note" knows only the text it contains and the type of message it contains (i.e. internal note, problem description, etc).

3.6 Understanding the Data

To help optimize the data and the preprocessing, we took an auxiliary step to draw insights about the data. This was an important part of the feature selection for this system. We analyzed the data to find where the structure of a case would make work difficult. Additionally, unused fields were identified to prevent pollution in pattern recognition.

3.6.1 Categorical Features

By identifying the commonly used fields, in contrast to those that were sparsely used, we selected features that would be potentially reliable data points when training. These

fields would later make up most of the structure of a case, rather than being small parts of the raw structure from the database.

3.6.2 Visualizing the Categorical Data

To draw useful insights from the data with respect to our selected features, the whole data set was used to show the distribution of unique values for each feature. This allowed us to see which fields would be useful in creating stronger relationships than other features. To make this easily understood, many charts were generated to show this information visually.

3.7 Cleaning Raw Details

The next step in preprocessing was to clean the data set, attempting to remove vestigial information that would not help training and classification.

3.7.1 Removing Duplicate Records

Because of the way the database containing the raw records is used, some filtering for unique cases was put in place. The database stores a history of all the cases. When a case is updated, a new record is injected with the most up to date information. So all the records were sorted by the time they were sent to the database, and only the most recent version was used moving forward.

3.7.2 Flattening Details

After cleaning the records to represent only the most recent records for each case, the next important step was to flatten the data structure. Each raw record contained some

meta data from the database and the caseData object. To make the data easier to work with, and to reduce the size of all the data, a new object was created with each case containing an id, and all the selected features at the root level of the object.

3.7.3 Flattening Notes

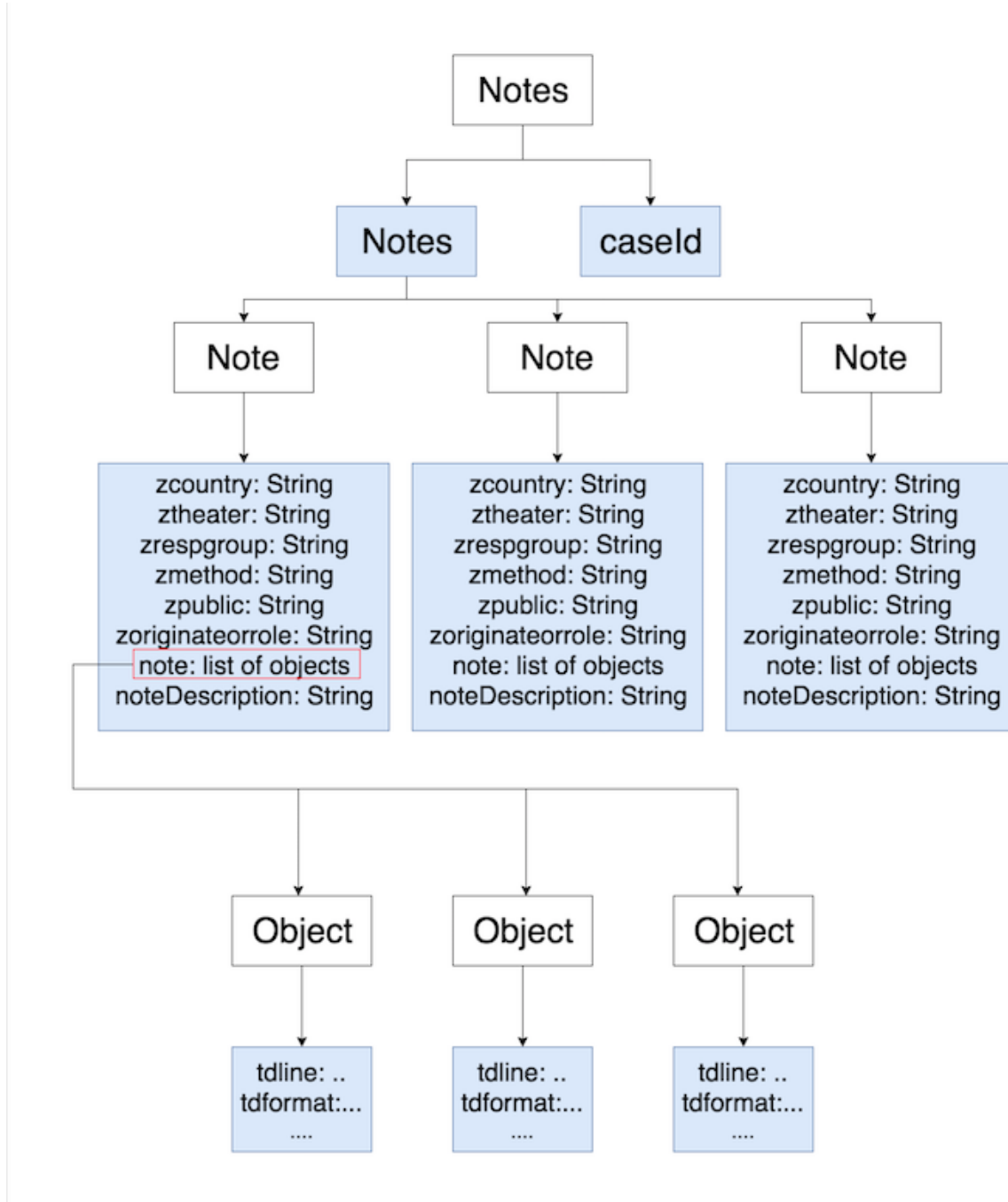


Figure 10: Preliminary tree structure of a single "notes" record

As shown in Figure 10 the notes were still stored in a complex tree structure at this point. Each "notes" object contains a list of "note" objects and a little meta data from the database. It also contains a string indicating the Case Id. In each of these "note" objects, some meta data about the conversation, but the important fields are the note description, indicating the message type, and a "note" object which is a list of all the parts making up the message. To make this structure easier to see as a human and to work with later we flattened this data in two steps. First, we created an intermediate form of the data structure where rather than each "notes.notes.note.note", the list of slices of the message, were concatenated into one string. Figure 11 demonstrates this structure.

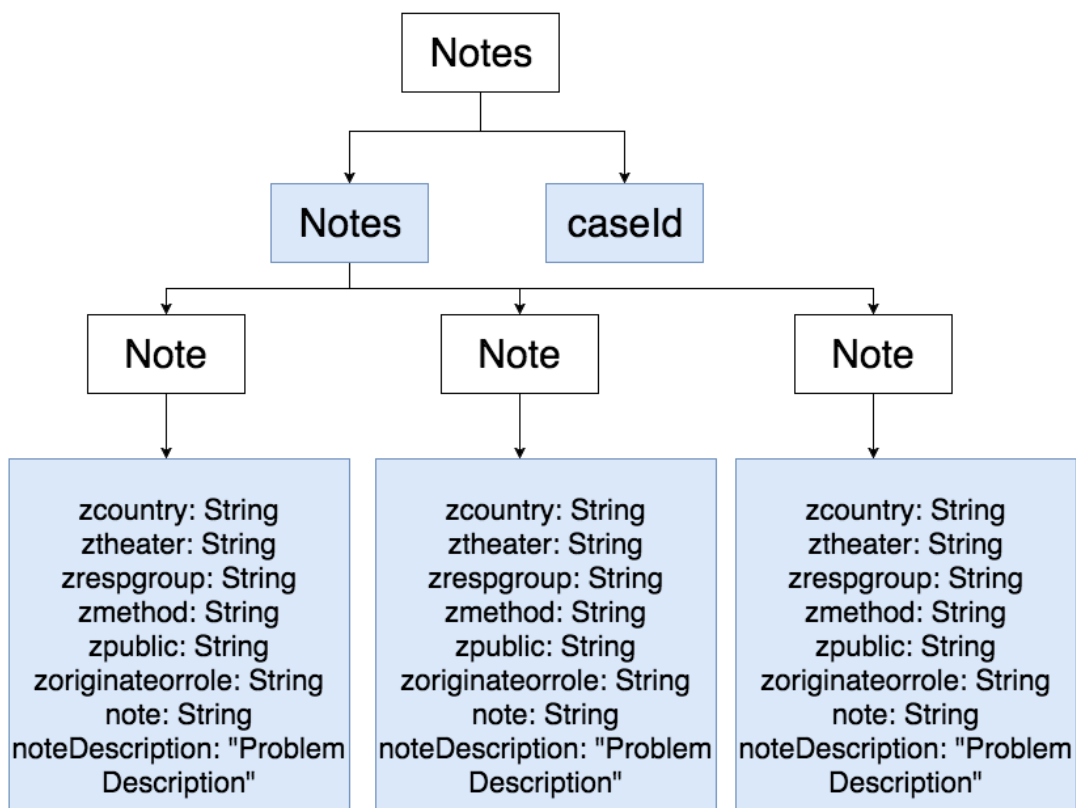


Figure 11: Intermediate tree structure of a single "notes" record

After concatenating the separate chunks of the message, we flattened down the key structure and focused on notes with with a note description of "Resolved". The cases were sorted to determine the most recent version on the database. Then a new flat "note" object was created and filled with the meta data from the "Resolved" note. Then all of the other notes were added to a list at the next level down. Figure 12 shows the structure of these flat objects, and we saved them to disk.

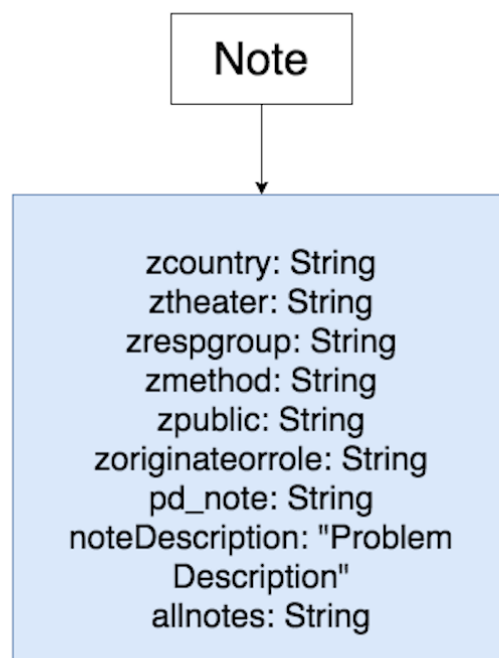


Figure 12: Final flat "note" tree structure of a single "notes" record

3.8 Converting from objects to tables

In order to work well with the libraries involved in encoding the data and training the models it was necessary to store each set of data in a csv table.

3.8.1 Details

The cleaned and formatted data still string based. So, we converted string based data to a numeric matrix for the model to consume.

3.9 A Focus on Details

We trained word2vec based on all notes we gathered. The word2vec gave us the vector value of each word in all of our text collection. We then used this set of word vectors to process the "Problem Descriptions" field in "Details". The process was simple. First we replace each word with the corresponding vector, then we added all vectors together. Then we got a vector representation of each description.

Then we made a training data with all one hot encoded categorical data and description vectors and trained a model with this data. We found that the result was not better than that produced by models which were trained on categorical data only.

At this point training a model with all notes because more notes would introduce more uncertainty into the system since many notes were just conversations people had about a given case and may not be directly related to how a case was finally resolved and closed. Using notes directly in an end to end classification system may not be an effective means of classifying. In order to use notes, another unsupervised learning method which could identify the topics and semantics of a sequences of notes should be used.

Because we had limited time, we focused on details only. First, we wanted to see if by details alone we could have reasonable results.

3.9.1 Encoding

First, we used a string indexer to convert a categorical string to an integer index. We used sci-kit learn library to do this. The output is a mapping from category string to integer. Because the indexer sorted categories alphabetically, a category with string "Axxx" had value 0, 1 or other small integers. A category with string "Zxxx" had bigger values. We named this mapping "encoding_map" and saved the result as a JSON file with the name "encoding_map.json".

Second, we used a one-hot encoder to encode string index to one-hot vector. We also used sci-kit learn to do this. The output was a Numpy matrix which could directly feed into the model. We split the data into training and testing. We then serialized both sets to disk using Scipy's "io.savemat" function. The naming convention was "xxx.mat"

3.9.2 Label Selection

We used the group names and engineer names as our labels. We built 2 models. One predicted group names and the other predicted engineer names.

3.10 Modeling

We utilized multiple frameworks and models to conduct the machine learning. Each of those tools has its own strength and weakness and targeted different problems. Those experiments help us to choose the most suitable ones for our project.

3.10.1 Javascript Analysis

To provide comparable work to the main model training flow, several smaller scale implementations were made in JavaScript to demonstrate the usability of other available libraries and techniques. These smaller scale examples were run over the entire data set, and used their own feature encoding. These examples were omitted from the final flow of data due to some restrictions in implementation and scalability.

3.10.2 Spark Machine Learning

Two machine learning pipelines were built using Spark. One is used for training the model and the other used for testing.

Training Pipeline

The training pipeline ingested the flattened csv's from step 3, where the data are well structured but not encoded. The ingested data was stored in a spark DataFrame and dropped columns "engineerId", "engineerName", "description" and "srCategory4". Then, this DataFrame passed through a pre-processing pipeline where it first encodes all string in the column of labels to columns of label indexes. The label indexes are from 0 to the number of labels, ordered by label frequencies, where the most frequent label gets index 0. After that, the pre-processing pipeline passed the DataFrame to be one-hot encoded that maps all features columns of label indexes to columns of binary vectors. The label column was not passing through onehot encoding since the column was automatically encoded as onehot vectors during the training process. Then, all features column in onehot vector forms were combined into a single vector column called "features". After transforming the DataFrame through the pre-processing pipeline, the pipeline was saved to disk.

After pre-processing the data, the DataFrame ingested by the model and the data was split that 75% of it was for training and 25% of it was for testing. To construct a neural network, the number of nodes in both input layer and output layer should be defined. Those two numbers were calculated through the DataFrame. The number of nodes in the input layer was the number of elements in the vector from the "features" column. For example, if the vector from "features" column is [1, 0, 0, 1, 0], then the number of nodes in the input layer was 5. The number of output layer nodes is the number of unique value in label column. A multilayer perceptron model was built with three layers consisting of the input layer, a hidden layer and an output layer. The number of nodes in the hidden layer was 200. There is no specific way to decide how many nodes in hidden layer, but usually a number that less than the number of nodes in previous layer and larger than the number of nodes in next layer should be chosen. In such way, the number of input nodes for each layer would be closer and closer to the the number of nodes in the output layer. Since there are only three layers and in input layer, there are around 1600 nodes and in output layer, there are around 130 nodes, 200 was chosen as the number of nodes in hidden layer. The activation function for the hidden layer was a sigmoid(logistic) function and the activation function for the output layer was a softmax function. The logistic loss function was used for optimization and L-BFGS was used as an optimization routine. After training, the model was then saved to disk. And the testing data then passed through the model and produced the accuracy for that testing dataset.

Through the training pipeline, the unique values for each category were saved to disk as a JSON file. The encoding map for the label column that maps the number back to string was also save as a JSON file.

Testing Pipeline

The flattened csv files containing the raw details data were ingested into the testing pipeline. The testing data was read into the pipeline and the JSON file that stored all unique values for each category was read into a python dictionary. For each record in the testing data, if there is new data that was unseen in the training dataset, which is determined by comparing the record with the values in the dictionary whose key is the category the record is in, the new data was replaced by None, an empty record. Then the testing data was saved as a csv file.

The pipeline next ingested this csv file to construct a Spark DataFrame. The DataFrame dropped the four columns "engineerId", "engineerName", "description" and "srCategory4". Then it loaded the saved pre-processing pipeline and the model from disk. The DataFrame firstly passed through the pre-processing pipeline and output a DataFrame that was ingested by the model. Then it passed through the multilayer perceptron model and outputted a DataFrame with a prediction for each record. By reading the JSON file containing the encoding method, it then mapped the prediction column with all numbers to a new column with human readable string label.

3.10.3 Keras with Tensorflow

We built a softmax classifier consisted of 4 layers. They were 1 output layer, 2 hidden layers and 1 output layers. The first hidden layer had 600 nodes and the second hidden layer had 200 hidden nodes. The activation function was ReLu except that the output layer used softmax activation function. The loss function is cross entropy and the optimizer is Adadelta. This was our base model for evaluations. The choice of 600 and 200 was empirical.

3.11 Evaluating the Models

Four Keras models were generated, where two were built on balanced data to predict group names and engineer names separately and two were built on unbalanced data to predict group names and engineer names separately. The balanced data has the same number of records for each group. This was avoiding the situation when some groups only have few records, the pattern was hard to find and those few records may only appear in the testing data or training data. For example, there was only one record for group A. Then this record was either included in training dataset or testing dataset. If it was included in the testing dataset, since this label never appeared before, the model could not make A prediction. If it was included in the training dataset, then because it only has one record, the model could not learn a pattern from it. The Keras models were evaluated with a different dataset and hyperparameters. The metrics used were loss, accuracy, recall and precision. Recall and precision together formed a confusion matrix. Since the Spark API does not allow it to output the accuracy and loss during the training process, one numeric number of accuracy was used for evaluation.

3.11.1 Data Split

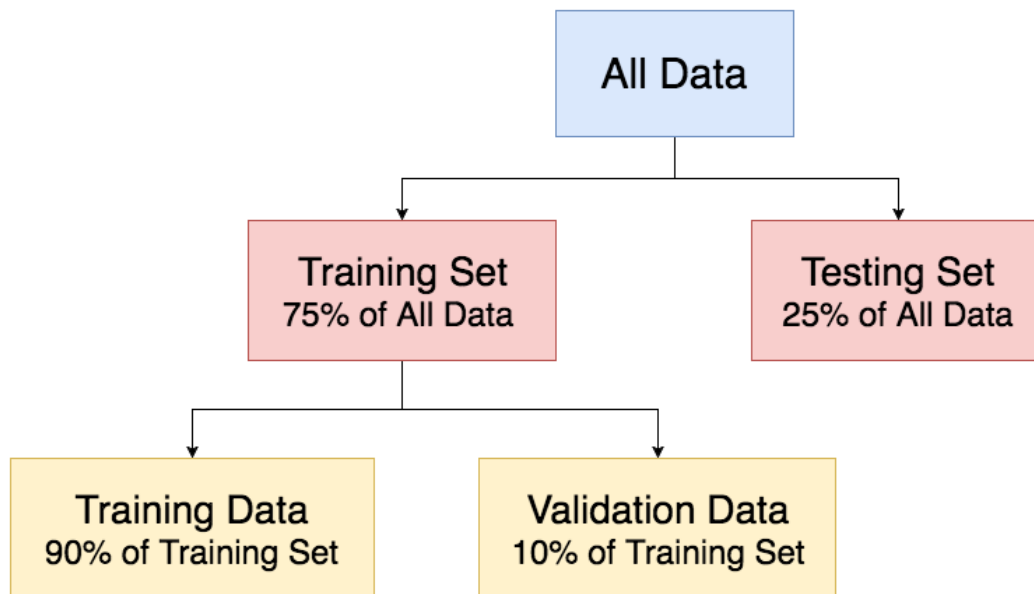


Figure 13: Data split for training and testing to build Keras models

The choice of 25% was a common practice. The idea here was testing data should be large enough to represent the distribution of the population.

There are two processes to generate a reasonable model. One is the process to create the model and the other is testing the static model. 75% of all data was used for the first process while 25% of all data was used for the second process. In the first process, the model was trained for multiple iterations. Therefore, after each iteration, among 75% of all data, 10% of it was used for validation so that we could know the model performance during the training process. This information was used to determine if an early termination or more iterations were needed. If the accuracy of training data increase while the accuracy of validation data remains same or decreases, it means the model is

likely to be overfitted, so a smaller number of iterations should be chosen. If the accuracies for both testing and validation are kept increasing, it means more iterations were needed. In a word, this information was used to decide the number of iterations we should choose in the model creating process.

After the first process was done, a static model was saved into disk. The testing data, which was 25% of the all data, was used in the second process. A static model was loaded and the testing data was passed through the model and the model would make predictions. This process was to test the performance of a static model, but doesn't involve building the model. Therefore, whatever the number is, it only shows the performance of this static model, but has no influence on how the model looks like.

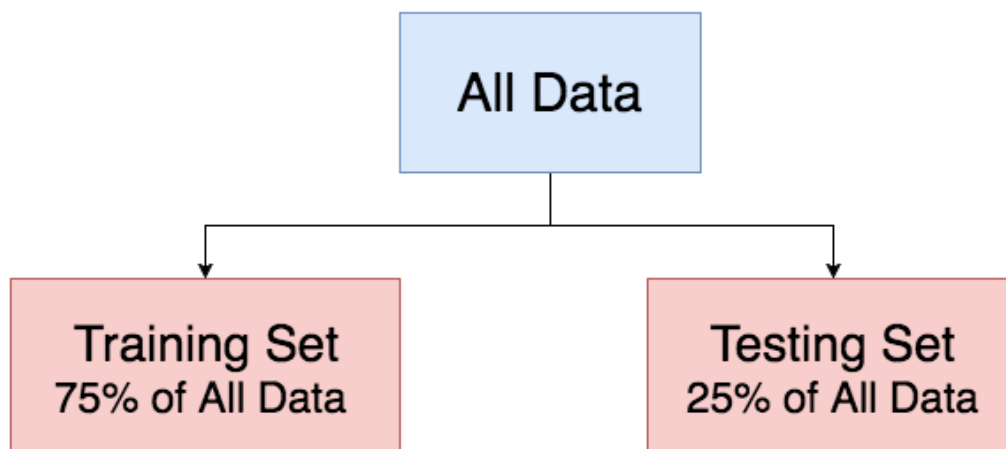


Figure 14: Data split for training and testing to build spark models

In Spark, there are only two parts of the data, where 25% of all data was used for testing and 75% of all data was used for training. There is no validation data involved in the training process because Spark 2.0.1 API doesn't support that functionality to set the validation data set. Moreover, in Spark, there is a parameter called "tol" which is the convergence

tolerance and a parameter used to set the maximum number of iterations. Therefore, when the tolerance is below a certain level, which was set by developers, the model was return without further training. In such case, developer don't need to manually set the number of iterations.

3.11.2 Measurements for Keras Model

To evaluate Keras models, four plots were drawn, which were named as accuracy_plot, loss_plot, confusion_matrix and distribution_plot for easier reference. Plots were drawn using the Python library Bokeh that can generate interactive html plots. Figure 15 to 18 are examples of how those four plots look like and more in depth explanation and analysis are discussed in result session.

During the training process, the data was split such 10% was used as for validation and 90% was used as for training.

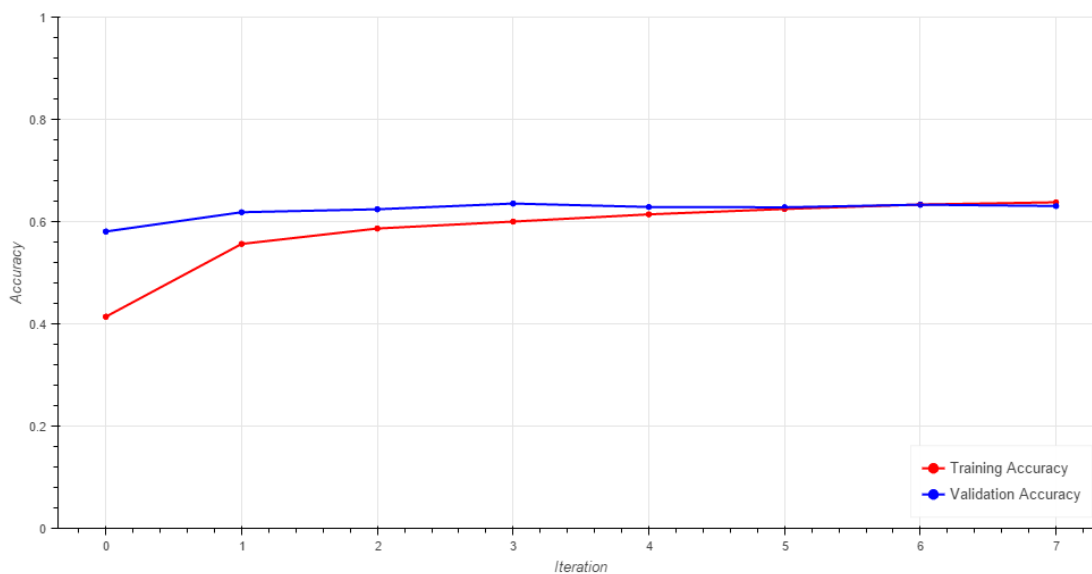


Figure 15: accuracy__plot example

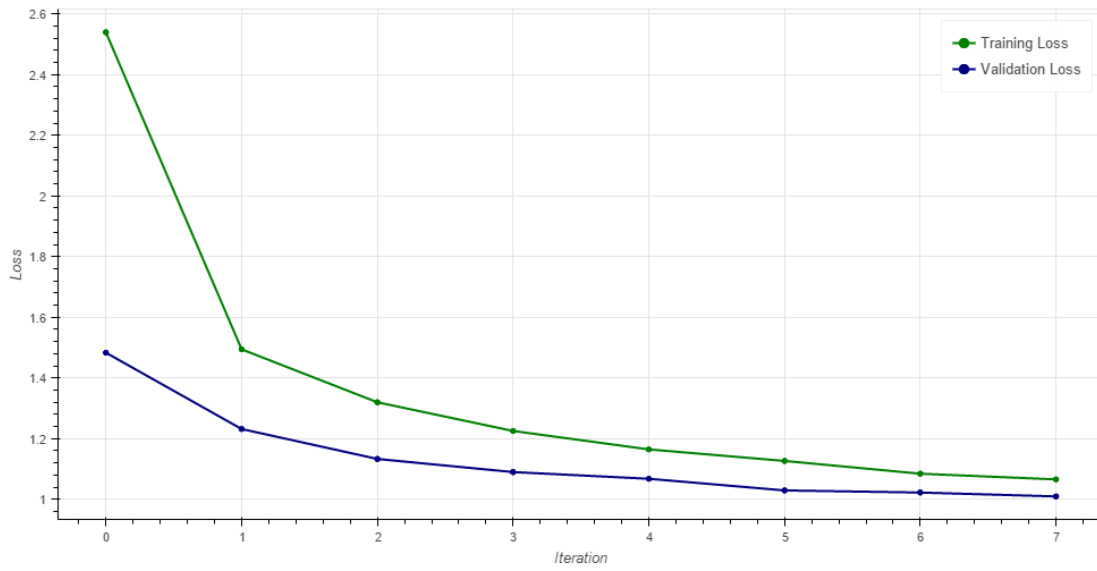


Figure 16: loss_plot example

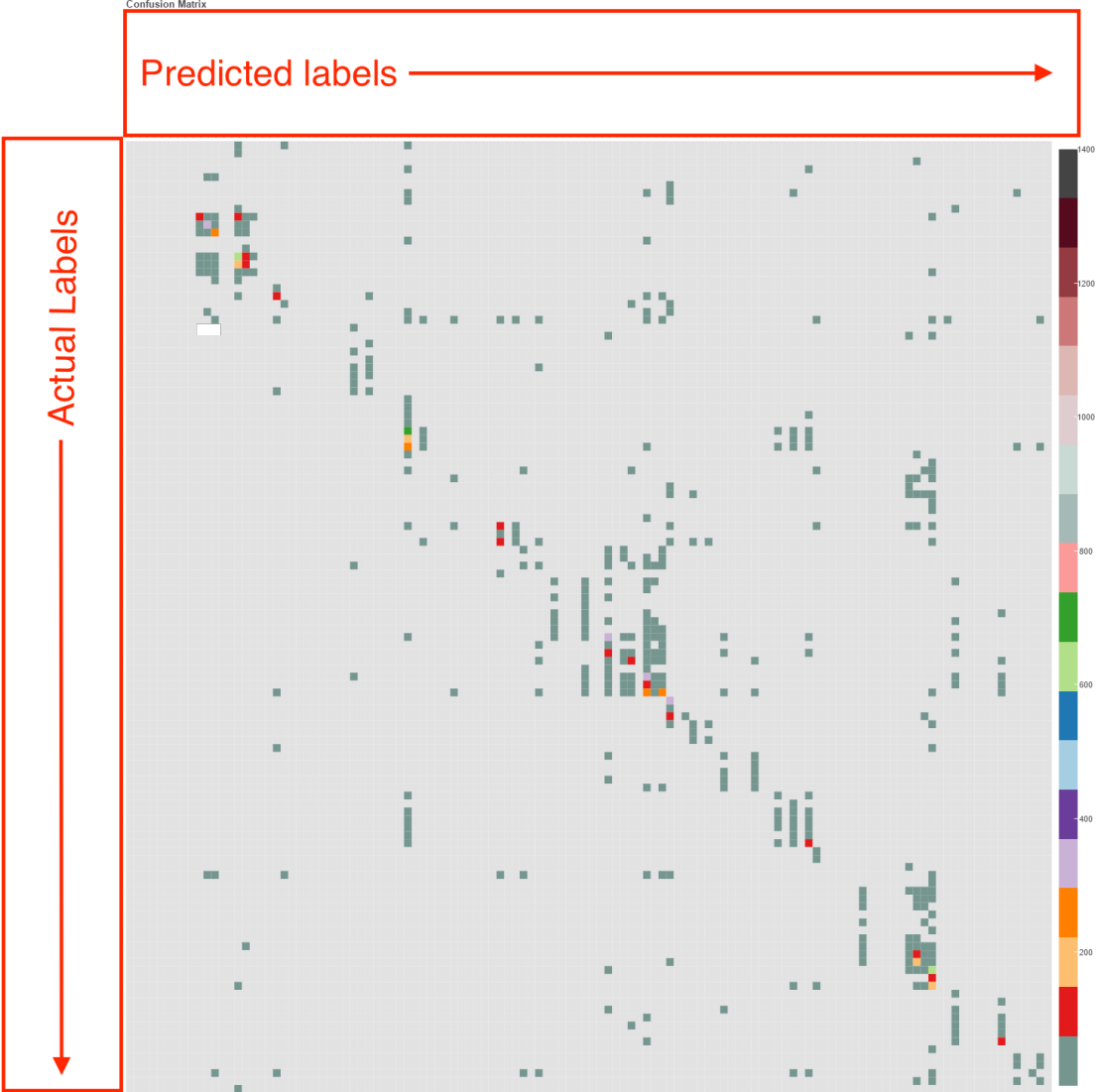


Figure 17: confusion_matrix example (labels hidden for privacy)

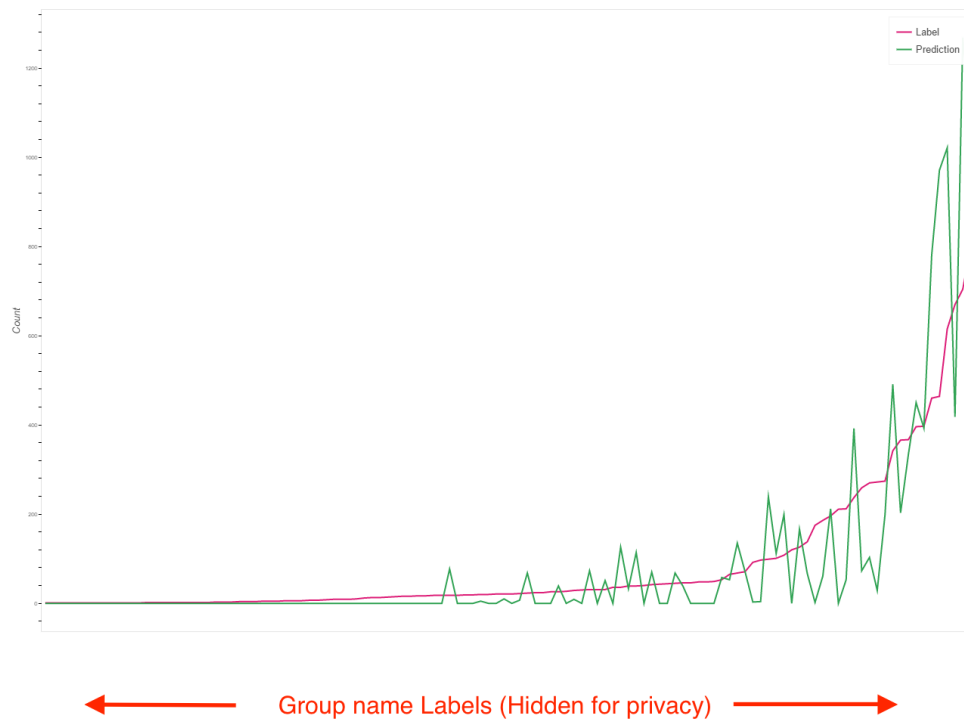


Figure 18: distribution of predictions and labels example

The accuracy_plot measures the accuracies after each iteration of training for both validation and training data. Two lines, one for validation data and the other for training data, were drawn in the plot where the xaxis were iterations and yaxis were the accuracy. Hovering over the points showed the exact number for accuracy and iteration. This plot was used to determine if the model is overfitting. If the accuracy of training data increases while the accuracy of testing data doesn't increase or not increase as much as the training data, that is usually a sign for overfitting. To generate models without much overfitting, we chose a relatively large number of iterations first. For example, we made the model be trained for 30 times. From the observation, we estimated that after 8 or 9 iterations, the model seems to be overfitted by observing the two lines' trending. Then we built a new model and only trained it 8 or 9 times to avoid the overfitting.

Machine learning algorithms seek to minimize the loss. During the training process, the loss values were what the model used for optimization, while the accuracy does not exist in the training process but calculated after the model was built. Therefore, loss functions need to be chosen in order to generate a reasonable model. If the model has low loss but also low accuracy, it means the loss function was not suitable and the model cannot be properly optimized. Two lines, one for validation and one for training data, were drawn in `loss_plot` to describe the loss after each iteration and compare this plot with the `accuracy_plot`, it indicated the performance of the loss function. Hovering over the points showed the exact number for accuracy and iteration.

The `confusion_matrix` was drawn with testing data, which was 25% of the total data. It reflects the relationship between predictions and labels. A color bar was drawn to indicate the color scale in the confusion matrix. When measuring the model to predict group names, the tick labels were drawn but when measuring the model to predict engineer names, the tick labels were not displayed since there were more than 900 of them. However, hovering over each triangular showed the category of the label, the category of the prediction and the number in such cases.

The `distribution_plot` was used to describe the distribution for predictions and labels. It is used to capture the model's performance which was not reflected by the accuracy. For example, if 70% of data was labeled as A, and the remainder was labeled as B. If all data was predicted as A, the accuracy was 70% but the model is quite biased. Therefore, two lines where one showed predictions and one showed labels were drawn. In order to have a better understanding, we sorted the list based on the number of labels. Therefore, the line indicating labels went smoothly upward where the line indicating predictions oscillates around it.

3.12 Webapp Demonstration

To demonstrate the flow of data in a way that a human can understand, we designed a webapp that would use our trained model to make predictions. This app was a prototype of how a full scale system might look. The app is accessed by the user in the browser and allows the user to submit a new case using the values from the original data set. The system makes predictions based off of this information and asks the user to provide feedback regarding the usefulness and correctness of the predictions.

3.12.1 Backend Server

Backend server was a Flask server. Flask is a Python web framework. The server was a HTTP server. Before opening the HTTP server, the server application loaded four .h5 models into memory. These four models are the following:

- Model 1. A model which was trained with balanced data and predicted groupName
- Model 2. A model which was trained with balanced data and predicted engineerName
- Model 3. A model which was trained with unbalanced data and predicted groupName
- Model 4. A model which was trained with unbalanced data and predicted engineerName

After the models were loaded, the server started to listen to HTTP connections and would respond accordingly.

3.12.2 API Endpoints

/Run This was a HTTP POST connection. The post data was the data of a case. The server would run all 4 models and send results back to the front end.

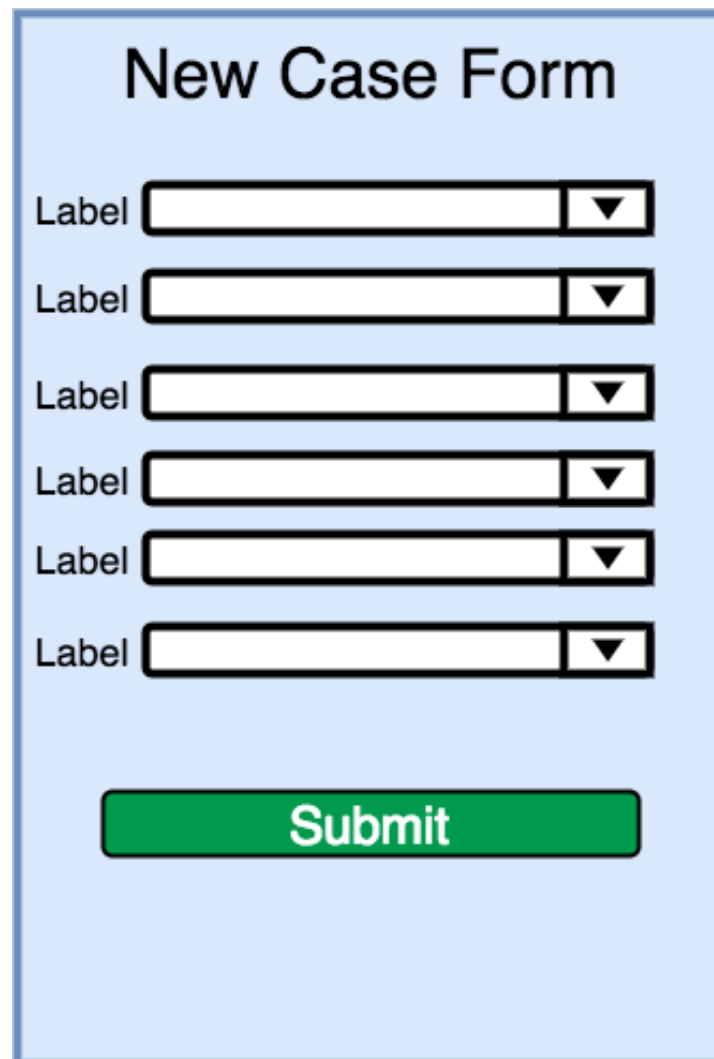
The server first ran 2 models which predicted groupName. We got 2 outputs, one from model 1 and another from model 3. Then the server would add these 2 results back to the input data and feed them to model 2 and model 4 to get engineerName predictions.

Now we had 4 results. The server sent 4 results back to front end as a JSON string. This JSON object describes which result was predicted by which model so that we could do a comparison. The JSON also contained the top 3 confident results of each model instead of just the top 1 result.

/Feedback This endpoint is used in the front end to send information to the backend via an HTTP POST request. The front sends an object representing the state of the user's feedback, which is included for each model the webapp has included.

3.12.3 Frontend

The user interacts directly with a web page with two states, the "New Case Form" and the "Prediction Results".



The image shows a mockup of a 'New Case Form' with a light blue background. At the top, the title 'New Case Form' is displayed in a large, bold, black font. Below the title, there are six identical input fields, each consisting of a white rectangular box with a black border and a small black downward-pointing triangle on the right side. Each input field is preceded by the word 'Label' in a black font. At the bottom of the form, there is a prominent green button with rounded corners and the word 'Submit' written in white, bold, sans-serif font.

Figure 19: a mockup of the New Case form

New Case Form Figure 19 shows the "new case" form which is a set of comboboxes, which allow the user to either select a predefined value or type in the box, to find a predefined value in the set of valid options. This form a new case for the model the makes predictions on.

Prediction Results The backend sends back prediction results after a case is submitted, and the front end displays each set of results with corresponding feedback questions.

| Predicted Labels | | |
|------------------|--------------|------------|
| # | Person | Confidence |
| 1 | Joey Perez | 39% |
| 2 | Ziyan Ding | 32% |
| 3 | Xuanzhe Wang | 11% |
| | | |

The Model has predicted that this case could be solved by one of these 3 people. Would assign it to any of them?

Joey Perez

Ziyan Ding

Xuanzhe Wang

None

Figure 20: A mockup of an example results and corresponding feedback

Results include the top 3 groups suggested for the case, and the top 3 employees suggested for the case, assuming the number 1 group is assigned. A case summary is displayed, which contains the information in the submitted case that the system is predicting groups and employees for. Figure 20 shows what the front end would look like if the model had suggested that our team were the 3 most likely engineers to solve the problem. The example shows that the model suggested the 3 of us, and the user selected Ziyan and Xuanzhe as the engineers that they would actually route the case to.

User Feedback for Results In correspondence with the results, feedback input is provided, so that the user can indicate whether or not he or she agrees with each model's predicted groups and employees. At the bottom of the application, there is a summary of the options the user chose, and what the model suggested, so that the user can see what is being sent to the backend server before hitting "submit".

4 Results

This section presents our findings and what they mean to this project. The graphs and tables are important to understanding how our system works, and to what level we succeeded in meeting our requirements.

4.1 Case Data Analysis

To understand the data we were working with, we ran several forms of analysis over the data. Looking at the distribution of some of the data based on all the various features and labels, we can discern some important points about the data set. Below is an overview of the number of cases in our data set by month. There are about 42,000 "Closed" and "Resolved" cases total, and they are pretty dense in the summer months. The x-axis is our month name, and the y-axis is the number of cases resolved in that. In 2016, we can see that many of the cases were solved in June and July. We had explored the number of "Closed" and "Resolved" cases from 2011 to 2016, and only found an increase of about 1,000 cases. So, we decided not to include anything before 2016.

4.1.1 Feature Distribution

We analyzed each feature to show how it fit into the data set. Any new customer support case should exist as a permutation of data in this domain. The actual details from this analysis is omitted for privacy reasons. This analysis is also important because it shows us how long a single piece of encoded data will be. Each feature is encoded based off of the number of unique possible values, so if you were to add up the number of possible values in this set, then a single piece of encoded data, would be of that length.

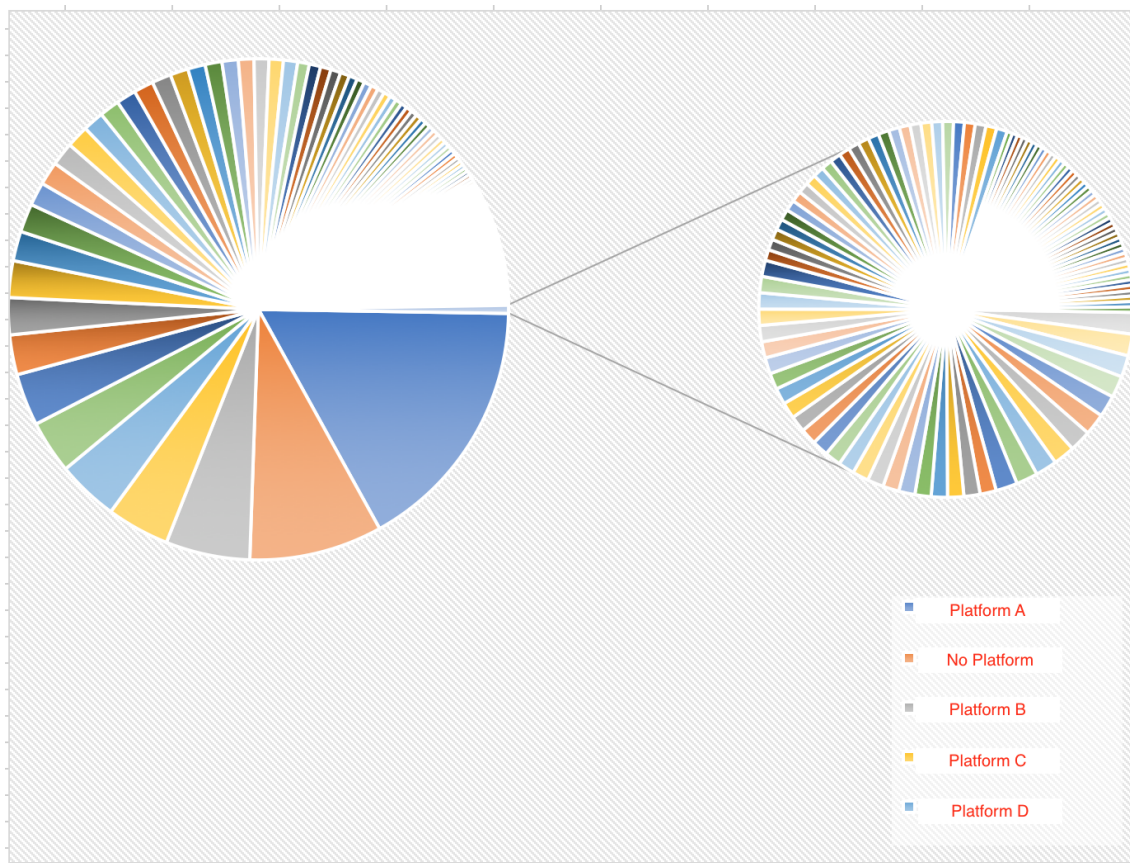


Figure 21: Distribution of specified platforms

In Figure 21 we can see that the distribution of platforms has a pretty large number of possible values. Importantly, in Figure 21 we see that the second most prevalent value, light orange in the first quadrant of the graph, is actually the number of cases with no platform specified. Semantically, this value could have several meanings. For instance, it could mean that the platform wasn't relevant, but it could also mean that the platform involved was not known when solving the problem. These two uses can have different effects on the pattern, so this value could have some negative effect on overall performance when training and when classifying.

4.1.2 Label Distribution

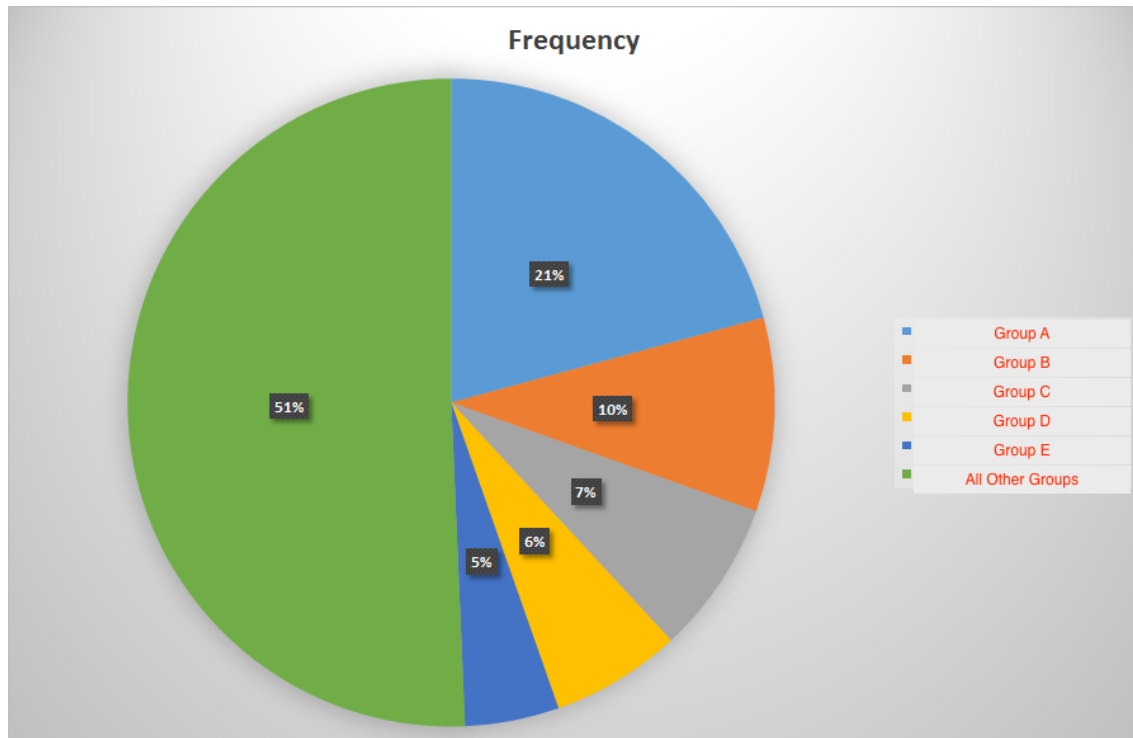


Figure 22: Distribution of "groupName" values

As Figure 22 demonstrates, 5 groups make up 49% of the sample. The other groups make up the rest. Given this label distribution, it is likely there is a strong bias towards these 5 majority groups. We expected that a classification model would be highly biased towards majority groups, and this is evidence that these majority groups exist.

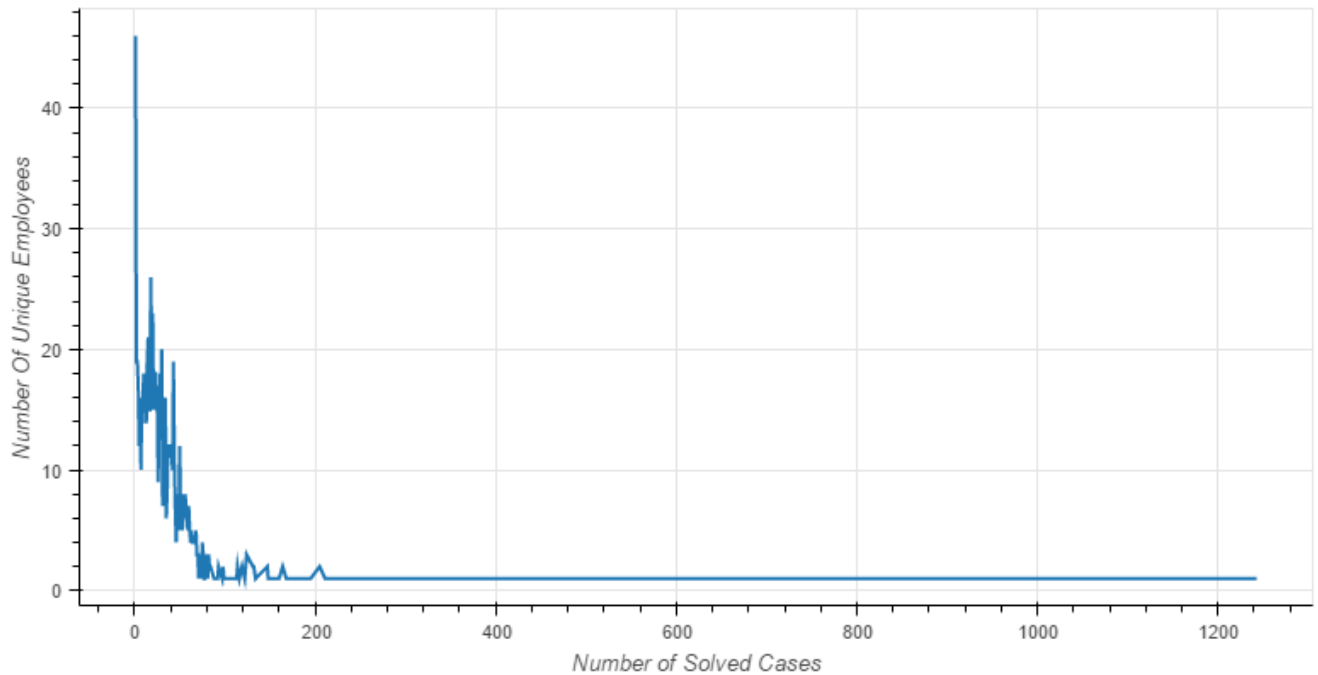


Figure 23: Distribution of Engineers assigned to cases

Figure 23 shows the distribution of engineers who solved our "Closed" and "Resolved" cases. The horizontal axis shows the number of cases solved. The vertical axis is the number of employees who solved that many cases. For example, at the left most data point, those employees only solved 1 case. At the right most data point, there is a single employee who solved 1243 cases. Most people solved between 1 to 50 cases, but most of the total cases were solved by less than 20 people. Also, there was a significant number of cases that did not have an engineer listed. This distribution indicates that the model will potentially have a huge bias towards the few people who solved most of the cases, which probably lead to a high accuracy but low recall model.

4.1.3 Case Note Insights

We trained a word2vec model with all the case notes and investigated the relationship between some words. We had used an analytics library called GINsim to find the difference between word pairs. The difference is determined by calculating the cosine value of the angle between the two vectors. For simplicity, here we show the similarities of words pairs:

The difference between 'network' and 'hardware' was 0.210. The difference between 'network' and 'software' 0.177. Semantically 'network' is more of a hardware topic. The difference here is small, so 'network' and 'hardware' are shown to be related.

The difference between 'router' and 'hardware' was 0.216, and the difference between 'router' and 'software' was 0.300. Semantically, 'router' should be more of a software topic, but this is not intuitive, as seen by a high difference between 'router' and 'hardware'.

The difference between 'security' and 'hardware' is 0.155, and the difference between 'security' and 'software' is 0.089. 'Security', outside of a networking context is often more a software topic, but this is likely not intuitive.

The difference between 'linux' and 'hardware' was 0.250, and 0.365 between 'linux' and 'software'. Realistically 'linux' is more of a software topic, and this is not intuitive.

The difference between 'switch' and 'hardware' is 0.359 and it is 0.294 between 'switch' and 'software'. 'switch' is more of a hardware topic, and it should be intuitive.

Note that similarity does not mean the similarity between definitions of the words. It is best understood as the probability of coexistence of 2 words in the same topic, or the likelihood of 2 words show up in the same context. The bigger the number is, the more likely that 2 words occur in the same context. The likelihood is not the "likelihood" in statistics. However, there are 2 ways to interpret this similarity. The first is that high similarity indicates similarity in the real world. That is, 'linux' is software in real life and a switch is indeed a piece of hardware. The second interpretation is that similarity indicates where the problem is. Specifically, since customers knows a router is a piece of hardware, the hardware part of a router is probably well designed and not likely to fail. So, when a case comes in, people discuss the software related information about a router more since the software part may have more problems. That's why 'software' and 'router' coexist more. Lastly, it is important to note that all these outcomes are only interpretations of the notes and not real world truths.

4.2 Model Performance

To understand how our models performed we used several metrics to evaluate performance including accuracy, loss, and recall. We mainly focused on accuracy as it is easy to understand and is generally indicative of success.

4.2.1 Keras Models

There are in total four Keras models and they are named as following for easily reference.

- **unbalanced_groupName**: A model trained with unbalanced data that predicted group name
- **balanced_groupName**: A model trained with balanced data that predicted group name

- **unbalanced_engineerName**: A model trained with unbalanced data that predicted engineer name
- **balanced_engineerName**: A model trained with balanced data that predicted engineer name

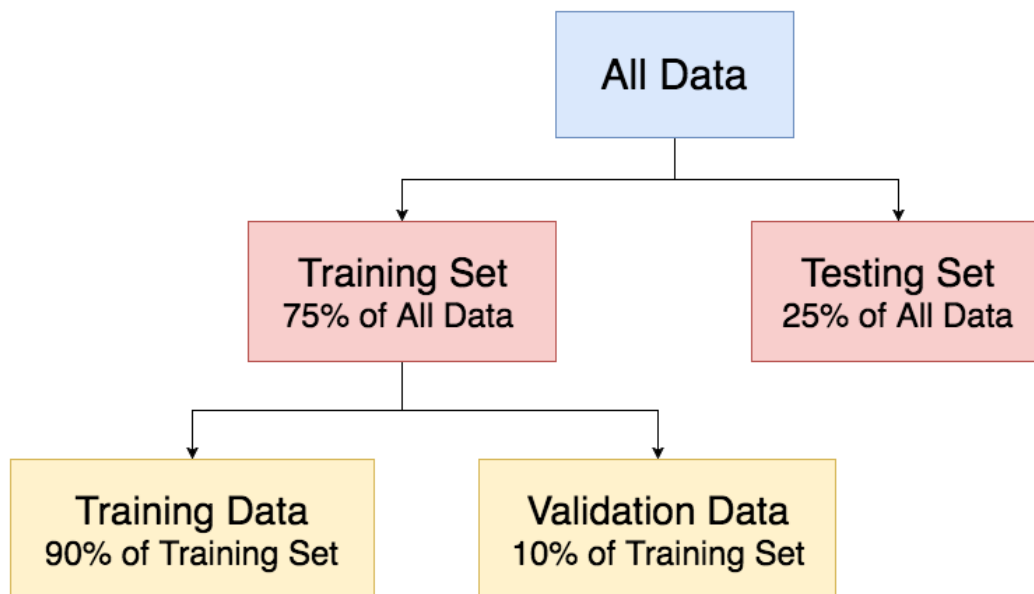


Figure 24: Data split for training and testing

As explained in section 3, Figure 24 shows that 75% of all data was used during the training process. After the model was generated and stored into disk, the model was then loaded and the testing data, which was 25% of all data, would go through the model and get predictions. Those 25% of data would not have any influence on the model, they just passed through the static model and get predictions. Among those 75% of all data, for each iteration, 90% of it was used to train and 10% of it was used to validate. Those validation data would help determine if the model was overfitting or not and unlike the testing set, validation data was involved in building the model and used

in each training iteration.

There are four different types of graphs. The plots of accuracy and loss captured the model performance during the training process. Therefore, the data involved were training data and validation data. The plots of distribution and confusion matrix captured the model performance on testing data, which are not involved during the training process.

4.2.2 Keras Model unbalanced_groupName

This Keras model was built on unbalanced data to predict group name.

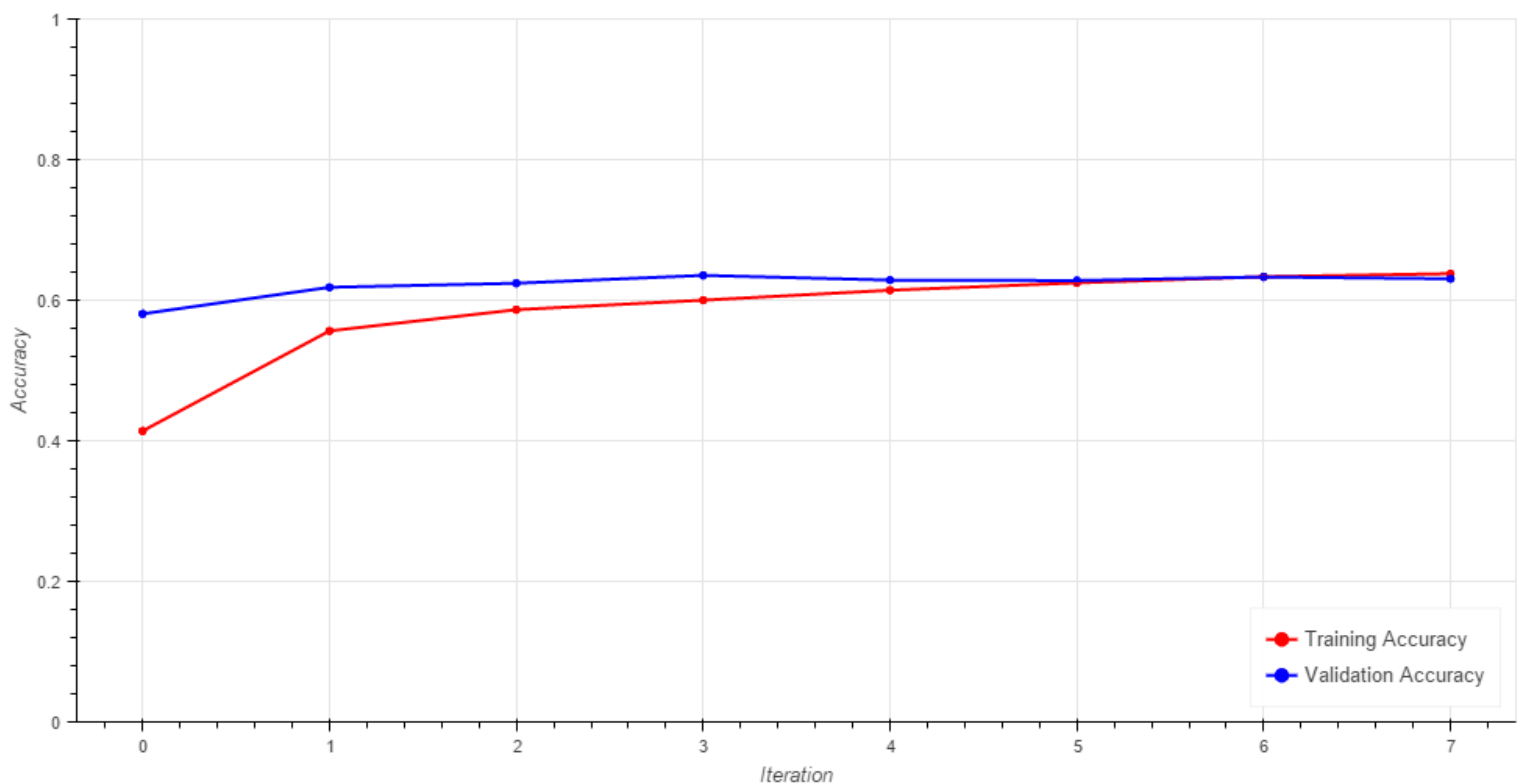


Figure 25: Accuracy during training where the model is built on unbalanced data and predicts the group name

Figure 25 describes the model's accuracies to predict group name on both unbalanced validation and training data during 8 training times. The x axis is the number of iterations, which is the number of times the model was trained. The y axis is the accuracy in decimal form. The red line demonstrates the trends for training data while the blue line demonstrates the trends for validation data. If the red line dramatically increases while the blue line decreases much, the model was overfitted. The model was firstly trained for 30 times and we noticed after 8 and 9 times, the red line keeps going up but the blue line remains the same and even going downward. Therefore, 8 was chosen as the number of iterations to train the final model. During these 8 times of training, both lines go smoothly upward, which means the model was not overfitted much. After 8 iterations, the accuracies for both validation and training data are around 63%.

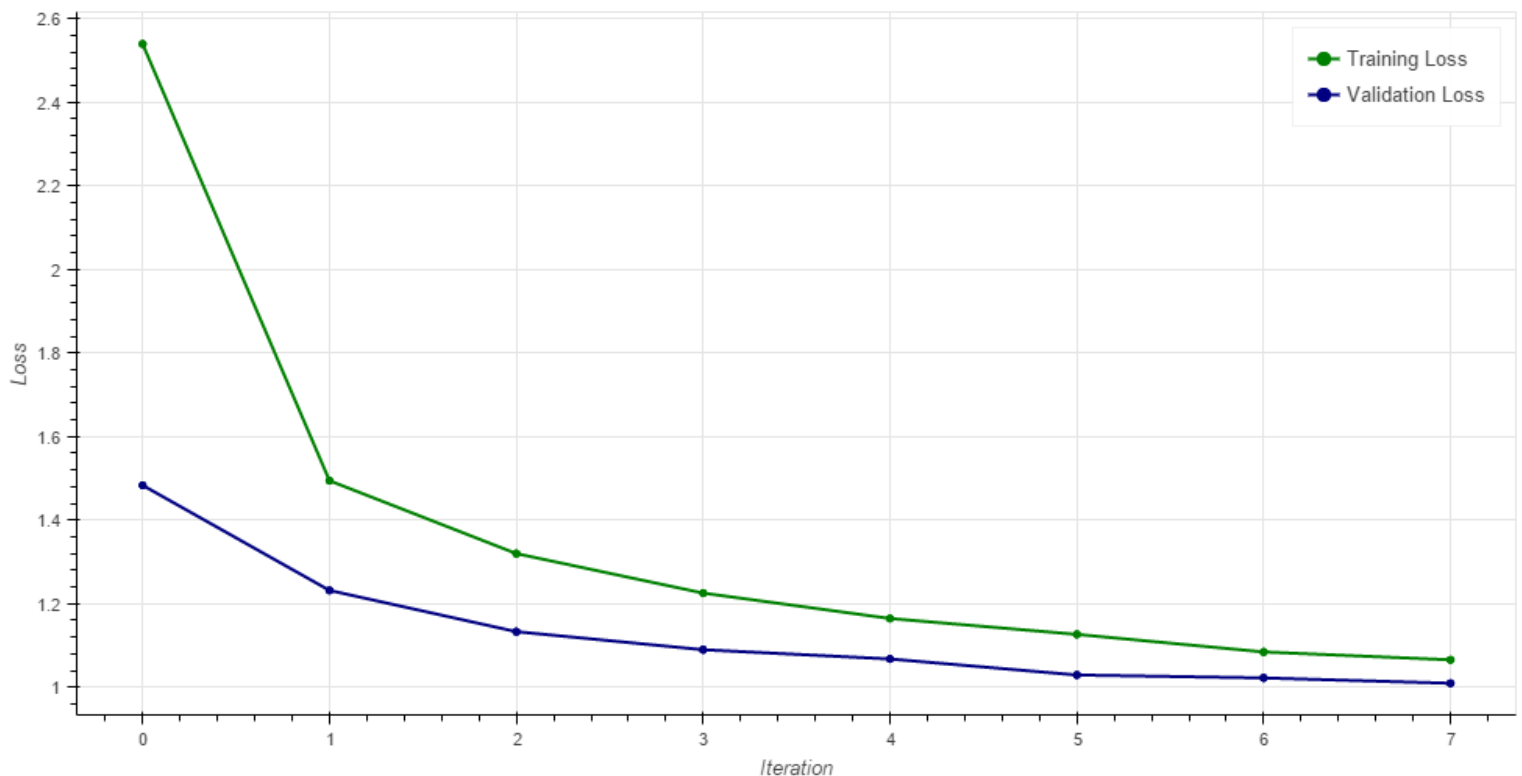


Figure 26: Loss during training where the model is built on unbalanced data and predicts the group name

Figure 26 describes the model's loss value to predict group name on both unbalanced validation and training data during 8 training times. The x axis is the number of iterations, which is the number of times the model was trained. The y axis is the loss in decimal form. The red line demonstrates the trends for training data while the blue line demonstrates the trends for validation data. Both lines go downward, which means the loss values for both validation and testing data decrease. As explained in section3, because accuracies increase as indicated in Figure 25 while losses decrease as indicated in Figure 26, it means the right loss function was chosen. After 8 iterations, the loss value for validation data becomes 1.009 and the loss value for training data is 1.065.

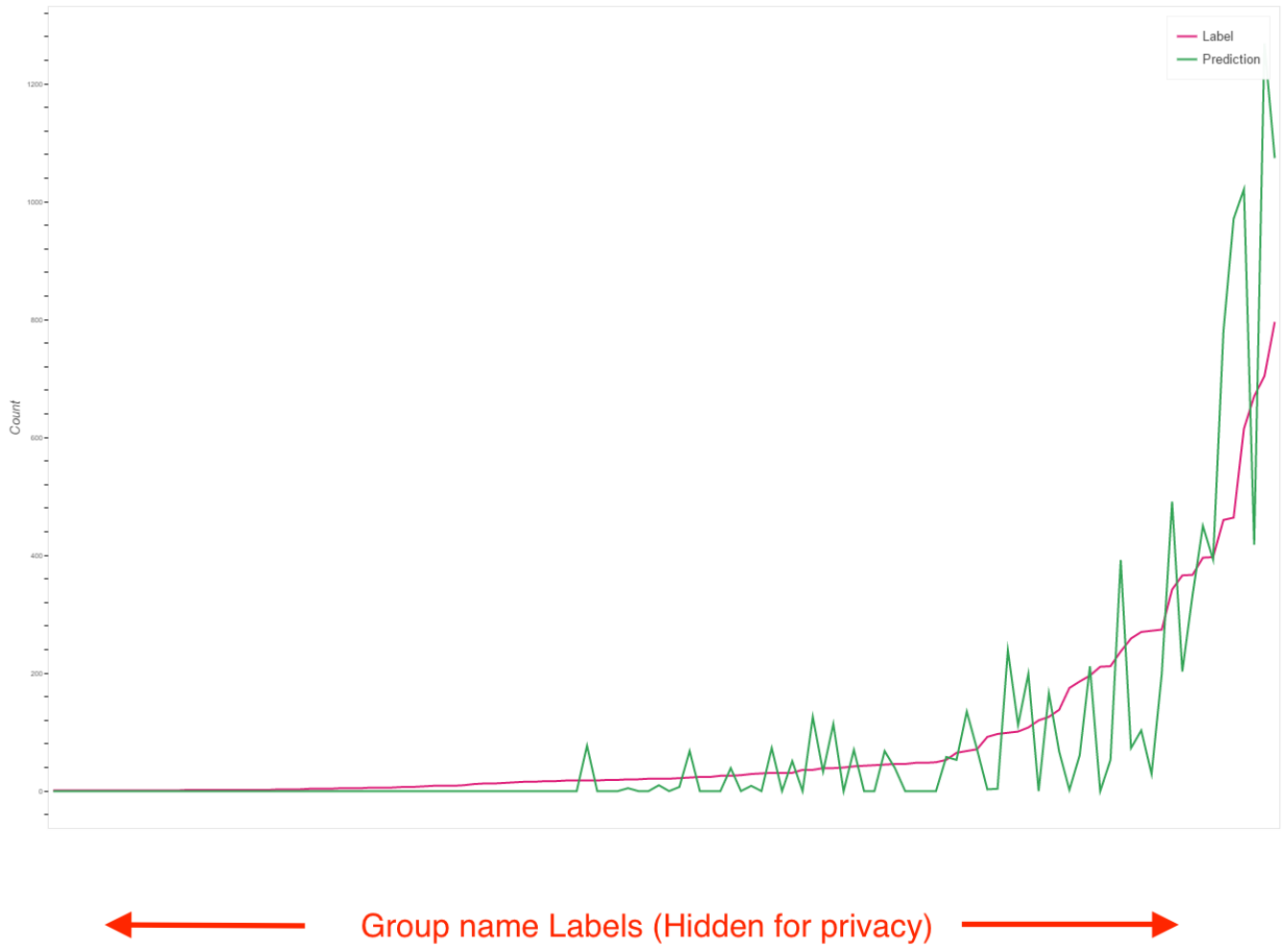


Figure 27: Distribution of group name predictions and labels for unbalanced testing data

25% of all unbalanced data was used for testing. The static model was loaded and testing data was passed through the model. Figure 27 shows the distribution of the predictions and labels. The x-axis is the group name and y-axis is the number of cases that solved by that group. For easier analyzing, it was sorted by the number of cases in labels. For example, for the group that solve most cases, its name was the right most value on x-axis. The y-axis value is in range from 0 to about 1200 and the maximum difference between red line and green line is about 400. Since the red line describes the

trending for labels and green line shows the trending for predictions, the red line smoothly increases while the green line vibrates around the red line.

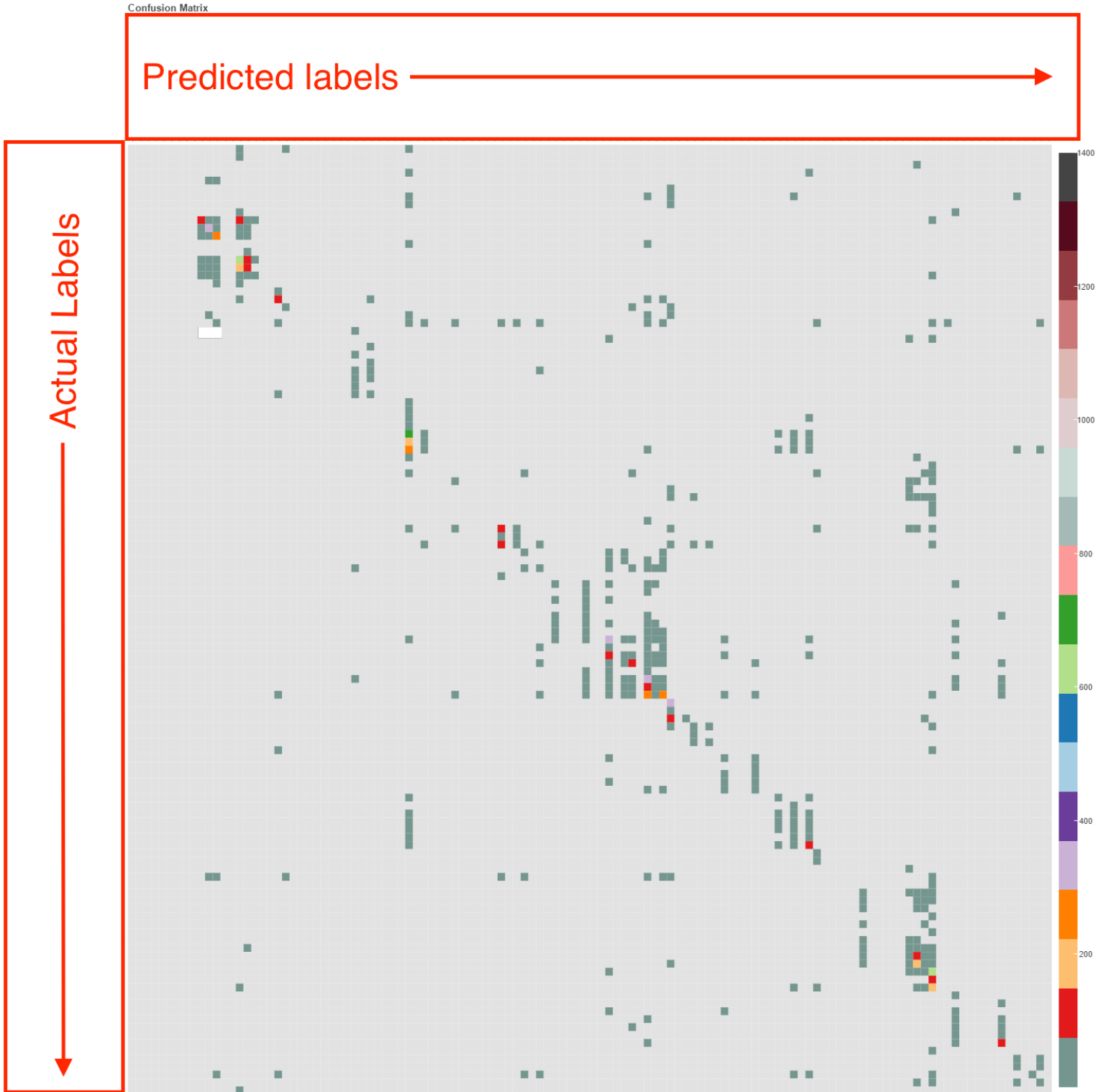


Figure 28: Confusion matrix of group name for unbalanced testing data

Figure 28 is the confusion matrix, drawn on unbalanced testing data. The x-axis is the group name in prediction while the y-axis is the group name in labels. Although there are outliers, there is an obvious diagonal line, which indicates the model finds the underlying relationship between the features and group name and can make an overall correct predictions.

4.2.3 Keras Model `balanced_groupName`

This Keras model was built on balanced data to predict group name.

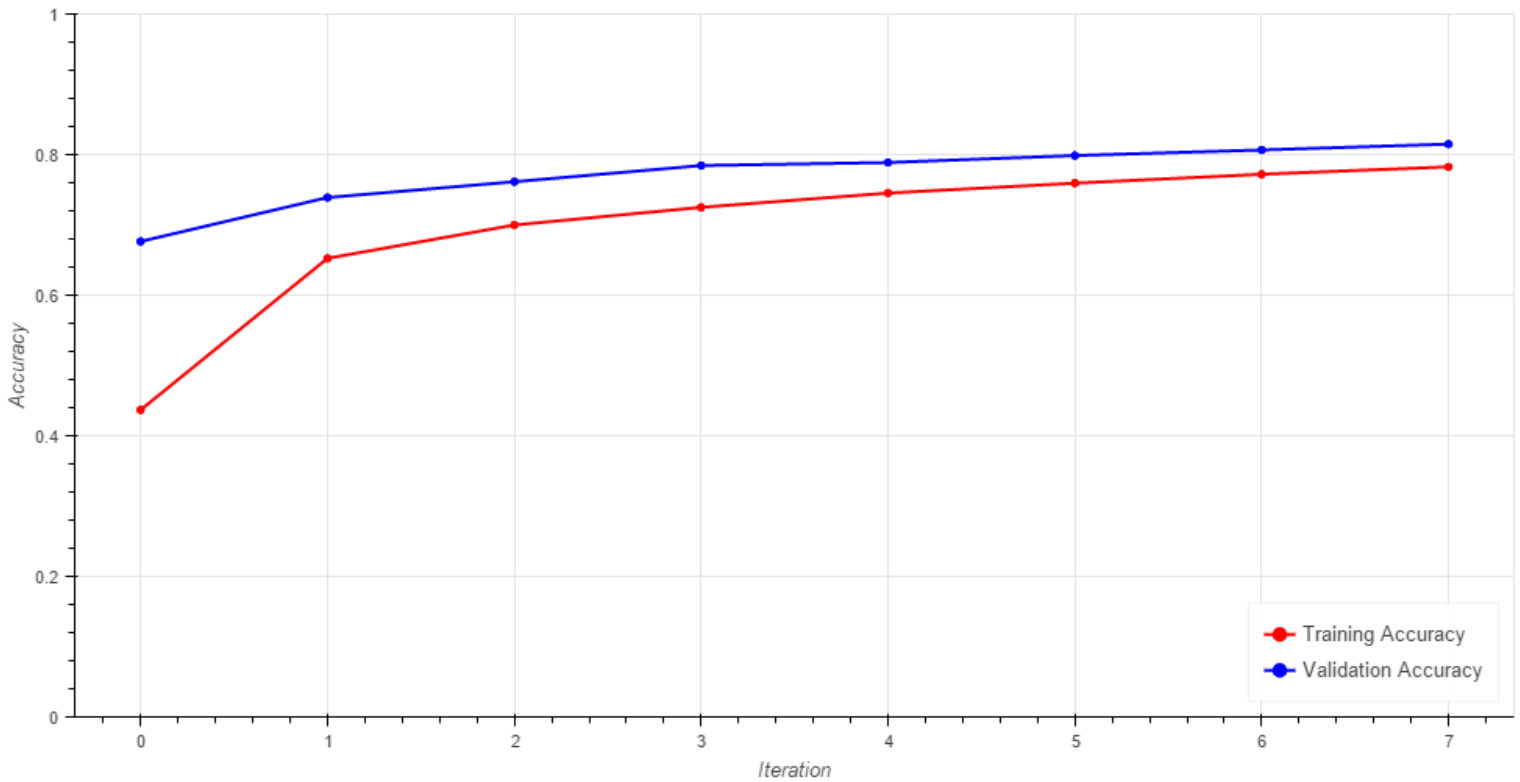


Figure 29: Accuracy during training where the model is built on balanced data and predicts the group name

Figure 25 describes the model's accuracies to predict group name on both balanced validation and training data during 8 training times. The x axis is the number of

iterations, which is the number of times the model was trained. The y axis is the accuracy in decimal form. The red line demonstrates the trends for training data while the blue line demonstrates the trends for validation data. Similar as the explanation for Figure 25, both lines go upward without crossing, so the model keeps improving during those 8 iterations. After 8 iterations, the accuracy for validation is 81.5% and the accuracy for training is 78.3%. Comparing to Figure 25, the model trained on balanced data has a better performance on the balanced testing data than the performance the model trained on unbalanced data has on unbalanced testing data.

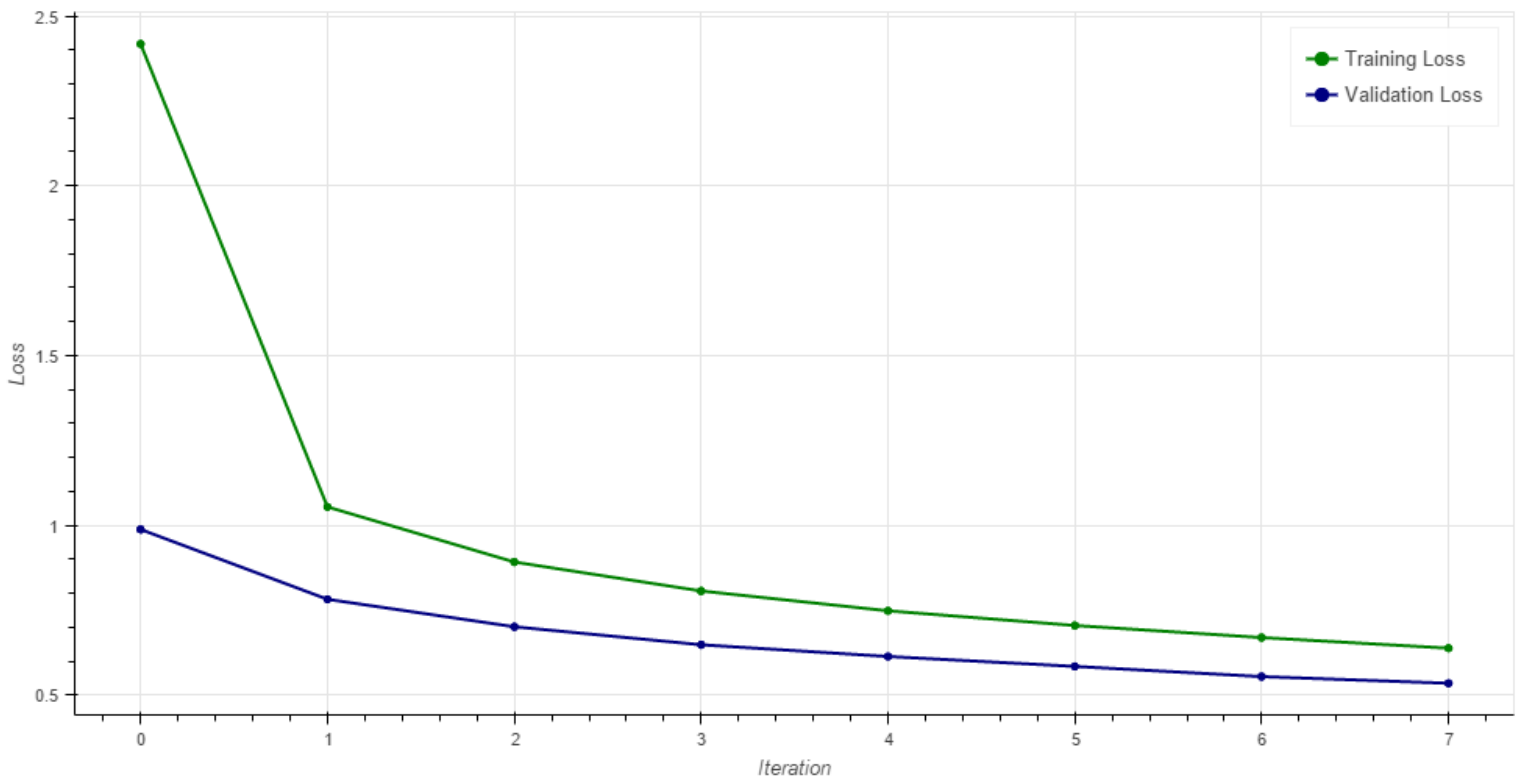


Figure 30: Loss during training where the model is built on balanced data and predicts the group name

Figure 30 describes the model's loss value to predict group name on both balanced validation and training data during 8 training times. The x axis is the number of

iterations, which is the number of times the model was trained. The y axis is the loss in decimal form. The red line demonstrates the trends for training data while the blue line demonstrates the trends for validation data. Similar as the explanation of Figure 26, both lines go downward during those 8 iterations, while the accuracies increase as shown in Figure 29. It indicates the model utilizes the right loss function and the model keeps improving during the 8 iterations. After training, the loss value for validation is 0.638 and the loss value for training is 0.638. Recalling that in Figure 26, the loss values of the model trained on unbalanced data after 8 iterations are around 1. This also confirms that the model trained on balanced data has better performance on balanced testing data than the performance the model trained on unbalanced data has on unbalanced data.

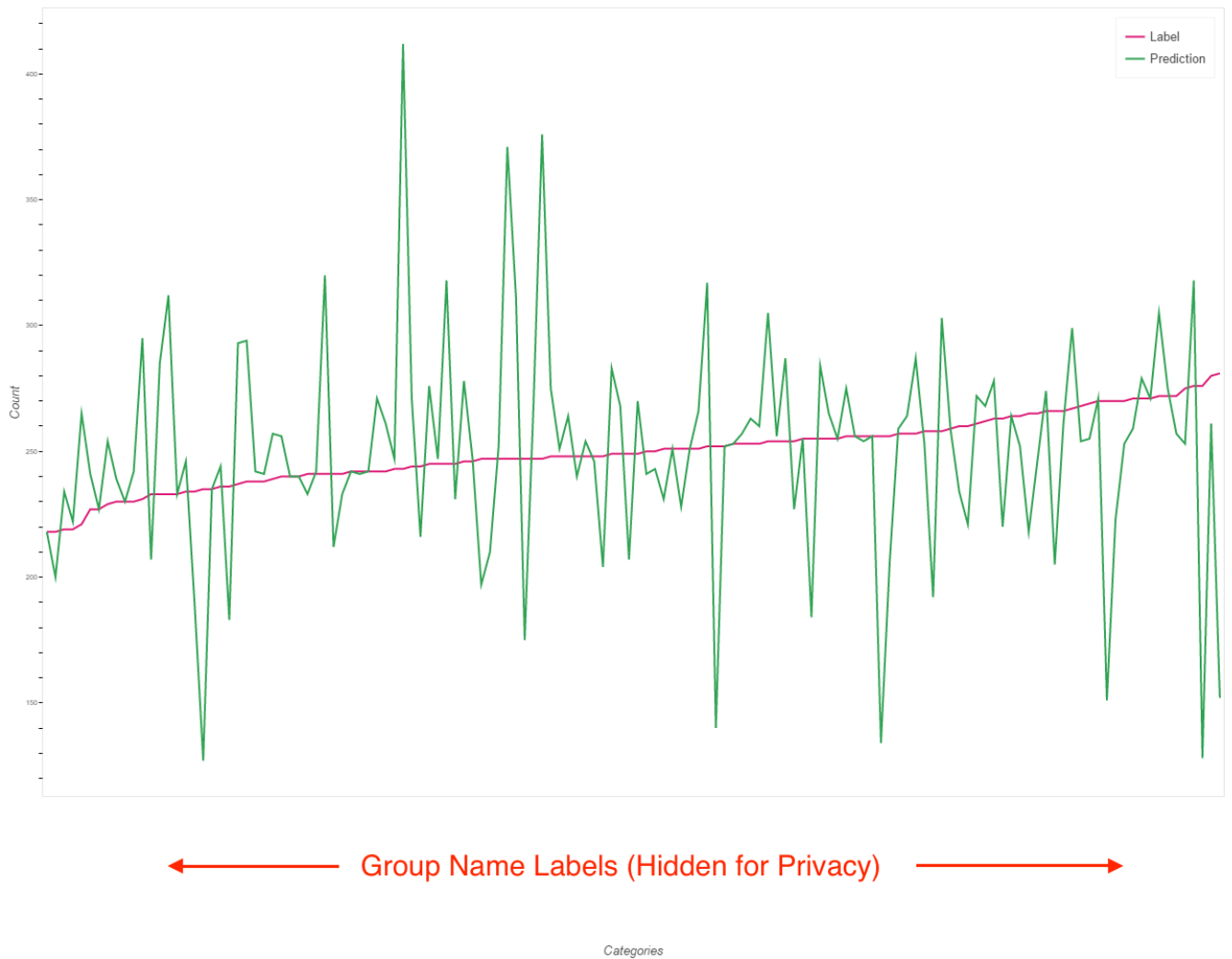


Figure 31: Distribution of group name predictions and labels for balanced testing data

25% of all balanced data was used for testing. The static model was loaded and testing data was passed through the model. Figure 31 shows the distribution of the predictions and labels. The x-axis is the group name and y-axis is the number of cases that solved by that group. For easier analyzing, it was sorted by the number of cases in labels. For example, for the group that solve most cases, its name was the right most value on x-axis. Since the red line describes the trending for labels and green line shows the trending for predictions, the red line smoothly increases while the green line vibrates around the red

line. Because the data was balanced by the number of group, which means each group has similar number of records, the red line is almost horizontal. The y-axis value is in range from 0 to about 400 and the red line is around 250. Therefore, the maximum difference between red line and green line is not more than 200. In Figure 27, which indicates the distribution of labels and predictions produced by the model trained on unbalanced data, the maximum difference between red line and green line is 400. Therefore, the distribution of predictions produced by the model trained on balanced data has less noise signals than the one produced by the model trained on unbalanced data. In other words, the similarity between the distributions of labels and predictions produced by the model trained on balanced data is bigger than the similarity between the distribution of labels and predictions produced by the model trained on unbalanced data.

Confusion Matrix

Prediction

Predicted Group Names

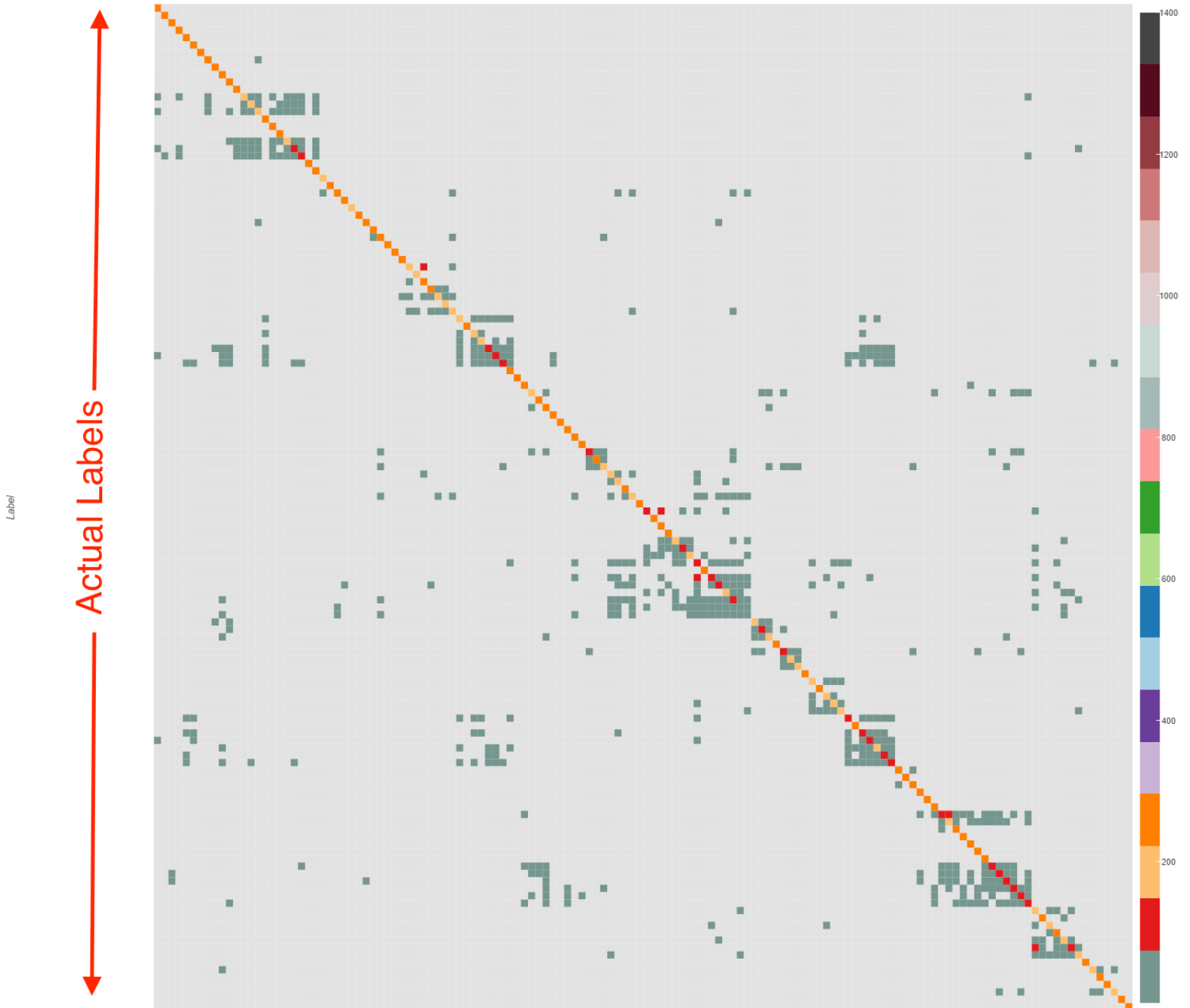


Figure 32: Confusion matrix of group name for balanced testing data

Figure 32 is the confusion matrix, drawn on balanced testing data. The x-axis is the group name in prediction while the y-axis is the group name in labels. Although there are outliers, there is a obvious diagonal line, which indicates the model finds the underlying relationship between the features and group name and can make an overall correct predictions. Comparing to Figure 32, there was less outliers in this plots and more dense points in the diagonal line, which means this model has high accuracy and better performance on testing data than the model trained on unbalanced data.

4.2.4 Keras Model unbalanced_engineerName

This model was built on unbalanced data to predict engineer name.

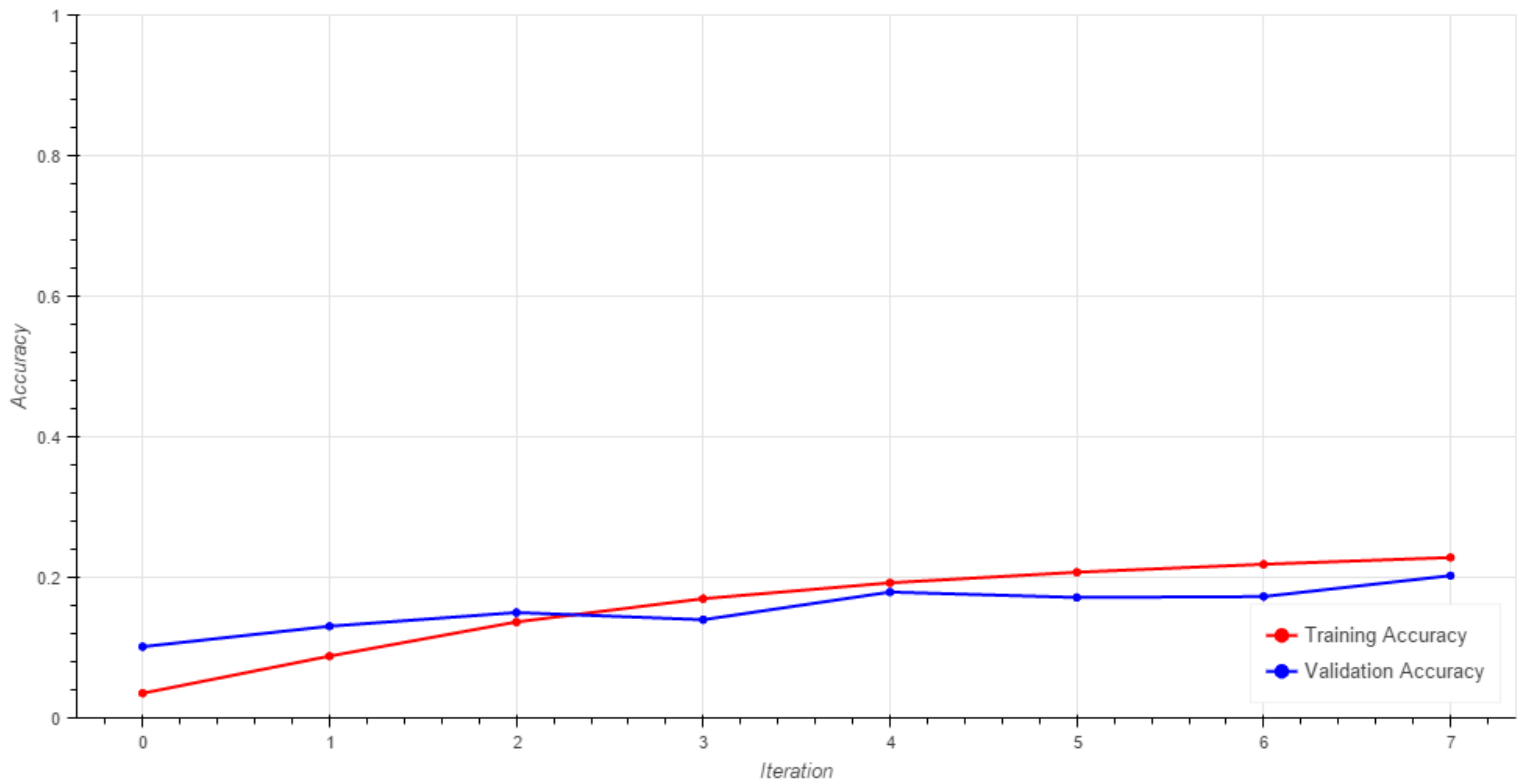


Figure 33: Accuracy during training where the model is built on unbalanced data and predicts the group name

Figure 33 describes the model's accuracies to predict group name on both unbalanced validation and training data during 8 training times. The x axis is the number of iterations, which is the number of times the model was trained. The y axis is the accuracy in decimal form. The red line demonstrates the trends for training data while the blue line demonstrates the trends for validation data. Similar to the explanation of the Figure 25, both lines go smoothly upward meaning the model keeps improving without much overfitting. After 8 iterations, the accuracy of validation data is 20.3% and the accuracy of training data is 22.9%.

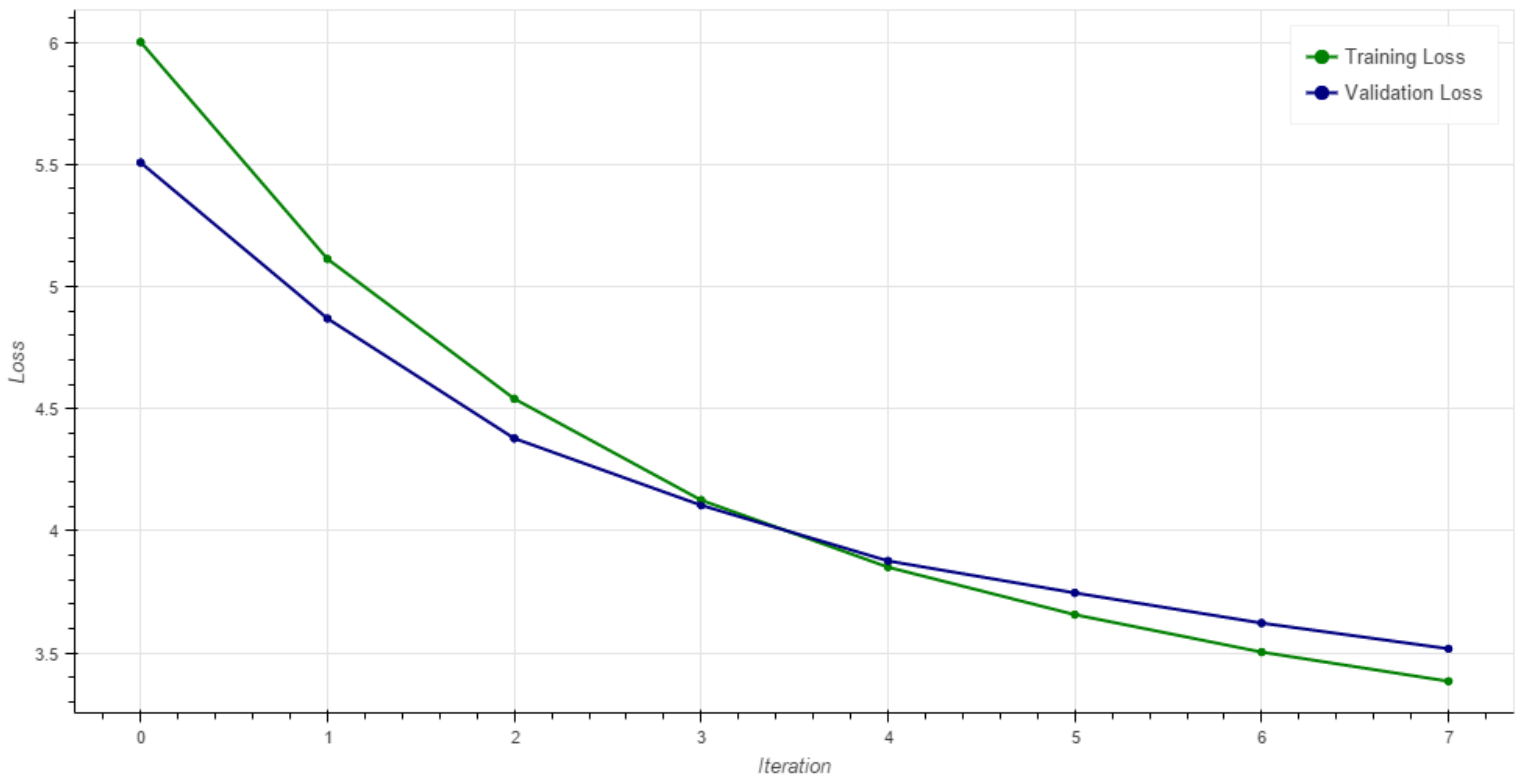


Figure 34: Loss during training where the model is built on unbalanced data and predicts the group name

Figure 34 describes the model's loss value to predict group name on both unbalanced validation and training data during 8 training times. The x axis is the number of

iterations, which is the number of times the model was trained. The y axis is the loss in decimal form. The red line demonstrates the trends for training data while the blue line demonstrates the trends for validation data. Similar as the explanation of Figure 26, both lines go downward meaning the model keeps improving and the right loss function was chose to build the model. After 8 iterations, the loss value for validation data is 3.517 while the loss value for training data is 3.384. Comparing to Figure 26 and Figure 30, the loss value for model unbalanced_engineerName is relative large, which also indicates this model has poor performance on the testing data.

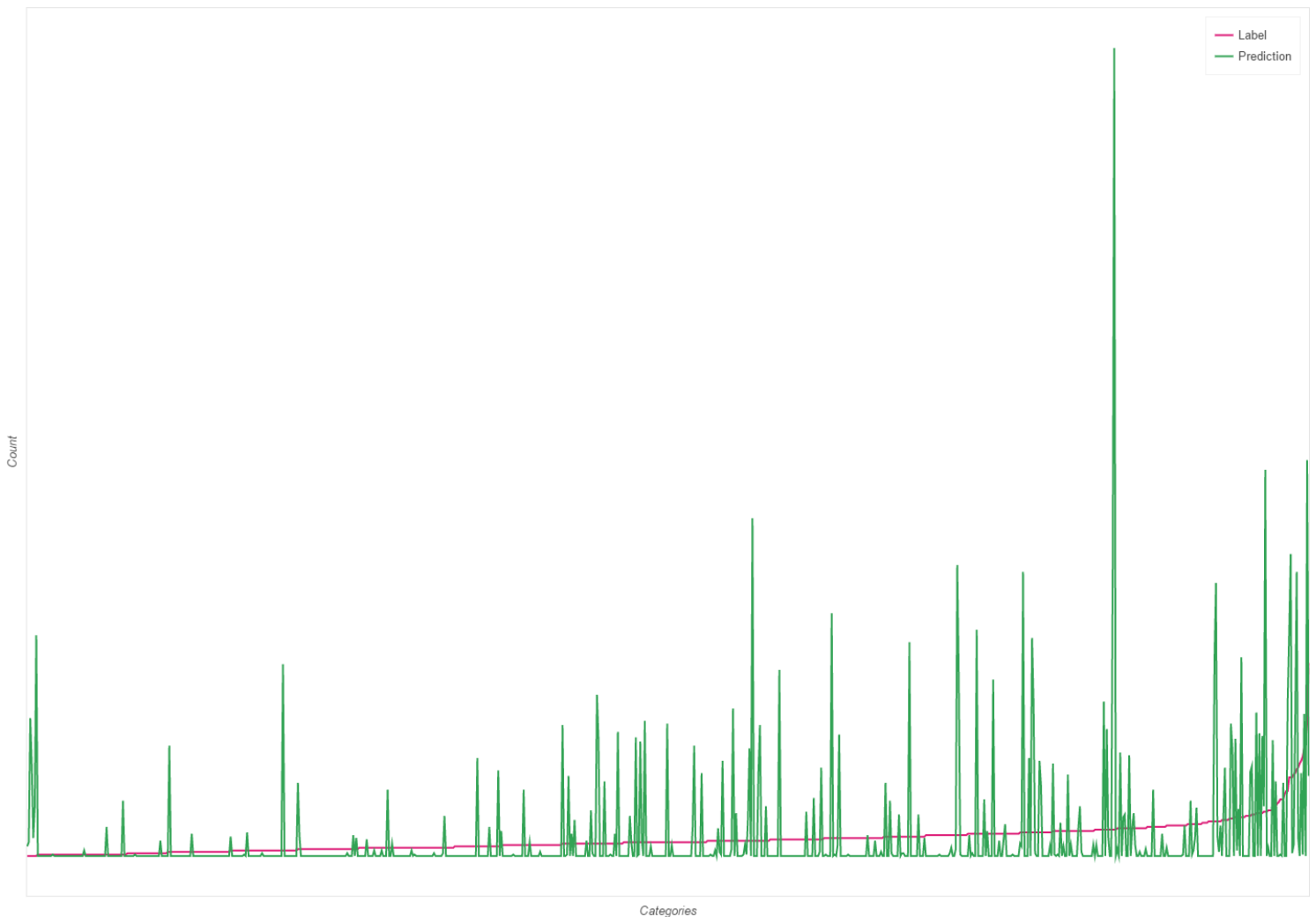


Figure 35: Distribution of group name predictions and labels for unbalanced testing data

25% of all unbalanced data was used for testing. The static model was loaded and testing data was passed through the model. Figure 35 shows the distribution of the predictions and labels. The x-axis is the engineer name and y-axis is the number of cases that solved by that group. For easier analyzing, it was sorted by the number of cases in labels. For example, for the group that solve most cases, its name was the right most value on x-axis. Since the red line describes the trending for labels and green line shows the trending for predictions, the red line smoothly increases while the green line vibrates around the red line. Figure 35 shows that most engineer only solve relative small number of cases, while a few of engineers solve lots of them. There are lots of noises in the green line, which means the model does not capture the underlying relationship perfectly.

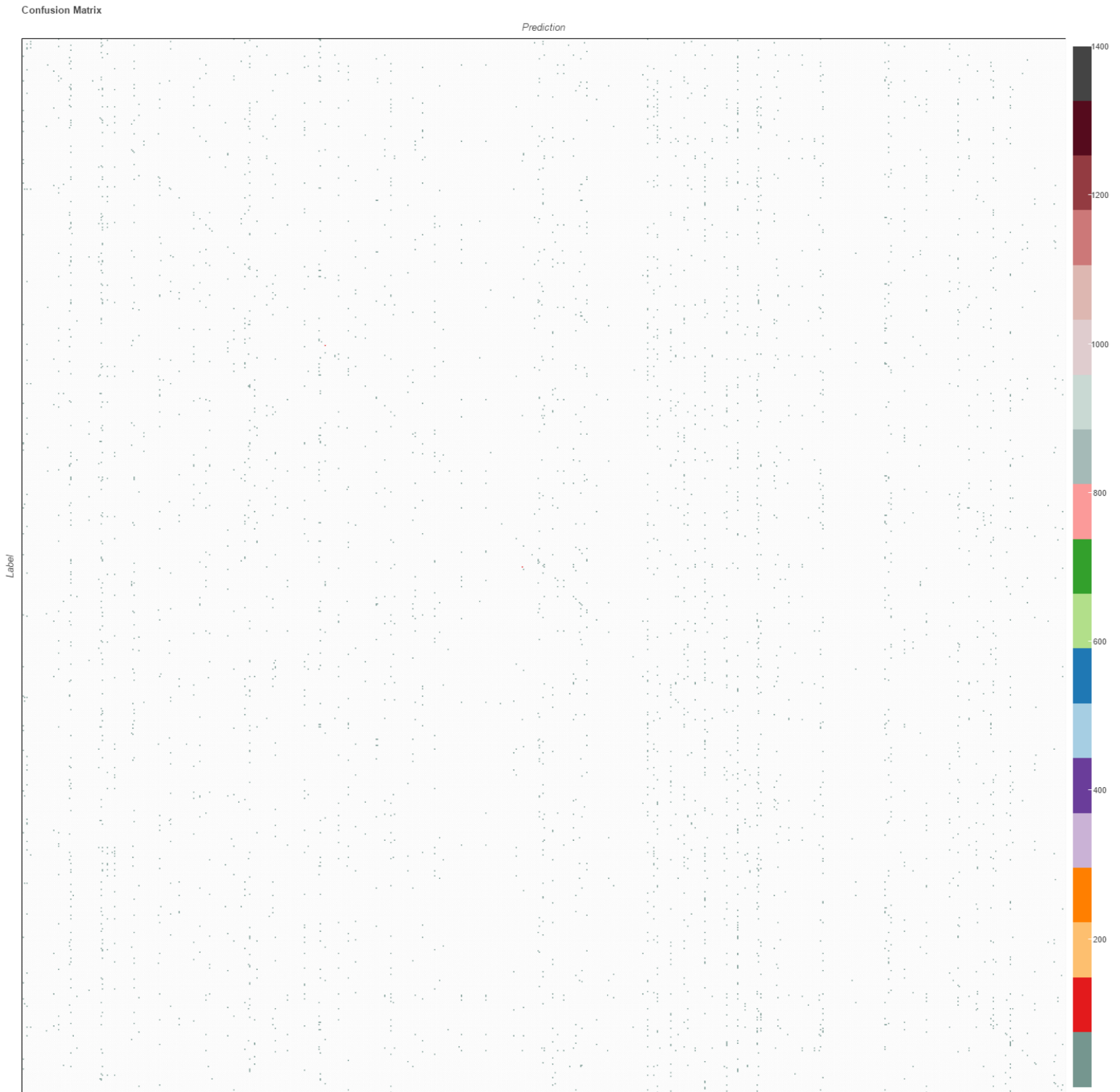


Figure 36: Confusion matrix of group name for unbalanced testing data

Figure 36 is the confusion matrix, drawn on unbalanced testing data. The x-axis is the engineer name in prediction while the y-axis is the engineer name in labels. The diagonal line can still be seen although there are a large numbers of outliers. It is also noticeable that there are some vertical lines in the graph. As shown in the distribution graph, a few engineers solved plenty of cases while most engineers only solved a few. Therefore, it is possible that the model likely to make predictions on those engineers solving plenty of cases.

4.2.5 Keras Model `balanced_engineerName`

This model was built on balanced data to predict engineer name.

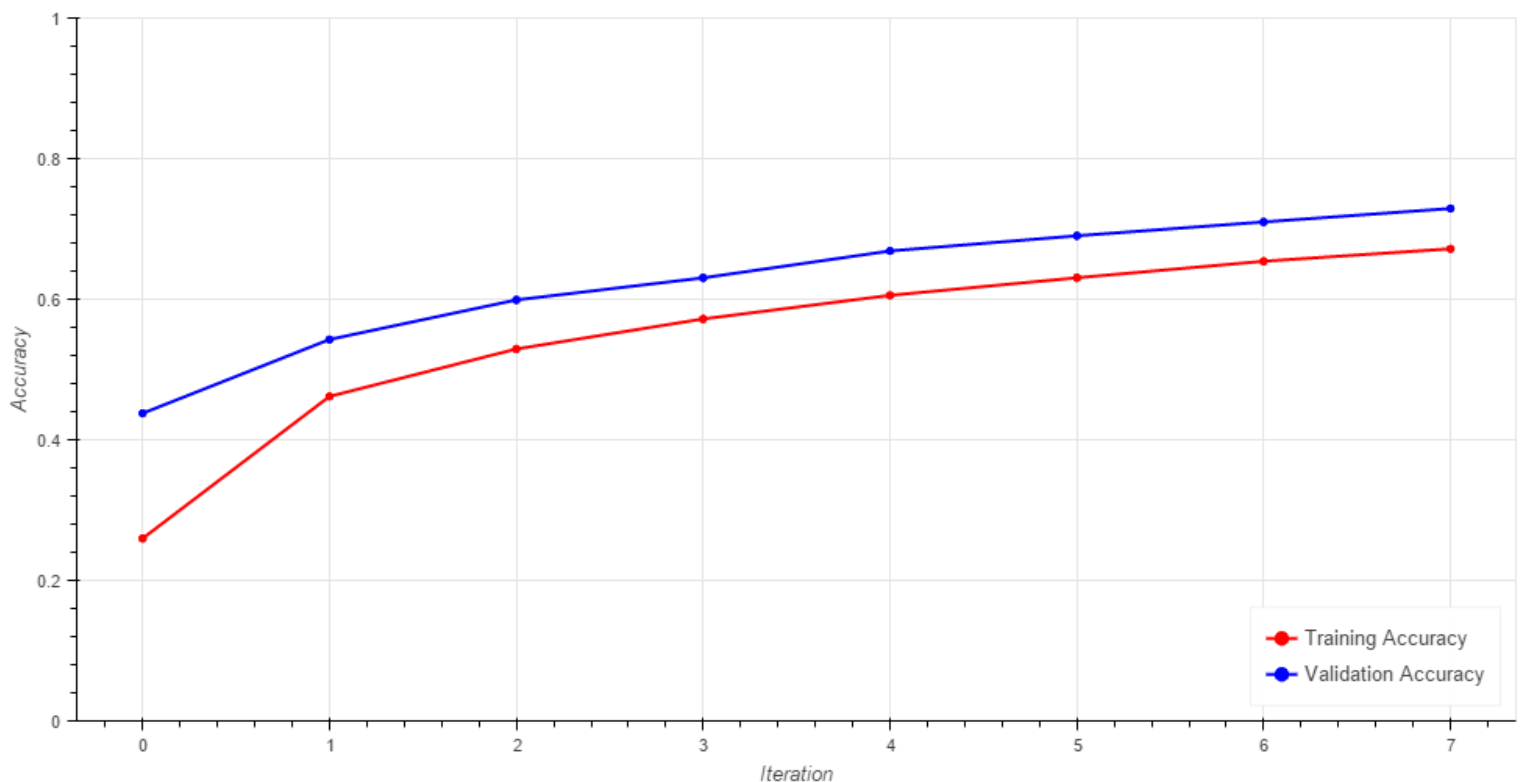


Figure 37: Accuracy during training where the model is built on balanced data and predicts the group name

Figure 37 describes the model's accuracies to predict group name on both balanced validation and training data during 8 training times. The x-axis is the number of iterations, which is the number of times the model was trained. The y-axis is the accuracy in decimal form. The red line demonstrates the trends for training data while the blue line demonstrates the trends for validation data. Similar as the explanation for Figure 25, both lines go upwards meaning the model keeps improving without much overfitting. After 8 iterations, the accuracy for the training data is 67.2% while the accuracy for the validation data is 72.9%.

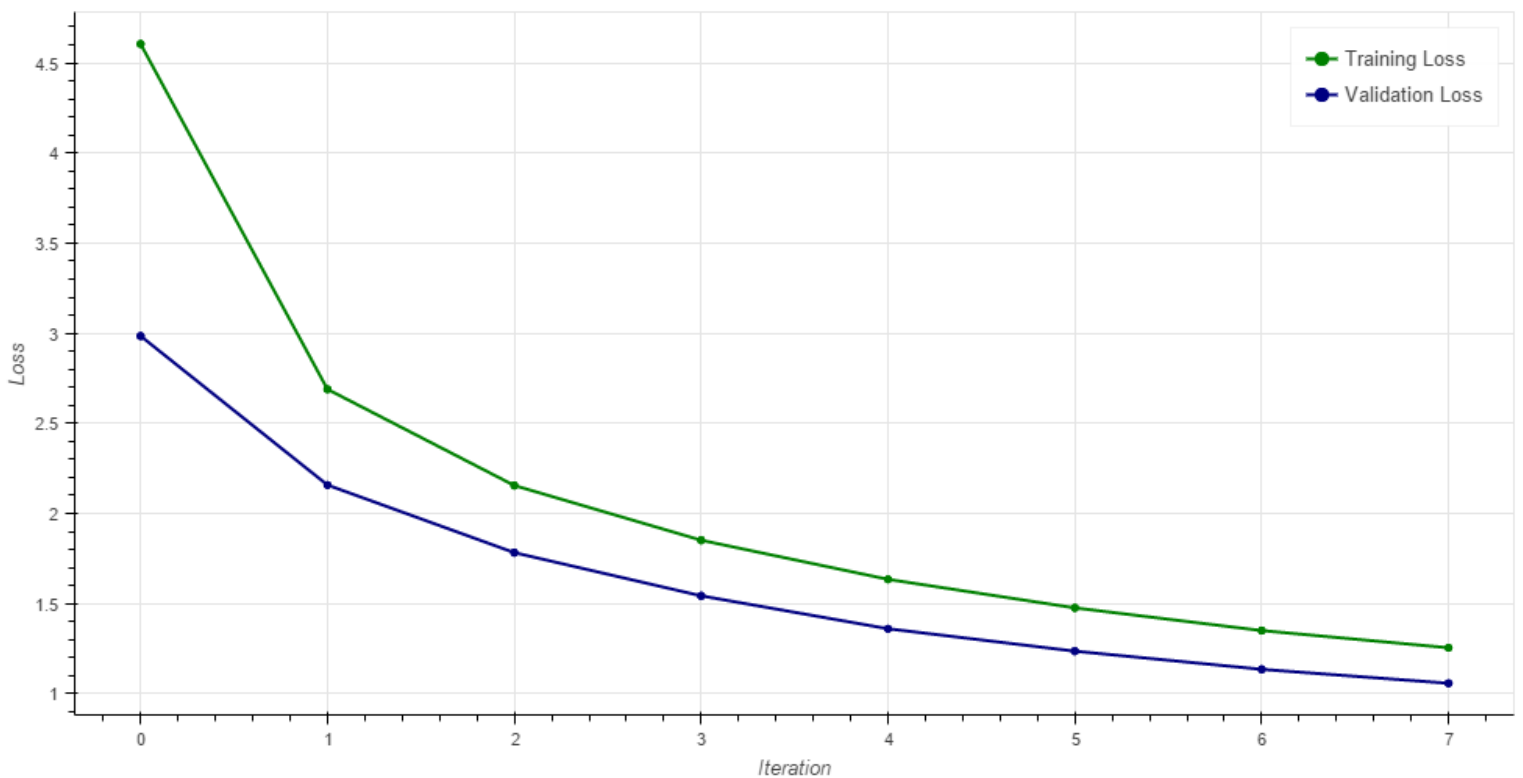


Figure 38: Loss during training where the model is built on balanced data and predicts the group name

Figure 38 describes the model's loss value to predict group name on both balanced validation and training data during 8 training times. The x axis is the number of

iterations, which is the number of times the model was trained. The y axis is the loss in decimal form. The red line demonstrates the trends for training data while the blue line demonstrates the trends for validation data. Similar to the explanation of Figure 26, both lines go downward meaning the loss values for both validation and training data decrease, which indicates the right loss function was chosen to build the model. After 8 iterations, the loss value for training data is 1.255 and the loss value for validation data is 1.058, which are much better than the value shown in Figure 34, which means this model has smaller error rate when predicts over balanced validation data than the model, which trained on unbalanced data, predicts over unbalanced validation data.

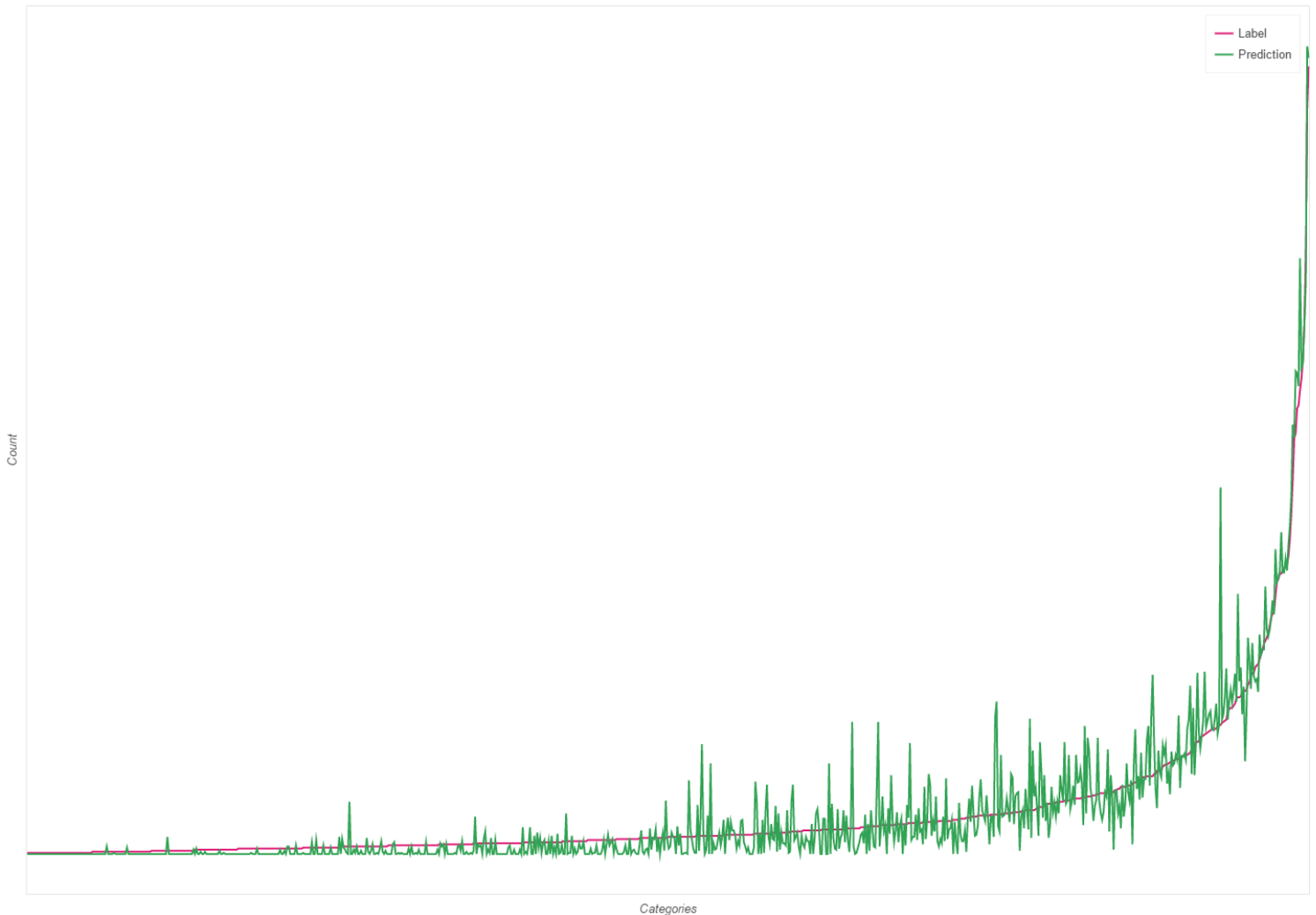


Figure 39: Distribution of group name predictions and labels for balanced testing data

25% of all balanced data was used for testing. The static model was loaded and testing data was passed through the model. Figure 39 shows the distribution of the predictions and labels. The x-axis is the engineer name and y-axis is the number of cases solved by that group. For easier analyzing, it was sorted by the number of cases in labels. For example, for the group that solved most cases, its name was the right most value on x-axis. Since the red line describes the trending for labels and green line shows the trending for predictions, the red line smoothly increases while the green line oscillates

around the red line. Comparing to figure 39 which shows the distribution for unbalanced data, the predictions in this plot has less noise and basically follows the trending of the red line. That means this model has a relative better performance on testing data than the one building on the unbalanced data.

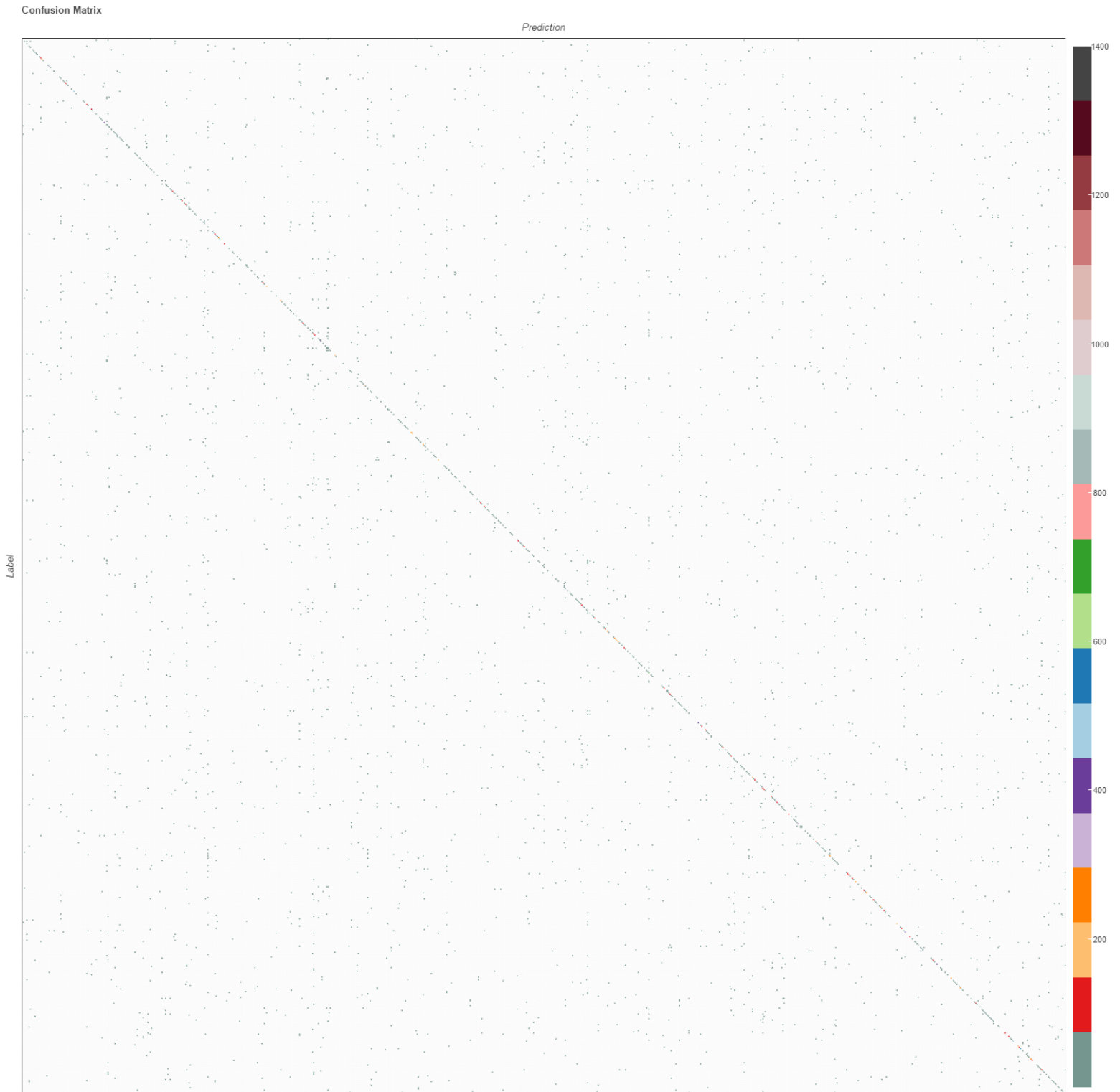


Figure 40: Confusion matrix of group name for balanced testing data

Figure 40 is the confusion matrix drawn on balanced testing data. The x-axis is the engineer name in prediction while the y-axis is the engineer name in labels. Comparing to 36, more points lies on diagonal lines, which indicates the model has a much better performance on testing data than the one building on unbalanced data.

4.2.6 Spark Model

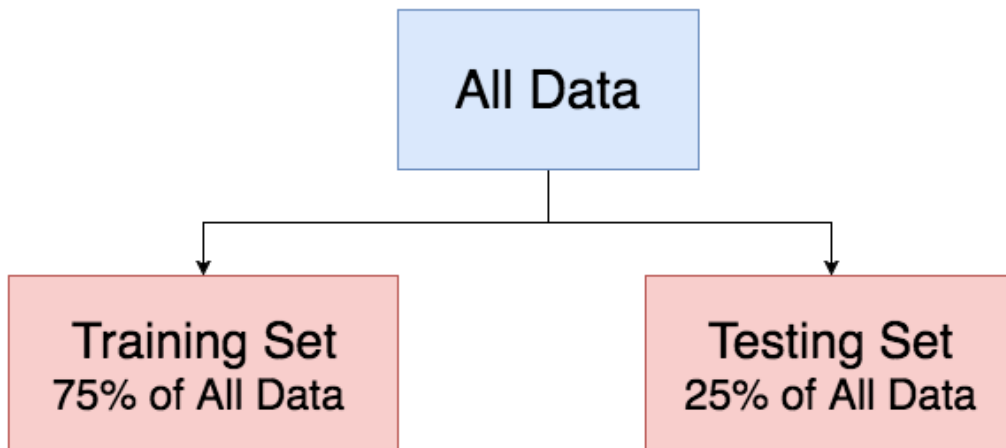


Figure 41: Data split for Spark model

Only one Spark model was built to predict group name. Currently, the Spark 2.1.0 API does not have the functionalities to get the top n answers. However, since the randomness and subtle underlying relationship between engineer name and cases, only predicting one engineer to solve the case is not useful and can not get a promising result. In such case, the Spark model was only used to predict group name.

As Figure 41 shows, there is no validation data set involved in building Spark model. As explained before, validation data was used for developer to analyze when the

overfitting occurs and manually changed the number of iterations that the model should be trained. However, in Spark, as shown in Figure 42, there is a parameter called "tol", which is the convergence tolerance of iterative algorithm. Therefore, in Spark, a maximum number of iterations was set and then when the convergence of tolerance became smaller than the specific value, in our case, 1×10^{-6} , the model was returned without further training.

```
class pyspark.ml.classification.MultilayerPerceptronClassifier(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, tol=1e-6, seed=None, layers=None, blockSize=128, stepSize=0.03, solver="l-bfgs", initialWeights=None) [source]
```

Figure 42: Spark multilayer perceptron classifier signatures
(spark.apache.org, 2017)

Since Spark API does not export accuracies and losses during the training process, only the accuracy of testing data was used to evaluate the model. 25% of all data was used for testing and 75% of all data was used for training. We run our coding to build the Spark model for 5 times. Each time the data was split randomly into 2 parts, 25% for testing and 75% for training. In other words, the Spark model was built using the same algorithm and parameters but on different data set for 5 times. Figures 43 to 47 shows the accuracies for those 5 times and the accuracy is around 62% which means the machine learning algorithm and parameters were well chosen and the result was stable.

```
Test set accuracy =0.6141889958996853
SUCCESS: The process with PID 5680 (child process of PID 9420) has been terminated.
SUCCESS: The process with PID 9420 (child process of PID 248) has been terminated.
SUCCESS: The process with PID 248 (child process of PID 15380) has been terminated.

Process finished with exit code 0
```

Figure 43: First Spark model build

```
Test set accuracy =0.6148084125612215
SUCCESS: The process with PID 9480 (child process of PID 6496) has been terminated.
SUCCESS: The process with PID 6496 (child process of PID 7148) has been terminated.
SUCCESS: The process with PID 7148 (child process of PID 2508) has been terminated.

Process finished with exit code 0
```

Figure 44: Second Spark model build

```
Test set accuracy =0.6224926971762414
SUCCESS: The process with PID 10952 (child process of PID 1628) has been terminated.
SUCCESS: The process with PID 1628 (child process of PID 3060) has been terminated.
SUCCESS: The process with PID 3060 (child process of PID 9776) has been terminated.

Process finished with exit code 0
```

Figure 45: Third Spark model build

```
Test set accuracy =0.6248303934871099
SUCCESS: The process with PID 12816 (child process of PID 4908) has been terminated.
SUCCESS: The process with PID 4908 (child process of PID 11836) has been terminated.
SUCCESS: The process with PID 11836 (child process of PID 13108) has been terminated.

Process finished with exit code 0
```

Figure 46: Fourth Spark model build

```
Test set accuracy =0.6256750355281857
SUCCESS: The process with PID 9508 (child process of PID 10500) has been terminated.
SUCCESS: The process with PID 10500 (child process of PID 504) has been terminated.
SUCCESS: The process with PID 504 (child process of PID 15816) has been terminated.

Process finished with exit code 0
```

Figure 47: Fifth Spark model time

4.2.7 Models Summary

In total we trained four Keras models, specified previously in subsection 4.2.1, they have been named the following for easy reference:

- **unbalanced_groupName**: A model trained with unbalanced data that predicted group name
- **balanced_groupName**: A model trained with balanced data that predicted group name
- **unbalanced_engineerName**: A model trained with unbalanced data that predicted engineer name
- **balanced_engineerName**: A model trained with balanced data that predicted engineer name

| Model name \ Number count as correct | Top1 (%) | Top3 (%) | Top5 (%) | Top7 (%) | Top10 (%) |
|--------------------------------------|----------|----------|----------|----------|-----------|
| unbalanced_groupName | 51.31 | 84.06 | 92.42 | 95.20 | 96.63 |
| balanced_groupName | 80.95 | 95.49 | 98.39 | 99.36 | 99.75 |
| unbalanced_engineerName | 10.55 | 21.89 | 30.43 | 36.49 | 42.69 |
| balanced_engineerName | 72.93 | 85.26 | 89.44 | 91.66 | 93.48 |

Figure 48: Accuracy summary for four models

In Figure 48, the "number count as correct" means if it predicts a number of labels, n , and one of them is the same as actual label, we count this prediction as correct. For example, the "top3" means that for each case, the model predicted the 3 most possible answers, and if one of them is same as the label, we count this prediction as correct.

Only one Spark model was constructed and it was built on original unbalanced data used to predict one single group name. The overall accuracy for this Spark model on testing data is around 62%. The accuracy of the Keras model trained on unbalanced data to predict one group name is around 51%. In other words, the Spark model did have a better performance to make prediction about group name over unbalanced data set, but was limited in functionality, so we didn't move forward with it.

5 Conclusion

This section draws conclusions about our work and results, and how our project's work can be expanded in the future.

5.1 Conclusions

Time is an important asset to any company and making customer support faster and potentially more efficient could have positive business implications. When a new customer support case is opened with Juniper Networks, it can be routed to several employees before reaching an engineer ready to solve it. This time could be shortened and the routes could be made more direct through automation.

Our project's goal was to explore and analyze customer support case routing and to help automate the process. Initially, we analyzed all the resolved cases in 2016, to understand the data set we'd be working with. We were able to discern that several categorical fields would be useful in defining a pattern. Further, we were able to see the distribution of groups and engineers responsible for solving the cases, which foreshadowed some biases in a machine learning approach. Next, using 2016 resolved cases we applied machine learning techniques to model the the customer support case routing currently in place. This has the potential to enable Juniper's customer support department to improve the turnaround time for a customer support case by assigning it to an engineer or group that has resolved similar cases in the past. We were able to prototype a pipeline for using past resolved cases to predict the engineers who could solve incoming cases. Our Intelligent Case Router is meant to automate the process of assigning cases, within the current scope of a assisting human. We used multiple machine learning approaches, mainly a deep learning neural network, to create a model

for the data set we chose. The Intelligent Case Router we created works quickly, and has extensive support for expansion.

To accurately make predictions we first had to model the relationships between a case's categorical information and group responsible for solving it. Having done that, we then trained a model to find the patterns between the case's categorical information combined with the responsible group so that we could then predict an engineer. Given these models, we then created a system where we would predict a group for a given case, and based off the case and the predicted group, we could finally predict an engineer. Using a subset of our data strictly to test the predictive abilities of our models, we were able to achieve an accurate prediction rate of about 50% for group names out of 124, when predicting 1 value, and about 10% for engineers out of 941 names. Because our Intelligent Case Router is meant to serve in a human assisting role, we also had examined the accurate prediction rate for group names, and engineers when predicting the top 3, 5, 7 or 10 most likely results. Interestingly, when we look for the correct label in the top 3 predictions for group names, our accuracy for group name predictions reach 84%, and 22% of engineers. This could be helpful in helping internal customer support employees route this case. Lastly, as can be seen below in Figure 48, we calculated the accuracy rate for two versions of our data set, the original data set, and one with an adjusted distribution of label values.

Lastly, to demonstrate the work we did to employees who could benefit from this work, we created a web-based application that uses our models. This application is the frontend of our Intelligent Case Router and its abilities. This last step provides tangible output to people who would use this system as well as a means of getting feedback on its effectiveness.

5.2 Future Work

We completed this project with expandability in mind, and a list of goals and accompanying steps that may build upon our work.

5.2.1 Goals

Given our work, and the resulting Intelligent Case Router system future work may include:

- **Expanding the Initial Data:** We worked with resolved cases in 2016 alone, and it would benefit the models to be exposed to a larger set of data points. With more data, more precise approximations can be made to understand the relationships between the cases and the engineers responsible for solving them.
- **Extracting Topical Information from Notes:** The case notes are conversation about a customer support case, and are an important piece in determining more about the case, and getting to the final destination. Future work should find a way to extract categorical information from the conversation for each case, and apply it to any natural language information in a new case.
- **Modeling the Full Flow of a case:** This project focused on finding the relationship between a customer support case and the engineer fixing the problem, but when a real case is routed, it undergoes many changes and goes through several statuses. To more precisely assess this process, future work should try to model the entire life cycle of a customer support case.
- **Build a Tool that Interfaces with Other Tools in Use:** Currently the only means of demonstrating our model's predictive abilities is the web application. Future work could include creating a more comprehensive application that can interact with tools that are already used to track and work with customer support cases.

5.2.2 Suggested Work

We have compiled a list of actionable steps that can be taken initially to achieve the goals outlined in the previous section.

Expanded Data Set To achieve this, future work should start by including previous years data from the database, which provides additional context. After that, new work could include a way to dynamically add new data to system. As new cases are resolved, the models should be trained again, perhaps as often as every night.

Natural Language Processing Due to our limited time our models were built purely around the "details" data. It is possible that a further investigation of the "notes" data would improve the models. Currently, we experimented with a naive word2vec method to explore the "notes" only on Juniper Networks' data set. However, it is likely that performing word2vec on a larger representation of the English language would provide a more accurate representation of natural language. For instance, by training word2vec using Wikipedia, a large set of natural language entries, the model could be using a stronger encoding scheme for case notes and user generated problem descriptions. Other, more sophisticated natural language techniques could also be used to explore "notes". For example, if one were to train a model that can pay attention to the importance of words or phrases in text, there may be a strong improvement in classification of customer support cases using natural language information from conversations between customer support employees and customers.

Case Life cycle Modeling If an Intelligent Case Router could better understand the current life cycle of any given customer support case, it may have a better abstract understanding of how the case ends up being resolved. For instance if we model the

reasoning why one case is moved laterally from group to group, or from engineer to engineer, a stronger relationship could be formed between the case and the final destination. Additionally, if more categorical information added as the case is routed, perhaps some of the categorical information could actually be predicted based on others, and the overall model could have a better understanding of cases simply based on less information.

End User Application If our Intelligent Case Router's Pipeline could be abstracted into a packaged application, it may better serve the purposes of the system. By making the pipeline abstracted, it would require less interaction from a human and would be able to dynamically assess a new case and route it immediately. Further, additional designs may provide an application with a more user friendly UI that requires very little input from the user.

References

- Chollet, F. (2017a). *Good news, Tensorflow choses Keras!* Retrieved 2017-02-24, from <https://github.com/fchollet/keras/issues/5050#issuecomment-272945570>
- Chollet, F. (2017b). *Keras*. Retrieved 2017-02-24, from <https://en.wikipedia.org/wiki/Keras>
- Chollet, F. (2017c). *keras.io*. keras.io. Retrieved 2017-02-24, from <https://keras.io/>
- Continuum.io. (2017). *Why Python for Big Data?* Continuum Analytics. Retrieved from <https://www.continuum.io/why-python>
- Google. (2017). *Tensorflow*. tensorflow.org. Retrieved 2017-02-24, from <https://www.tensorflow.org/>
- Jain, A., Gohil, A., Chun, T., & Zhou, Y. (2017). *Intelligent case routing*.
- Juniper Networks. (2017). *Juniper Networks*. Retrieved from <https://www.juniper.net/us/en/company/profile/>
- Karpathy, A. (2015). *Cs231n open course*. Retrieved from <http://cs231n.github.io>
- Karpathy, A. (2016). *Linear classification*. Retrieved from <http://cs231n.github.io/linear-classify/>
- Krewell, K. (2009). *What's the difference between a cpu and a gpu?* nvidia. Retrieved from <https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/>
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet classification with deep convolutional neural networks*. quora.com. Retrieved from <http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>
- Metz, C. (2015). *Google Just open sourced TensorFlow, its artificial intelligence engine*. Wired Magazine. Retrieved from <https://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/>
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality*. Google. Retrieved from <https://arxiv.org/abs/1310.4546v1>
- NodeJs Foundation. (2017). *About Node.js*. Author. Retrieved from <https://nodejs.org/en/about/>
- Numpy.org. (n.d.). *Numpy.org*. numpy.org. Retrieved 2017-02-24, from <http://www.numpy.org/>
- Peterse, Y. (2015). *Goodbye mongoddb, hello postgresql*. olery.com. Retrieved from <http://developer.olery.com/blog/goodbye-mongoddb-hello-postgresql/>
- Rauschmayer, A. (2014). *Chapter 2: Why javascript?* O'Reilly Media Inc. Retrieved from <http://speakingjs.com/es5/ch02.html>
- Rouse, M. (2011). *Natural language processing*. TechTarget. Retrieved from <http://searchcontentmanagement.techtarget.com/definition/natural-language-processing-NLP>

- Rouse, M. (2013). *Hadoop Distributed File System (HDFS)*. TechTarget. Retrieved from <http://searchbusinessanalytics.techtarget.com/definition/Hadoop-Distributed-File-System-HDFS>
- Rouse, M. (2016). *Hadoop*. TechTarget. Retrieved from <http://searchcloudcomputing.techtarget.com/definition/Hadoop>
- Sas.com. (2016). *Machine learning: What it is and why it matters*. Sas Institute, Inc. Retrieved from http://www.sas.com/en_us/insights/analytics/machine-learning.html#machine-learning-users
- Scikit-learn.org. (n.d.). *scikit-learn*. scikit-learn.org. Retrieved 2017-02-24, from <http://scikit-learn.org/stable/index.html>
- Scipy.org. (n.d.). *Scipy getting started*. scipy.org. Retrieved 2017-02-24, from <http://https://www.scipy.org/getting-started.html>
- Sirer, E. G. (2013). *Broken by design: MongoDB fault tolerance*. hackingdistributed.com. Retrieved from <http://hackingdistributed.com/2013/01/29/mongo-ft/>
- spark.apache.org. (2017). *Pyspark documentation*. Retrieved from <http://spark.apache.org/docs/latest/api/python/pyspark.ml.html>
- Strand, H. H. (2016). *What is one hot encoding and when is it used in data science?* quora.com. Retrieved from <https://www.quora.com/What-is-one-hot-encoding-and-when-is-it-used-in-data-science>
- Tensorflow.org. (2016). *Vector representations of words*. Retrieved from <https://www.tensorflow.org/tutorials/word2vec>
- The Apache Spark Foundation. (2016). *Apache Spark*. Retrieved from <https://spark.apache.org/>
- wildml.com. (2015). *Recurrent neural networks tutorial, part 1 – introduction to rnns*. Retrieved from <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>