Size Optimization for Intent Analysis in Voice Commanding

Parth Milind Shettiwar
parthisultimate@gmail.com
Indian Institute of Technology
Bombay, India

Koushiki Chaudhuri
koushikidasguptachaudhuri@gmail.com
Indian Institute of Technology
Kharagpur, India

Ankit Jain
Ankit.1573@gmail.com
Microsoft India

Shivam Goel
goel.shivam117@gmail.com
Thapar University, India

Abhirupa Mitra
mitra.abhirupa@gmail.com
Vellore Institute of Technology, India

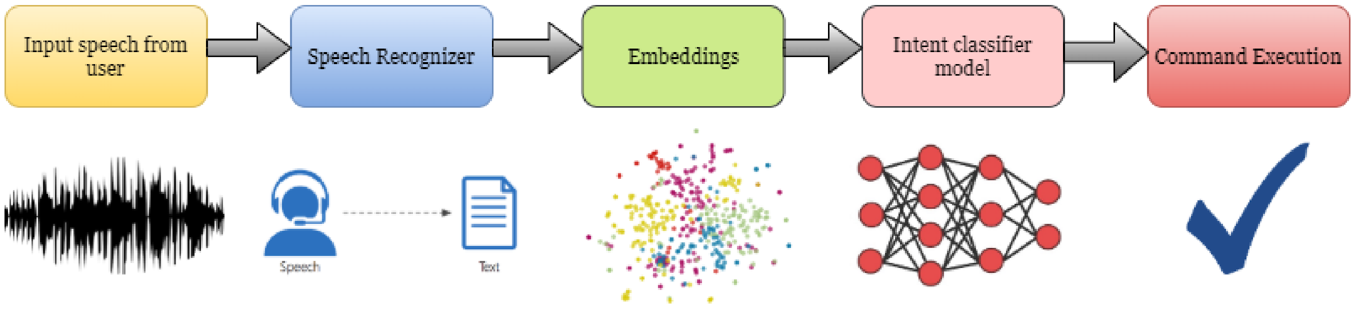

Figure 1: Overall Pipeline of Voice commanding

ABSTRACT

Voice technology is getting more and more ubiquitous and with that, the need for it to work always. In this technology, a major highlight is to incorporate voice commanding in an application on a device. This suffers inaccuracies at two levels, firstly the conversion of the user's speech command to text and secondly classification of the underlying intent from this text. This two-fold process is presently either done completely On-server limiting the functionality to only when user is online or incorporates two Deep Learning models leaving behind huge memory footprints in the app. In this work, we show how this problem can be solved by training only a single size optimized Intent classifier covering up the inaccuracies incurred by the device offline speech recognizer. A quantitative size comparison is first shown between various existing Machine Learning and Deep Learning Intent Classifier Models followed by an end to end implementation of this in Microsoft Word App.

CCS CONCEPTS

• **Natural Language Processing → Intent Classification**.

KEYWORDS

Natural Language Processing, Embeddings, Data Augmentation, Model Size

ACM Reference Format:
Parth Milind Shettiwar, Koushiki Chaudhuri, Ankit Jain, Shivam Goel, and Abhirupa Mitra. 2020. Size Optimization for Intent Analysis in Voice Commanding. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As we move towards an era of automation, Voice Technology is evolving at a rapid rate. Voice commanding is a prevalent feature nowadays supported by various virtual assistants like Siri, Alexa, Cortana, and Google Assistant. This feature is an intricate process where first the user's speech is converted into text and then the text is understood as one of the supported commands by the application followed by its execution .

The voice commanding is presently done using an On-server Model which does both speech recognition and intent classification and sends the result back to the application which then executes the task accordingly. This approach gives very accurate predictions. However the method is limited to only when user is online putting the user at backfoot when he loses Internet connectivity abruptly. Further, there is a direct cost associated with an all time server running. A second approach, as adopted by [1], where we train our own speech recognizer and language understanding model and deploy them in the app. This gives great results and eludes the issues faced by first approach. However this approach would leave a massive memory footprint, on account of two Learning Models, in the apps adding to their already huge sizes, discouraging users to download them.

In this work we show, how in a proper setting, this problem can be solved by evading both of the above issues and execute commands with promising accuracy. We exploit the already deployed Offline Speech recognizers on phone devices and use their inaccurate text conversions in our size optimized Intent Classifier Model to accurately recognize the command. Our key contributions from this work are:

(1) Leveraging the existing offline speech recognizer on the device and augmenting the available dataset with its inaccurate text conversions.
(2) A comprehensive size comparison among existing Intent Classification Models and their size optimization.
(3) End to end implementation of the our proposed pipeline in Microsoft Word App with a minimal size incurrence.

2 METHOD AND IMPLEMENTATION

Our approach is formulated in the following manner:

- We first use an online available commands voice dataset of a particular accent and get the corresponding text transcriptions after passing through the On-device offline speech recognizer.
- This is followed by training our size optimized Intent classifier model on the dataset formed from above step.

We discuss both of the above steps in detail in the following subsections. The following discussion will be in reference to a specific voice commanding task of formatting a Word document, having Intents like Bold, Italics, Underline etc and its subsequent implementation.

2.1 Transcriptions from Offline Speech recognizer

It is observed that speech recognizers on phone devices give quite inaccurate transcriptions when the user is offline. This discourages their usage in preparing datasets for Natural Language Processing tasks like Machine Translation, Question Answering or Summarization. However for the current scenario of Intent Classification task, it is crucial to observe that we won't have very big sentence commands. A command length can range from a word to maximum 9-10 words. This limits the overall variations that our offline Speech recognizer can produce for a command. For example, Bold the text is transcribed as "old the text", "board the tex" and a few more deviations are only possible. Leveraging this fact, we pass an available dataset of voice commands through our phone's offline speech recogniser and automate the whole process. We add these transcriptions to our Intent Classification Model dataset.

2.2 Size and Accuracy Analysis of Intent Classifier Models

In this section we explore the existing Intent Classification Deep Learning and Machine learning models and compare their sizes and accuracies on the dataset formed after Section 2.1. The dataset had 26 labels, each corresponding to a formatting option in a document. A 9:1 split is formed for training-testing and models are run for 300 epochs. As shown in 1, we have implemented 7 models and reported their sizes and accuracies. The transformer based model was a single transformer, as described in [2], of encoder decoder architecture with decoder modified with a softmax layer to give probabilities of all labels. The CNN based model had an embedding layer followed by 2 conv layers, followed by 2 dense layers. Similarly Bi-Directional LSTM was implemented as, an embedding layer, followed by Bi-LSTM layer and then a dense layer. The Average Word2Vec model was implemented as shown in Figure 2

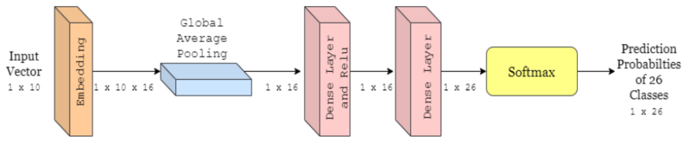

Figure 2: Average Word2Vec Model

	Model Size	Accuracy
Transformer based(p)	451.9 kb	88.4%
CNN based(k)	244 kb	89.9%
Random Forest(s)	595 kb	88.5%
Naive Bayes(s)	44 kb	81.2 %
Bi-Directional LSTM(k)	7.2 mb	96 %
Linear Regression(s)	**22 kb**	**95.2** %
Average Word2Vec(k)	**79 kb**	**97.6** %

Table 1: Quantitative Size and Accuracy comparison between various Intent Classification models. ***Model implementations: (p) - pytorch, (k) - keras, (s) - sklearn***

Further, we implemented 3 Machine Learning Models namely Random Forest, Naive Bayes, and Linear Regression. We conclude that Word2Vec and Linear Regression Models gave a promising accuracy and had minimal model sizes which can be easily added to app sizes.

2.3 Size Reduction and Implementation in Word App

The deployment of the above models in an App have to be done after converting the above models to tflite (for Android Devices). However, during inference, the tflite needs an extra dependency which is itself 3.6 mb in size. The following 2 approaches are taken for implementation of each model in Word App:

2.3.1 *Linear Regression.*

The Linear Regression model is an effectively Matrix multiplication operation. $Wx + b = Conf$ where W, b are weights and biases, learnt from training. x is the input vector and $Conf$ is array of confidence values for each label. Leveraging this fact, we directly imported the learnt parameters W, b into the app, and performed the matrix multiplication, avoiding import of any other external dependency during inference in the app. Effectively, the overall app size increment reduced to only **12 kb** since all parameters are written explicitly in code-base contrast to importing the model (22 kb) inside the app.

2.3.2 Average Word2Vec. Unlike Linear Regression, as this is a Deep Learning Model, the weights can't be explicitly written inside the code-base(more than 3k parameters!). Alternate approach was taken where we strip the tflite binary which is imported in the app. We develop a custom binary supporting only Relu, Softmax, Fully Connected layer, Mean, Gather and Cast operations, required by our model, bringing down the overall size to 268.9 kb for armv7-a architecture (covers all Android Devices). Further the model was converted to tflite extension bringing its size down to 12 kb, leading to overall **280.9 kb** size increment of app.

3 CONCLUSION AND FUTURE WORK

We keep both the models open for future work in voice commanding where Linear Regression enjoys a size of mere 12kb however not scalable to complex commanding. Similarly Average Word2Vec model is easily scalable, however imports a size of about 280kb in the apps. Both enjoy a promising accuracy as seen.
The above pipeline was successfully adopted in Microsoft Word App and can be extended to other apps like Outlook, Excel etc.

REFERENCES

[1] Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces. *ArXiv* abs/1805.10190 (2018).

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. *ArXiv* abs/1706.03762 (2017).