Intelligent Case Classification

April 2023

H19570

White Paper

Abstract

This white paper presents Intelligent Case Classification, an advanced Natural Language Processing and Machine Learning-driven text classification approach for analyzing Dell customer support interactions. ICC uncovers issue patterns, amplifies operational efficiency, elevates customer satisfaction, and enhances operational productivity, culminating in considerable business benefits and cost

Dell Technologies Solutions

Copyright

Contents

Introduction

In this white paper, we present a text classification solution known as Intelligent Case Classification (ICC) to assist categorization and grouping of various technical issues faced by customers related to their Dell products.

Dell Support generates millions of text records weekly through interactions with customers using different communication channels including email, phone conversations, and chat. Critical information related to each interaction (also known as a support case) is logged in textual format through a complex Customer Relationship Management (CRM) system. Manually sifting through large volumes of case logs (logged textual data) to extract insights such as the identification of most frequently occurring issues is tedious and time-consuming.

ICC automates the process of classifying these case logs into a hierarchical issue taxonomy using Natural Language Processing (NLP) and Machine Learning (ML). ICC facilitates the identification and analysis of the intents and root causes behind customers contacting Dell Technologies and enables profiling of issues. The introduction of a standardized issue taxonomy helps identify issue patterns, trends, and their correlation to various processes and tools so that effective issue preventive strategies can be implemented. Moreover, precise identification of technical issues helps to identify and predict different actions or resolutions steps that can be taken to resolve issues.

Overall, ICC improves customer experience by reducing average handling time for support agents leading to improved operational efficiency. This is done by tracking the issues that need to be fixed and thereby improving the resolution time. surfacing trends and patterns of issues so that strategies can be developed to address and prevent such issues.

Document purpose

This white paper presents an overview of the ML modeling strategy and describes the model deployment architecture in detail. It also shares some example business outcomes using ICC and describes the directions for future research.

We value your feedback

Dell Technologies and the authors of this document welcome your feedback on the solution and the solution documentation. Contact the Dell Technologies Solutions team by email.

Author: Asanka Wasala, Kshitiz Khatri, Prashant Shaw, Nick Caldwell, Ahmed Hani, and Konark Paul

Contributors: Rohitt Punjj, Tejas Naren, Peter Gautam, Shilna Sasheendran, Chad McGougan, Martin Duggan and Jeannie Fitzgerald, and Rana Moussa

Note: For links to additional documentation for this solution, see Dell Technologies Solutions Info Hub for Artificial Intelligence.

Methodology

This section describes the methodology adopted for building the solution and the current architecture in detail.

Data set curation

First, a supervised dataset consisting of case logs and applicable target issue categories was built. This step is primarily because supervised ML techniques offer straightforward experimentation paths and ease of evaluation due to the availability of ground truth.

The dataset generation task is broken down into two subtasks:

- Derivation of the standardized issue taxonomy
- Building of supervised training dataset.

Standardized issue taxonomy

This task involved construction of a hierarchical taxonomy for standardization of technical issue descriptions. For this purpose, we used several existing technical issue taxonomies used by different business analytics applications and teams. In parallel, we performed an Exploratory Data Analysis (EDA) of case logs to understand the detailed requirements of taxonomy including capturing technical issue semantics, nomenclature, and minimizing duplication and the number of categories (see Exploratory data analysis). We opted for three-tier label hierarchy for describing each technical issue, where the:

- First level (known as T1) is the least granular level, which represents an overall issue category (for example, boot related)
- Second level represents subissue types within first level (known as T2)
- Third tier represents the most granular and specific issue information (known T3).

For example, consider the following T1-T2-T3 label:

Power | AC Adapter | Adapter Noise Issue

Here, T1 = [Power] is the least granular level of information and under power; T2 = [AC Adapter], and T3 = [AC Adapter Noise Issue] is most granular level of information provided.

The combination of the *T1-T2-T3* label hierarchy can succinctly describe over 85 percent of frequently occurring technical issues. In addition to deriving labels with the aid of EDA, we liaised with the domain experts to consolidate several exiting taxonomies and curate a set of all-encompassing *T1-T2-T3* labels.

Furthermore, a set of syntactic rules were established on how to format each *T1-T2-T3* label. These rules ensure future conformity and standardization as new labels are added. Example formatting rules include:

- Use title case and capitalize all abbreviations and acronyms
- Use the pipe “|” character as the delimiter for each label component (so that commas and hyphens can be preserved in a label component if required)
- Add spaces around the pipe | delimiter.
- Remove spaces around the “/”, “-“and “:” characters.
- Replace “and” with the ampersand “&”, to make the label shorter.

Table 1. Example T1-T2-T3 Labels for describing consumer technical Issues

Tier 1	Tier 2	Tier 3
Bluescreen	After OSRI	On Boot
Backup & Data Management	Backup Products & Services	Drivers
Chassis	Damage	Hinges
Display (internal)	Damage	LCD bezel

A major concern of having a fixed set of T1-T2-T3 labels for describing technical issues is that novel and unseen issues may not be represented or captured by downstream ML models. To address this concern, a label governance process has been established that ensures new standardized labels are introduced periodically while minimizing duplication with respect to existing labels. Another concern is the growing number of labels over time with the new additions, and therefore the governance process also includes steps to invalidate or remove infrequent or outdated labels.

Supervised raw training dataset

Having defined issue taxonomy and a comprehensive set of labels, the next step builds an ML training dataset.

As mentioned in the Introduction, Dell support agents use CRM to record customer issues, queries, requests, and other important diagnostic/troubleshooting notes, which are known as “Case Logs.” Case logs are free text entered by agents that contain information about customer-facing issues, diagnostic steps, and resolution recommendations. Therefore, we decided to use these logs for training our ICC model. The logs are saved in a database along with other key information related to products and troubleshooting activities.

To build the training dataset, two approaches were used:

- The first approach is a CRM system was augmented to include a new user interface (UI) field for technical issue classification (alongside the case log field). Next, this feature was rolled out to a selected set of support agents, where they were asked to annotate each case log with appropriate standardized label as a part of their support activity workflow. Label search functionality was implemented to facilitate selection of an existing label; however, support agents were also allowed to enter their own label.
- The other approach involved in-house development of a dedicated data annotation tool with active learning capability known as Elixir. The latter was deployed internally and used by 10 to 15 subject matter experts, who annotated a substantial number of (>40k) textual case logs within a brief period. The functionality and implementation of the tool is described in Elixir – A Novel Data Labelling and Prediction Validation Tool.

The results are multiclass datasets for consumer issues with approximately 24 K records in English, Mandarin, and Portuguese collected during 2019 to 2020. Each record primarily consists of two textual fields, which make the source: log title, log description and the target into one or more standardized T1-T2-T3 labels. In addition, a primary key was also stored for traceability – linking to each diagnostic/communication activity.

Exploratory data analysis

Main tasks of EDA involved defining noise in case logs and performing class distribution and textual distribution analysis.

In addition to performing *n*-gram (unigram, bigram, and trigram) analysis on case logs, we also performed analysis on the average word count, character count, and language distribution analysis. These analyses were instrumental in building optimal models (see Model Experimentation) and understanding how to identify and handle lengthy or extremely short text logs.

Noise analysis helped in defining necessary preprocessing steps. For example, it was noticed that case logs contain many acronyms and abbreviations introduced by support agents. Misspellings were also common. DellNP, an NLP product previously developed in-house, effectively addressed these challenges. It was also noticed that some case logs contain machine-generated content such as the logs from different diagnostic tools. Strategies were identified to extract relevant information from such logs and remove redundant content that can lead to model confusion. Unigram analysis was also useful in identifying and building a custom stop-word list for minimizing noise.

We also performed an in-depth analysis of target label distribution. As anticipated, we noticed a huge class imbalance. For example, the most frequently occurring label had 480 instances, whereas the least frequently occurring label only had nine instances in the dataset. The analysis also revealed that the nature of the problem was multiclass classification.

EDA was also helpful in proposing some new labels. One such label we added is "*Not Sufficient Information".* It is useful when a case log presents minimal to no information about a technical issue. We also noticed that depending on the level of technical details present in a case-log, the target label needs to be carefully drawn from the *T1-T2-T3* label hierarchy that is, T1 – minimal information present, or *T1-T2* - some technical/issue details are available or *T1-T2-T3* – all necessary details present so that we can precisely identify the issue.

Model experimentation

An ML model must have sufficient examples for each class to "learn" how to make inferences for that class on unseen text. Therefore, we decided to use a minimum threshold of 10 examples for each label, which eliminated some rarely occurring issues from our dataset. The total number of classes remaining was 234.

We conducted a time-bound literature review to understand the different techniques that would be most beneficial for the use case involving multiclass text classification of documents with variable character length and class imbalance. In addition to exploring academic literature, leaderboards of various NLP benchmarks such as superGLUE were analyzed to identify applicable NLP techniques. During the analysis of leaderboards, we discovered that XLNET outperformed other NLP techniques in most of the tasks. However, classical ML techniques and deep learning (DL) text classification approaches, including hierarchical classification techniques were also identified as candidate techniques handling text features, a large number of classes, and class imbalance.

Next, we designed key experiments for attempting different ML/DL techniques, text preprocessing steps, featurization of text, and handling class imbalance. The techniques experimented include tree-based methods (for example, Catboost, Random Forest), Support Vector Classifier (SVC), XLNET, and DL techniques for hierarchical and imbalance classification. Additional experiments included the application of pseudolabeling strategies to address the lesser amount of training data for certain classes, and application of Synthetic Minority Oversampling Technique (SMOTE) to treat the class imbalance. However, the latter techniques such as pseudolabelling and SMOTE did not lead to considerable performance improvements. For representing the text, we attempted word/character count-based vectors, word/character Term Frequency-Inverse Document Frequency (TF-IDF) vectors, as well as word/document embeddings.

The following table summarizes model performances of the key techniques attempted (for English model):

Table 2. Model experiment

Experiment	Macro-fscore
Textcnn	0.136225
Textrnn	0.106093
Textrcnn	0.098848
FastText	0.009733
AttentionConva	0.110873
Transformer	0.059808
TextVDCNN	0.003865
DRNN	0.022878
XLNET	0.1442
SVC	0.25

Similar models were developed for Mandarin and Portuguese and used custom preprocessing for each language. Along with looking at macro-fscore, Normalized Discounted Cumulative Gain (NDCG) for three models, known asT1, T1-T2 and T1-T2-T3 models, was also calculated. Based on the experimentation results and considering the resource requirements for model deployment, SVC was chosen as the final model. SVC model was trained using TF-IDF features. TF-IDF penalizes the high-frequent and common words that are not useful for classification. The combination of character n-grams TF-IDF features and SVC provided the best overall results for this use-case when compared to other model-feature set combinations. The NDCG score achieved for T1, T1-T2 and T1-T2-T3 models were 0.9, 0.8 and 0.6 for the top three labels. The NDCG metric is more applicable for this use-case as the top three model recommendations are shown to the support agents.

Architecture

This section presents the overall solution architecture in production. Due to business requirements, we deployed two instances of the models:

1. near real-time inference (for English)
2. batch inference (for English, Chinese, and Portuguese). The following figure illustrates the overall architecture for both model inference pipelines:

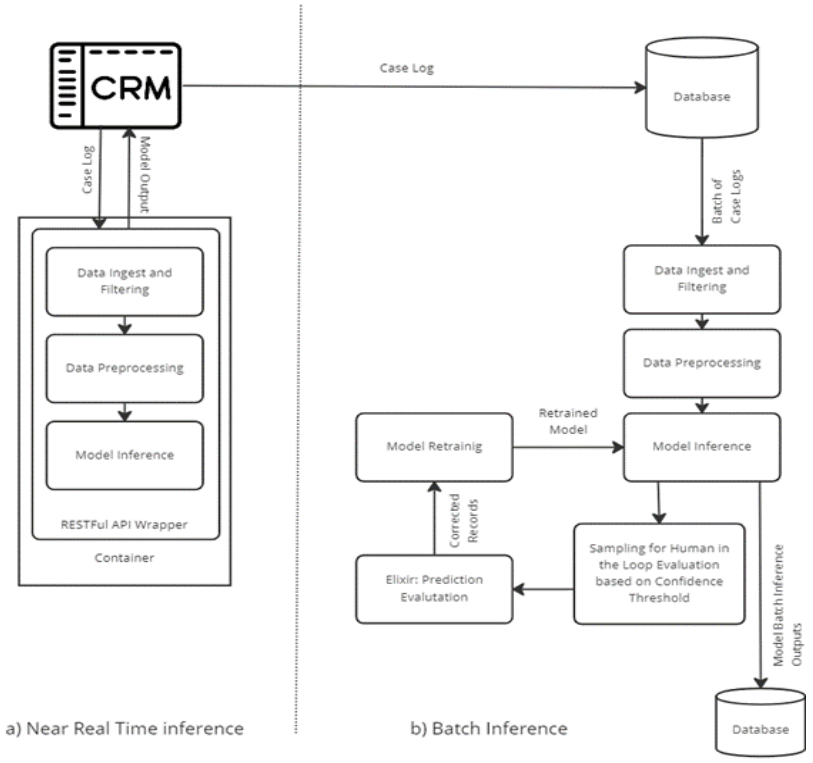

Figure 1. (a) Left: Near Real Time Prediction Pipeline (b) Right: Batch Prediction Pipeline

Batch Prediction Pipeline

The architecture includes several key stages:

- **Data Ingest and Filtering**—This step obtains the latest data at three hourly cadences. Next, it runs a language detection model on raw data and stores detected languages against records. Then, it performs data filtering operations including filtering out unsupported languages, records with missing text data, or records with only one or two words.
- **Data Preprocessing and Normalization**—This step minimizes noise for the model including removal of irrelevant tool-generated content and machine-generated prompts from chat records. Next, it expands commonly used acronyms and abbreviations by support agents with the aid of DellNLP. The final step involves lemmatizing text.
- **Model Inference**—This steps converts filtered, cleaned text into ML features and feeds it to the model to obtain the classification result. As discussed in Model Experimentation, character level TF-IDF data is used and the final model is an augmented Linear Support Vector Machine, which outputs model confidence for individual prediction. Final output is a T1-T2-T3 label, given a textual case log.

The preceding stages have been implemented as separate Python packages following single-responsibility principles. Development of independent packages enabled effective

management of updates, tracking of versions, improved code reuse, and ease of automation through Continuous Integration (CI/CD) pipelines.

Automated Model Retraining

One of the major concerns of using supervised ML for this classification task is that the model fails to effectively recognize and classify novel and unseen issues at the inference stage. To address this issue, a human-in-the-loop model feedback mechanism is implemented. The main idea is that a proportion of model predictions are validated daily by a group of experts. The experts manually analyze textual logs with associated model predictions and propose any corrections to labels, including the addition of new labels. Authors primarily focus on validating low-confidence predictions but used a staggered model confidence-based random sampling strategy for selecting predictions for validation. They allocated 70 percent of low confidence predictions based on a fixed low-confidence threshold and, 10% of high-confidence predictions based on a high-confidence threshold and finally 20% randomly sampled predictions for manual validation.

Elixir – A Novel Data Labelling and Prediction Validation Tool

Due to the large number of labels from which to choose, it is extremely difficult, even for experts, to choose an optimum label in certain situations. Therefore, we investigated open-source data labeling/validation tools to accelerate and facilitate the validation process. This analysis revealed that most existing solutions do not satisfy the requirements, in particular, support for a large number of labels (over 500), the ability to search and insert labels, and embedding of a model. Therefore, the authors implemented our own active learning-enabled data labeling and validation tool known as Elixir, also considering future data annotation/validation requirements in mind. The following figure shows Elixir's UI.Main User Interface of Elixir

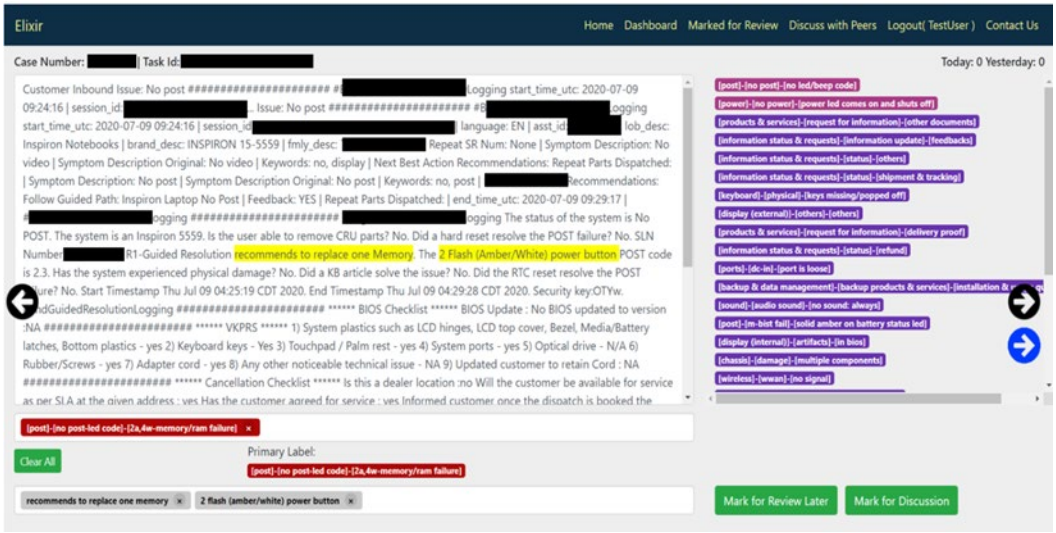

Figure 2. Main user interface of Elixir

Model prediction records require manual validations to be automatically distributed and allocated to a group of experts using Elixir. Case logs are shown in the main text area where the experts can highlight relevant keywords indicating the correct classification. Elixir uses the same SVC model and generates the top 20 predictions. These predictions are shown in the right side of the UI (purple labels in the preceding figure). The top three predictions are highlighted in red and pink. We noticed that 90% of the time, the accurate label can be found within these top 20 labels. Therefore, experts do not have to scroll through a large number of labels to find the most relevant classification. Experts can choose the best matching labels from the labels on the right side and move to the next case log. If a suitable label is not found in the right side, they can enter some keywords in the label area where a list of matching labels (out of entire list of labels) is shown. Experts can then choose the relevant label from the retrieved label. Finally, if a novel issue is noticed, experts can type the new issue classification.

The model predictions are consistently validated by experts and validated records are being used for incremental training of the model. This architecture ensures model accuracy keeps improving while learning new records. The following figure shows the implementation architecture for Elixir:

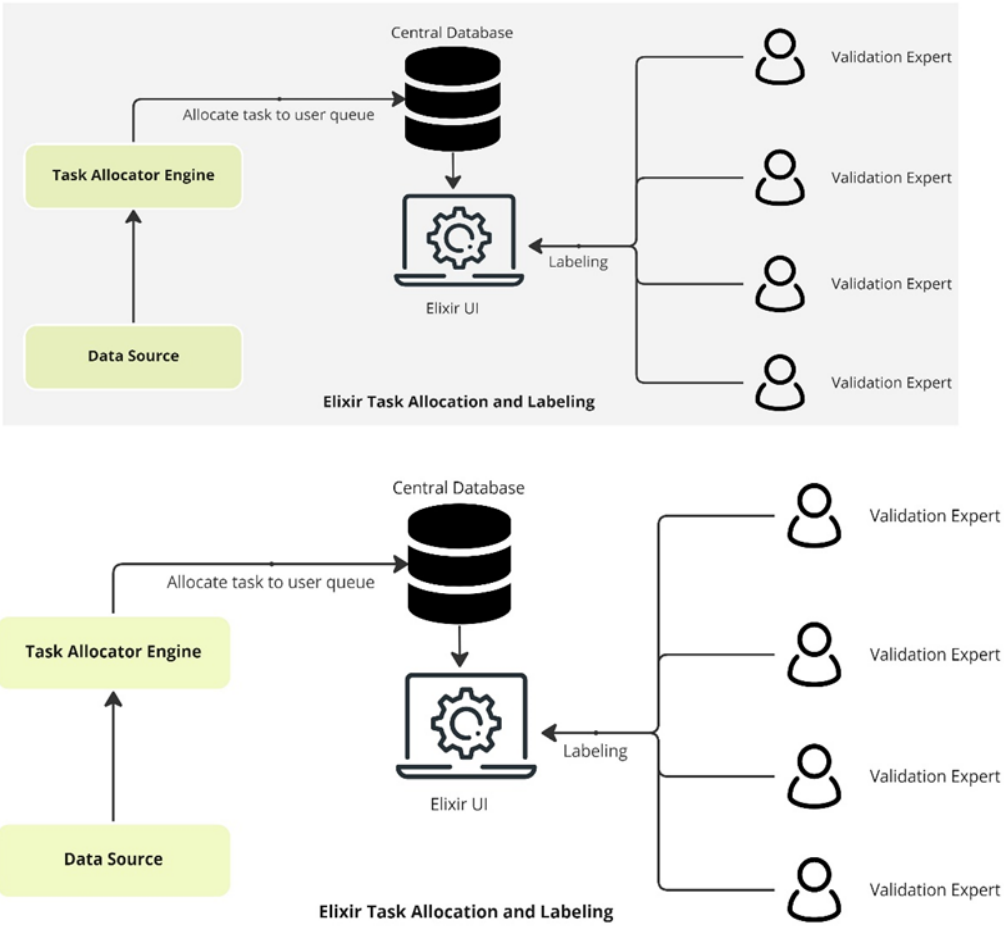

Figure 3. Implementation of Elixir for Data Labeling and Validation

The main components of Elixir include a task allocator engine, a central database, and the UI. Validation experts register on the tool by placing a registration request. As soon as the request is placed, an approval link is sent to the administrator through automated instant messaging notification. Once approved, the user is onboarded to the tool, followed by automated assignment of prediction validation tasks for the user by the task allocator engine. The engine runs at scheduled intervals, continuously keeping track of task queues for each user and allocates validation tasks from a common pool (data source) to each user queue assuring that each user has no more than 100 pending tasks in queue. Elixir also provides daily statistics to the user such as the total tasks allocated to them and number of tasks completed per day, week, and so on. These statistics help them track their validation progress. In addition, a weekly report containing a summary of all validation statistics by the entire team is generated and shared with the team leaders. Furthermore, additional useful features were implemented for validation experts: for example, the ability to mark certain difficult tasks for reviewing later. Such tasks will be moved to separate queues where subsequently users can mark them as reviewed once completed the validation. This feature was useful especially for newly registered experts to quickly complete easy validation tasks first but once they gain some experience with the tool and case logs, they can revisit more difficult tasks for validation. Another useful feature is the “mark to discuss” feature where users can mark certain tasks and access these tasks later in a separate dashboard to discuss them with peers and validate together.

Near Real-time Prediction Service

The objective of the model prediction service is to provide model predictions to the business CRM application on demand.

As the same model is used in the batch pipeline and for the service, it is necessary to focus on the reusability of the model. Independent from the service to which the model is imported, the model must be able to be ported to the service to ensure that the same version of the model is maintained for use-cases. A wheel file (.whl) has been created and stored in Dell's private Python Package Index (PYPI) repository to be an artifact for the model. This package could be installed in the service environment. During the build stage of the service, it draws that artifact and package it with the other dependencies.

To deliver the service, the service performance, compatibility, and portability has been considered. For performance, the service needs to respond to requests within seconds, even at times of increased load. To integrate with the business CRM system, the service must be available over a compatible interface. To future proof the service, this interface must be industry standard. And finally, the service must be portable to different infrastructures with minimal effort.

In reverse order, to develop a service that can be ported from one environment to another, the service was built into a Docker container. Container services are widely available within Dell Technologies and with third-party vendors. The container is currently deployed to a Pivotal Kubernetes Service (PKS), a container-based environment. To integrate with the business application, an HTTP RESTful interface is exposed, an industry standard protocol and design style, which abstracts from the implementation details of the model service. To develop a service that responds within latency requirements, the service start-up time delay by loading model implementations from disk had to be considered. Thus, by using Uvicorn, an Async web server, it was mitigated as much as possible. It is worth noting that, while most of the computation is CPU bound, in the filtering stage the compute may be Input/Output (IO) bound as it communicates with an external language detection service depending on the service configuration. To scale at peak load, Kubernetes and Gunicorn are used to perform with horizontal scaling.

For the model artifact creation and container build, it is mandatory to make sure the artifacts generated do not violate the security standards of Dell Technologies. So, the CI/CD pipeline that does the security checks for the model artifact and the service container is put in place.

The following figure shows the model service architecture:

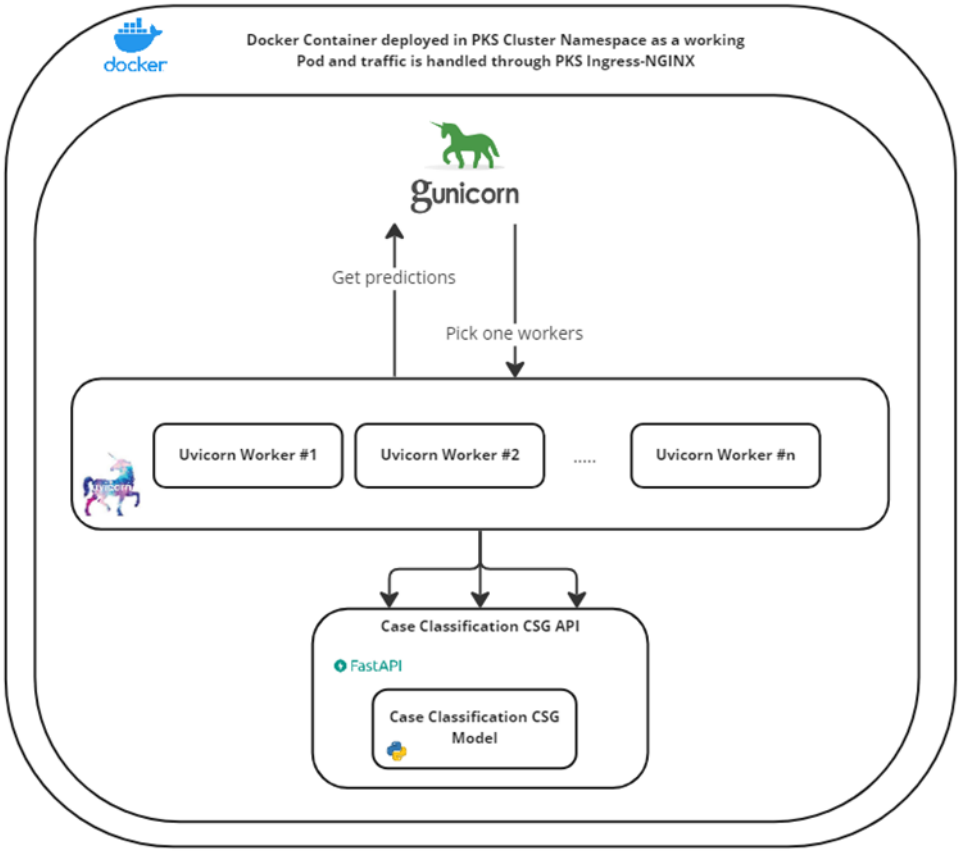

Figure 4. Model Service Architecture

The model service is called through the CRM UI by triggering the model by sending API requests that contain the textual case logs. Next, the steps a Batch Prediction Pipeline are run and classification is obtained. The classification will then be displayed through the CRM, where support agents may alter the classification to provide additional feedback for model retraining which is leveraged in combination of validation data obtained through Elixir.

Conclusions and Future Work

In this white paper, we described the implementation of a multiclass text classification model for classifying support services case logs into a predetermined set of issue categories. We discussed important design decisions behind deriving classification taxonomy, the methodology adopted for training-data curation, and model selection strategy. Key experiments carried out including different modeling techniques and feature representation techniques along with their results have also been presented. The deployment architecture of the model has been elaborated in detail. Model architecture includes the batch prediction architecture as well as the near real time prediction architecture with the details on different pipelines involved and model service architecture. It also includes the implementation of model feedback loop leveraging in-house developed prediction validation tool known as Elixir.

The Intelligent Case Classification (ICC) solution helps to understand the intent behind customer-initiated contacts and enables profiling of issues. ICC has already been proven to bring substantial business benefits. It has helped surface issue patterns related to repeat customer contacts and repeat dispatches of hardware components. It has been instrumental in identification of substantial repeat contacts reduction opportunities leading to large cost reductions. The results mentioned above are achieved by identifying technical issue patterns resulting from ineffective usage of tech support tools. Furthermore, ICC is also effective in identifying trending issue patterns. Standardization of issue descriptions has enabled interoperability between data analysis tools and reporting. Compared to a manual process, with the help of ICC, we have been able to scale the solution to all the regions for three languages with consistent results. ICC predictions are also being used as an input to other ML/AI systems.

Future work includes exploration of large, pretrained multilingual models for expansion to additional languages. Experimentation with multilingual feature representation techniques, such as multilingual embeddings, is already underway to streamline models. The aim is to train a single model that can accept case logs of a given set of languages as the input and predict the T1-T2-T3 issue category as the output. Furthermore, unsupervised methodologies are being explored to address the limitations imposed by static nature of labels.

Other work focuses on creating a fully automated model retraining pipeline with modularized components for data ingestion, data preprocessing, feature engineering, model training with hyperparameter tuning, model inference, and model monitoring.

Documentation

The following external documentation provides additional information:

- Wasala, A., Garg, M., Shaw, P., Arora, A., Singh, S., & Punjj, R. (2021). Dell Technologies DellNLP: A Centralized Text Processing Engine | Dell Technologies Info Hub. https://infohub.delltechnologies.com/t/dell-technologies-dellnlp-a-centralized-text-processing-engine/
- Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. Advances in neural information processing systems, 32.
- Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., ... & Bowman, S. (2019). Superglue: A stickier benchmark for general-purpose language understanding systems. Advances in neural information processing systems, 32.
- Fielding, R. T., & Taylor, R. N. (2002). Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT), 2(2), 115-150.
- Sun, A., Lim, E. P., & Liu, Y. (2009). On strategies for imbalanced text classification using SVM: A comparative study. Decision Support Systems, 48(1), 191-201.
- Liqun Liu, Funan Mu, Pengyu Li, Xin Mu, Jing Tang, Xingsheng Ai, Ran Fu, Lifeng Wang, and Xing Zhou. 2019. NeuralClassifier: An Open-source Neural Hierarchical Multi-label Text Classification Toolkit. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 87–92, Florence, Italy. Association for Computational Linguistics.
- Matthias Damaschk, Tillmann Dönicke, and Florian Lux. 2019. Multiclass Text Classification on Unbalanced, Sparse and Noisy Data. In Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing, pages 58–65, Turku, Finland. Linköping University Electronic Press.