Amazon Intent Natural Language Classification

Muaaz Noor mnoor@upenn.edu
Kevin Xie kevinhx@upenn.edu
Lance Lepelstat lancelep@upenn.edu
Adiwid Devahastin adiwid@upenn.edu

Abstract

Our goal is to build a multi-classification NLP model that ingests input from a user and outputs a probabilistic prediction of each of the class labels that message/intention falls into. Our literature review surveyed various approaches from traditional Naive Bayes Classifiers, Support Vector Machines, and LSTM models to more sophisticated hierarchical models. Our extensions were based on such hierarchical approaches: the first extension was a hierarchical modeling tree, while our second extension created a hierarchy by combining target labels into parent classes to create a parent-child labels and made predictions using an LSTM. Our Hierarchical Modeling Tree had an accuracy of 0.8551 and F1 score of 0.8634, which underperformed our Parent-Child Hierarchical LSTM model which had a best accuracy of 0.8697 and F1 score of 0.8681 on the dev set after hyperparameter tuning. On the test set, we saw testing set accuracy scores of 0.8439 and 0.8524 respectively and F1 scores of 0.8672 and 0.8503 respectively, exceeding strong baseline performance slightly.

1 Introduction

Classifying user intent has multitudinous applications for problems across industries. For example, in the consumer banking industry, automated chatbots take in queries from a user and return a number of suggested self-help links via the classification of their issues based on certain phrases in the input. In the airline industry, customers call to inquire about flight information and expect to be directed to the right source to execute their demands. More broadly though, virtual assistants such as Amazon Alexa are tasked with parsing language to then classify intentions in order to elicit the appropriate response. The Amazon Massive Intent Dataset provides a rich source of data to train an NLP model to accomplish this.

The below table illustrates this task using examples from the dataset. We observe natural language sequences (inputs) paired with intent class labels (outputs) drawn from a unique set. In the first example, the user's request is paired with an intent class related to setting the alarm. It is clear how the contents of the sequences can be systematically processed to determine the correct classifications.

label	text	label_text
48	wake me up at nine am on friday	alarm_set
46	olly pause for ten seconds	audio_volume_mute

Table 1: Text and Intent Classes

For this project, we define our problem as follows: create a NLP model that can correctly classify the intent of as many sequences from our data as possible.

Using linguistic theory and advances in computation, scientists have developed many successful techniques to draw insights from natural language toward the goal of intent classification. We chose this project to build upon these advances and to experiment with the tools we have developed in this course by expanding beyond the tasks we have previously undertaken. Our work with sentiment classification in this course provided an essential foundation, but we were excited to challenge ourselves by developing a model that can predict from a larger number of classes. We also felt motivated by this problem due to its importance in industry.

By undertaking our own experiment in intent classification, we can better understand the challenges and successes in this domain that have shaped industry in recent years.

2 Literature Review

2.1 Paper 1: "Intent Classification for Dialogue Utterances"

In this paper, Frasincar and Schuurmans (Frasincar & Schuurmans, 2019) compare different models to classify intents on dialogue utterances. Furthermore, the paper proposes two different approaches for each model: (i) a flat model, which performs classification using a predefined set of intents, and a (ii) hierarchical approach that classifies intents into 'main' categories, forming an intent tree, and then uses a local classifier at each tree node. To start off, they deploy a baseline model which uses a Bag of Words (BoW) along with a Naive Bayes Classifier. This model gave a better performance with the hierarchical approach, getting an F1 score of 0.614 compared to 0.541 for the flat model. The second model uses an SVM classifier which is paired with three different types of embeddings (FastText, GloVe, Word2Vec). The embeddings are then (i) summed up and (ii) averaged for each utterance sequence. The results show that the best flat SVM classifier gave an F1 score of 0.752, while the best hierarchical SVM classifier had an F1 score of 0.782. It was also noted that taking the average of the embeddings performed better than taking the sum. The last model that the authors tried was a recurrent neural network, specifically an (i) LSTM and a (ii) bi-directional LSTM. The best model amongst these was the uni-directional LSTM which had an F1 score of 0.605, concluding that the SVM models outperformed the deep LSTM models on the given dataset. Moreover, the paper also introduced the novel approach to implement a hierarchical classification tree described above, which was shown to have increased performance in the results.

2.2 Paper 2: "Intent Classification in Question-Answering Using LSTM Architectures"

In "Intent Classification in Question-Answering Using LSTM Architectures", Di Gennaro et al. (2021) implement an NLP model to perform intent multi-classification on a dataset of 6,000 English questions. These questions are labeled using a two-level class hierarchy. The first level consists of six broad categories of intent: "abbreviation, entity, description, human, location, numeric." The second level contains multiple sub-categories for each of these six. For example, the location category has subcategories including "city" and "country." The questions are preprocessed by removing special characters and punctuation and converting to lowercase. The authors use GloVe embeddings with dimension of 300 for the words in these questions. The authors' first model classifies using only the first level of the class hierarchy. For each question, the embedded words are passed as input into an LSTM model. The final hidden state is passed through a network with Softmax activation to predict one of the six higher level classes. This model is trained by backpropagation through time using the categorical cross-entropy loss criterion. Using the optimal hidden state dimension of 75, this model achieves 90.60% accuracy on the test data. The second model classifies using both levels of the class hierarchy. A blank pad is added to the end of each question. The LSTM architecture from the first model is applied, with the hidden state from the final word in the question passed on to the LSTM cell for the blank pad to predict the second class level. This allows information from the first class level prediction to influence prediction for the second class level. Two separate networks with Softmax activation predict the two class levels. Backpropagation through time is used again, but with two separate categorical cross-entropy loss functions for the two goals. Using different hidden state dimensions, this model achieves around 90% accuracy for the first class level and around 80% accuracy for the second class level using test data.

2.3 Paper 3: "Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling"

Liu and Lane's 2016 paper titled "Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling" essentially extends the attention-based encoder-decoder RNN model. This paper tackles our problem with a focus on slot filling, which refers to the identification of particular slots from a running dialog that correspond to parameters of a user's question. Since our focus is on intent classification though, we can adopt the joint architecture of their proposed model to suit our purposes. The bidirectional RNN ingests the source text from both directions. At each step t, they predict the concatenated backward and forward hidden states.The pre-computed hidden states produce the intent class probability distribution. When attention is not used, mean pooling across time steps then logistic regression is used to get the intent classification. However, with attention, the weighted average over time of the hidden states is used. The results confirm the attention-based model is more computationally efficient and does better. This ended up with a 95.75% F1 score on the ATIS airport dataset.

3 Experimental Design

3.1 Data

We have selected the Amazon Massive Intent Dataset, which provides a rich source of data in JSON format to train an NLP model to accomplish our intent classification goal. The format of the data can be found in Table 1 in our introduction. We aim to do a 70/12/18 train-dev-test split, which will result in 11,514 training samples, 2,033 development samples, and 2,974 testing samples.

3.2 Evaluation Script

For this project, we elect to use total classification accuracy as our evaluation metric. Because we are attempting to implement models that correctly classify input sequences into multiple intent classification labels, we require a metric that reports our overall performance toward this goal.

This metric allows us to measure the proportion of inputs which our models classify correctly across all classes. It is calculated using the simple formula below:

$$\textbf{Total Classification Accuracy} = \sum_{i=1}^{n} \frac{1\{y_i = \hat{y}_i\}}{n}$$

This metric has been successfully used in a wide variety of multiclass classification studies, including the study in the Di Gennaro paper, which performs multiclass classification of intent using LSTM architectures.

We also use the F1 metric as an alternative evaluation metric, which is the harmonic average between precision and recall.

3.3 Simple Baseline

We attempt to construct two such simple baselines: the majority class baseline and a k-Nearest Neighbors (kNN) classifier baseline.

The majority class baseline simply predicts the label that shows up most frequently in the training set for all examples. In the training set, the majority class merely makes up 7.035% of the dataset. Using this majority prediction, we have a training accuracy of 7.035% and testing accuracy of 7.028%, which are incredibly low but consistent hence underfit. This would be far too trivial to beat.

As a result, we try to explore a better but still naive baseline by running a Nearest Neighbors Classifier with k=1 that uses Glove embeddings to compute the embedding of each word in an example text, then averages them to get a sentence level embedding for the example. We do this for all examples in both train and test sets. Then, for each testing example, we compute the cosine similarity of its sentence level embedding with the embedding of each of the training examples. The label of this testing example is then taken to be the same label as the training example with the highest cosine similarity. This baseline produces a testing accuracy of 74.65%, which is somewhat more realistic as a simple baseline to beat.

4 Experimental Results

4.1 Published/Strong Baseline

The strong baseline model is based on the Generro et al. paper that takes in tokenized user utterances, gets their Glove embeddings, passes them through a bidirectional LSTM and fully connected layers, and outputs the predictions of intent.

We have the following LSTM equations:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

where:

- h_t = hidden state at time t
- c_t = cell state at time t
- x_t = input at timet t

- h_{t-1} = hidden state at time $t-1$
- i_t = input gate
- f_t = forget gate
- g_t = cell gate
- o_t = output gate
- $\sigma(\dots)$ = sigmoid activation function
- $\odot$ = Hadamard (element-wise) product

We use a batch size of 128, a learning rate of 0.01, and train for 10 epochs. The strong baseline LSTM model produces a testing accuracy of around 85.5%, which is already decent but still gives us some room for improvement by doing extensions.

4.2 Extension 1: Parent-Child Prediction using LSTM

We attempted to improve upon our strong baseline by implementing an extended LSTM model. Because the individual labels in our data contain prefixes that group them into parent classes, it seemed possible that our model could use these parent classes to learn more during its training process and thus increase performance.

Similar to the strong baseline model, the extended LSTM model takes in tokenized user utterances, gets their Glove embeddings, passes them through a bidirectional LSTM and fully connected layers, and outputs the predictions of intent.

The main difference between the strong baseline model and extended model is the way that loss is calculated. Instead of measuring loss solely based on the given labels, we place those that are related to each other into the same category and give them a parent label. To avoid confusion, we call the given labels “child labels” and the created labels “parent labels”. Since we are required to predict both parent and child labels, there are some minor changes to the baseline architecture: the input to the extended LSTM model is padded with an additional <PAD> token at the end of all text samples.

The output of the actual last token of the sentences produces the parent class prediction. The output of the <PAD> token produces the child class prediction, which is the one used to compute classification accuracy. The loss of the extended model becomes the sum of the losses pertaining to the parent class prediction and child class prediction.

Figure 1 shows a diagram of the neural network architecture of our extended model.

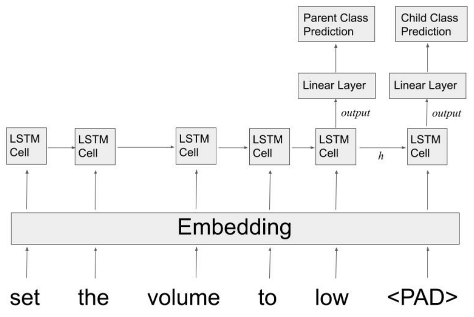

Figure 1: Extension 1 Model Architecture

We experimented with hyperparameter tuning for this extended model by comparing different combinations of learning rate and batch size. The below table shows our accuracy and F1 score for each combination using the development data.

Learn Rate	Batch Size	Accuracy	F1
0.001	64	0.8593	0.8587
0.001	128	0.8549	0.8534
0.001	256	0.8529	0.8516
0.01	64	0.8633	0.8632
0.01	128	0.8697	0.8681
0.01	256	0.8687	0.8693
0.1	64	0.7290	0.7241
0.1	128	0.7570	0.7511
0.1	256	0.7752	0.7731

Table 2: Hyperparameter Tuning for Extension 1

We found that using a learning rate of 0.01 and a batch size of 128 yielded the best accuracy of 0.8697. The F1 score of the model with these hyperparameters is 0.8681. We found that there are not major issues with class imbalance for these data and that changes in our F1 scores are comparable to changes in our accuracy scores.

4.3 Extension 2: Hierarchical Modeling Tree

Looking at the performance of extension 1, we decided to exploit the hierarchical structure of our data through a more complex modeling approach.

Similar to extension 1, we classified our labels into 18 “parent classes”.

For the purpose of modeling, we devised a modeling tree, where we have a multi-class classification model at the root node, that classifies each sequence into one of the 18 parent classes. Depending on the classification of this model, we traverse down to the respective child node, which uses a different model trained specifically on data belonging to that parent class. The intuition behind such a structure was that each child model would be well-trained and better suited to preform predictions for similar data belonging to one class.

All of the models in the root node and the child node use the same LSTM structure that was specified in the published baseline. Input tokens are passed through an embedding layer that uses Glove embeddings. These are then fed into the LSTM layer, the output of which is passed through a linear layer which outputs one of the classes.

The structure of the model can be visualized in the figure shown below.

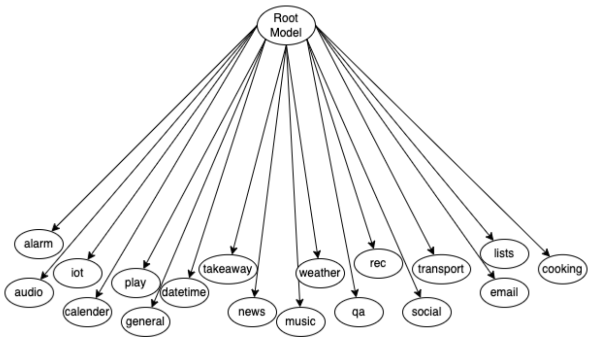

Figure 2: Extension 2 Modeling Tree Structure

As we can see from the figure, our tree modeling structure has a root model at the top that classifies into one of the 18 parent classes, each of which have a model of their own.

We tested out our model on the development data using different combinations of hyperparameters: (i) Learning rate was changed uniformly across all models in the tree, while (ii) different combinations of batch sizes were used for the root model and the child models. The results of hyperparameter tuning are summarized in the table below.

LR	Root BS	Child BS	Accuracy
0.001	32	64	0.8351
0.001	32	128	0.8493
0.001	64	64	0.8401
0.001	64	128	0.8509
0.01	32	64	0.8379
0.01	32	128	0.8521
0.01	64	64	0.8431
0.01	64	128	0.8551

Table 3: Hyperparameter Tuning for Extension 2

After hyperparameter tuning, the best model achieved an accuracy of 0.8551 using a root batch size of 128, a child batch size of 64 and a learning rate of 0.01. The F1 score for this combination of hyperparameters was 0.8634. Contrary to our hypothesis, our extension 2 was not able to outperform extension 1 in the best case scenario when looking at the overall accuracy. However, the F1 score, while on par with that of extension 1, is relatively higher than the accuracy achieved by this model. This shows that this architecture was able to better counter the class imbalance within the dataset.

4.4 Final Results

We finally test out our three models on the test data. The results are summarized in the table below for each of our model.

Model	Accuracy	F1 Score
Strong Baseline	0.8433	0.8412
Extension 1	0.8524	0.8503
Extension 2	0.8439	0.8672

Table 4: Comparison of Final Models

We find that our extension 1 model demonstrates the best accuracy of 0.8524. However, in terms of F1 score, extension 2 outperformed all models achieving an F1 score of 0.8672.

5 Error Analysis

The two `text_label` classes that had the highest proportion of misclassifications were `general_quirky` (label 12) and `qa_factoid` (label 49). The former represents random requests while the latter represents requests for factual information.

Within these top 2 label misclassifications, we deduced 3 major categories of errors:

1. Slang and Faux Word Confusion
2. Oversensitivity/Overfitting to Food-Themed Words
3. Short and Vague Commands

Examples of Category 1 include phrases like "`wakey wakey eggs and bakey`", where the only word that really made sense dictionary-wise was "eggs". The model was unable to understand the terms "wakey" and "bakey" which were really meant to be clear signals for quirky requests, but our model focused on "eggs" as the only word that made sense and thought it was a recipe request and so predicted label 9 `cooking_recipe`. This is also an example of Category 2 errors.

A more prominent example of Category 2 error is the phrase "`are jello shots calorie free`" which was a clear question with a fixed yes/no factual answer. However, our model honed in on "jello shots" and thought the food term warranted a recipe request and likewise predicted label 9 as well.

As for Category 3, there were many examples of this where requests were far too terse that there just wasn't enough information or context for our model to go off of. For example, there were requests that were solely "`none`" or "`the cosmos`" or "`sports`" which were so vague that our model ended up honing into the only nouns that were available to make the closest categorical prediction.

6 Conclusions

Our published baseline demonstrated the use of a simple LSTM structure for the purpose of multi-class classification. Our dataset had 60 class labels, and the published baseline was able to achieve an accuracy of 84.33% on the test set.

For our extensions, we decided to exploit the hierarchical nature of our dataset through two modeling approaches: (i) an LSTM model that predicts the parent class along with the child class, and (ii) a modeling tree architecture that uses the parent class predictions to decide on which pre-trained child class model to use. The first extension was able to achieve an accuracy of 85.24% on the test set, which outperformed the published baseline. Moreover, the second extension was able to achieve an accuracy of 0.8439, which was not able to improve on the results of extension 1. However, the F1 score for this model was 0.8672, which is higher than the F1 score of 0.8503 achieved by extension 1.

Extension 1 outperformed all of our models and gave the best results in our experiments. Looking at the mistakes this model made, we were able to identify three error classes: (i) Slang and Faux words, (ii) Oversensitivity/overfitting to food-themed words, and (iii) Short and vague commands.

For Extension 2, we dived deeper into understanding why it was not able to meet expectations. While the extension was still able to beat the published baseline, one of the reasons for it to not meet expectations could be the significantly larger number of hyperparamters used. For the purpose of this project, we used uniform hyperparameters across each child model. However, each child model could be further optimized to get the best hyperparameters. Moreover, diving deeper into each model showed that the root model was able to achieve around 90% accuracy, which means that 10% of the data points were not being classified by the correct child model, which could explain the lower accuracy.

Acknowledgments

This document has been adapted with the guidance from Yifei Li and Professor Mark Yatskar. This project would not have been possible without the weekly check-ins with Yifei to update him on our progress and our consultations with Professor Yatskar. We had thoroughly enjoyed this challenging project and the chance to apply what we've learnt in class to a relevant industry problem.

Bibliography

Schuurmans, Jetze & Frasincar, Flavius. (2019). Intent Classification for Dialogue Utterances. IEEE Intelligent Systems. PP. 1-1. 10.1109/MIS.2019.2954966.

Di Gennaro, G., Buonanno, A., Di Girolamo, A., Ospedale, A., Palmieri, F.A.N. (2021). Intent Classification in Question-Answering Using LSTM

Architectures. In: Esposito, A., Faundez-Zanuy, M., Morabito, F., Pasero, E. (eds) Progresses in Artificial Intelligence and Neural Systems. Smart Innovation, Systems and Technologies, vol 184. Springer, Singapore.

Liu, B., Lane, I. R. (2016). Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling. CoRR. https://doi.org/abs/1609.01454