Chapter 2

Methodology for Code Completion in Visual Codes for Scratch3 files

The methodology is described in four steps for next token prediction for Scratch3 [1] Visual codes files.

- Extracting Scratch3 [1] revisions file paths
- Splitting training and testing dataset
- Training dataset on Kenlm [10], nltk [7] and bilstm [0]
- Evaluation of model

2.1 Extracting Scratch3 Revisions file paths

The Scratch3 [1] revision files path was generated from the Abstract Syntax tree [2] which is the intermediary representation for each file revision saved in our database. The database content table contains the hashes; an md5 representation of the parsed revision file along with the original contents saved as JSON [3] string value. We utilized the hashes to retrieve its matching contents; a JSON [3] string value deserialized to a dictionary to access the connections of each specific Scratch3 [1] revision file. This connection is a nested list with each element list containing a connection from the root node to every node in the tree. This was recursively traversed from the root node to every node in the tree to form all the connections in order. For instance, if we have a tree of an Scratch3 [1] file shown below.

Listing 2.1: Abstract Syntax Tree of a Simple Scratch3 File

```
[
        "simple_test",
        [
            [
                "event_whenflagclicked",
                [
                    [
                        "motion_movesteps",
                        [
                            [
                                "STEPS",
                                [
                                    "10"
                                ]
                            ]
                        ]
                    ]
                ]
            ]
        ]
    ]
```

We recursively visit every node from the root to each node in the tree to form a list of connections so as to form an order of nodes suitable for creating an ngram [5] model for training.

Listing 2.2: Connections from a Simple Scratch3 File

```
[
            [
                "simple_test",
                "event_whenflagclicked"
            ],
            [
                "simple_test",
                "event_whenflagclicked",
                "motion_movesteps"
            ],
            [
                "simple_test",
                "event_whenflagclicked",
                "motion_movesteps",
                "STEPS"
            ],
            [
                "simple_test",
                "event_whenflagclicked",
                "motion_movesteps",
                "STEPS",
                "10"
            ]
        ]
```

The extracted paths were further preprocessed by striping the first element of each connection which is the title of the Scratch3 [1] file so as to keep the connections as only visual code nodes. The next step involved normalizing the connections by replacing the non visual codes such as regular values like Hello, 10 and other values that are specifically entered by the user and replacing them with ⟨S⟩. This was finally extracted into a different txt file for each Scratch3 [1] revision file.

2.2 Splitting Training and Testing Dataset

The previous steps led to the extractions of 20699 paths. We utilized the scikitlearn [4] train_test_split method to split the dataset into ninety percent training set and ten percent train set. This was necessary to ensure that testing is done on a new dataset it hasnt seen before. The training data was padded with ⟨s⟩token corresponding to the ngram [5] order employed for training. For instance if we are to train with bigrams we padded the paths with two ⟨s⟩tokens at the start and end to ensure consistency.

2.2.1 Training Dataset on NLTK Language

We applied tokenization on the training set using the word_tokenize[6] method from the nltk [7] module to enhance analysis and training. In order to account for edge cases where preceding tokens might be insufficient for context, we employed the nltk padded_everygram_pipeline[8] to pad the dataset matching the order of ngram[5] we intend to train.The result of this along with the training data is passed onto the MLE [9] fit method to determine the maximum likelihood estimates of next visual code token. The model was saved in a pickle file to be used for evaluation as would be described in the evaluation steps.

2.2.2 Training Dataset on Kenlm

I utilized the kenlm[10] repository to estimate the language model on our training dataset. The approach as described in the official documentation was adopted for training and responses saved in an arpa [11] file. In more details, the Scratch3 [1] corpus which is sent through the stdin to the modified KneserNey smoothing [12] for estimation. Additionally, parameters such as the ngram [5] order and discount fallback were parsed for the estimation. This is sent to an arpa [11] file used for evaluation.

2.2.3 Training Dataset on Bilstm

Similar to the nltk[7] library, tokenization was applied to split the corpus into smaller words. Furthermore, we created a vocabulary from the token where a unique index is assigned to each token in the corpus. We applied padding on these tokens using the tensorflow pad_sequence [13] function to enable consistency with the number_of_steps length. In essence we made it match the sequence with the maximum length in our corpus. Subsequently we prepared features and labels for the tokens and applied one hot encoding on the last token in the sequence. Finally our data is ready for training. We applied some of these hyper parameters for our training as displayed in the table below and utilized the TensorFlow[14] fit method for training our dataset for 50 epochs.

Table 2.1: Hyper parameters used for Training on bilstm

Hyper parameters	**Values**
Embedding	100
Learning Rate	0.01
Epochs	50
Activation Type	softmax
Optimizer	adam
LSTM	150

2.3 Evaluation of Models

The evaluation was carried out on the various models to ascertain how accurately it could predict the next visual code token. The procedure adopted for evaluating the nltk[7] model is akin to the kenlm [10] model. In general the evaluation was done using a different dataset the model hasnt seen before to ensure an efficient model. The steps for evaluating each model are described in details below.

2.3.1 Evaluation of NLTK Model

The evaluation was carried out on the test dataset. We employed the context as all preceding tokens excluding the last token. The last token was the ground truth token adopted as the accurate predicted token. The nltk score [7] method was adopted to estimate the log likelihood score of each token sequence from the trained data alongside the context from the test data. The max score was selected and compared with the ground truth predicted token in our case the last token. This was computed and results were compared for accuracy, precision, recall and f1 score of the model. Fig 1 shows the ngram [5] order 2 to 10 with the highest accuracy seen in the 10-gram with an accuracy of 0.245. After 10gram order the model maintained similar accuracy as no significant increase in accuracy was observed. The figure below shows the graph of ngram [5] order on the horizontal axis and the scores on the vertical axis. The legend also contains the precision, recall and f1 score of the model during evaluation with a similar pattern seen as well. The highest precision and f1 score was seen in the 10th gram with a score of 0.226 and 0.2176 respectively. We also evaluated the model for ngrams order 10 to 15 as shown in figure 2 with results showing no significant increase in any of the metrics; accuracy, precision, f1 and recall.

2.3.2 Evaluation of Kenlm Model

The evaluation of the Kenlm [10] model was carried out on the ARPA [11] file created during the training process. We utilized the arpa [11] file to create a vocabulary consisting of unique tokens which are mostly the tokens that appear under the one grams. Each line in the test data excluding the last token was concatenated with each word in the vocabulary to ascertain the token from the vocabulary that results in the maximum score chosen as the next predicted token. Figure 3 shows the result of evaluating the model from ngrams order 2 to 20 with the highest accuracy and recall score attained in ngram order 5 at 0.577. On the other hand the model had the peak precision in ngram order 4 with a score of 0.762 and had the highest f1 score of 0.539 similar to accuracy and recall in ngram order 5.

2.3.3 Evaluation of bilstm model

The evaluation of the bi-lstm [0] model was done using the accuracy and loss metrics made available on the keras.Sequential fit method [0]. The training done on 50 epochs as shown in Figure 4 has results showing a max training accuracy closer to 0.70 at the 9th epoch as saw a downward spiral in training accuracy after 10 epochs. The training loss as shown in Figure 4 also experienced a downward decline in training loss as we train on more epochs. Initially we had a training loss of 1.043 with the lowest training loss of 0.9140 seen at 8 epochs.

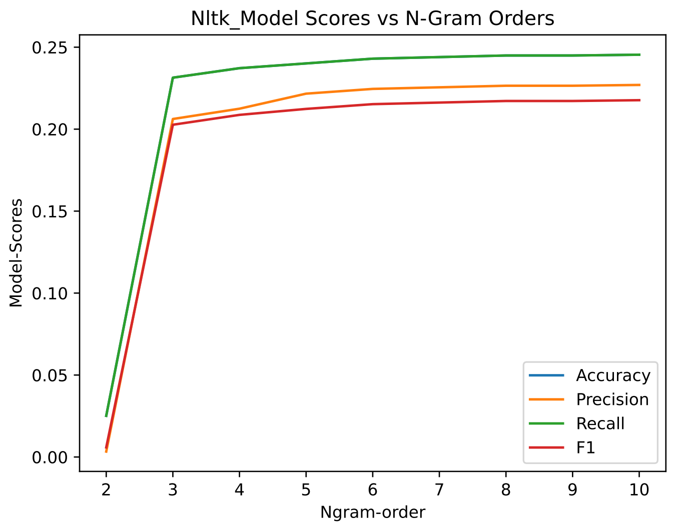

Figure 2.1: Nltk Model Scores vs NGram Orders 2_10

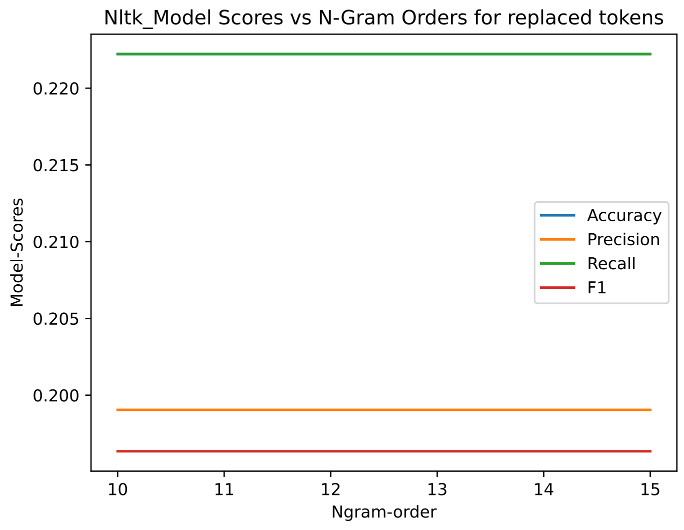

Figure 2.2: Nltk Model Scores vs NGram Orders 10_15

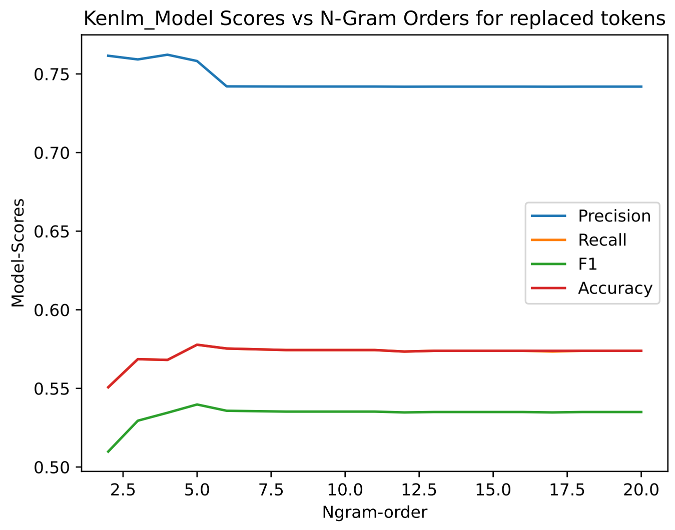

Figure 2.3: Kenlm_Model Scores vs NGram Orders for replaced tokens

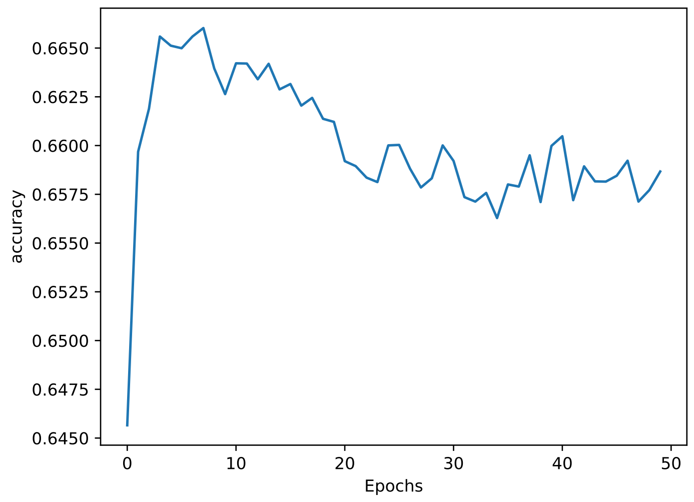

Figure 2.4: Training Accuracy Plot on Bilstm Model

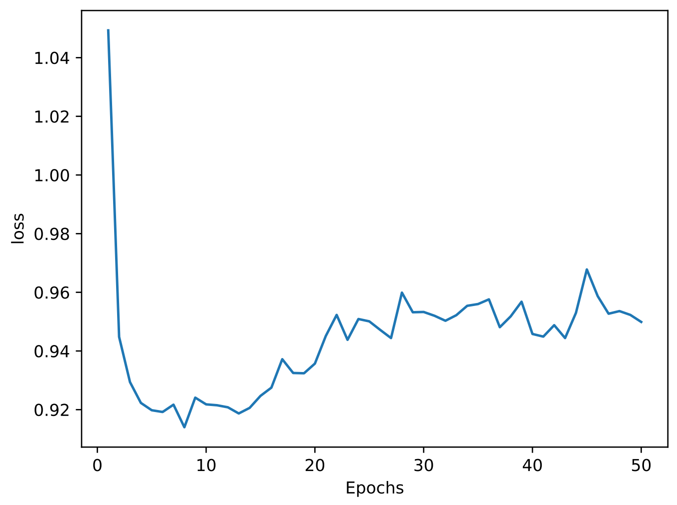

Figure 2.5: Training Loss Plot on Bilstm Model

References

[1] M. Berry, “Welcome to scratch 3!,” 2019.

[10] K. Heafield, *kenlm*, https://github.com/kpu/kenlm, 2013.

[7] E. Loper and S. Bird, “Nltk: The natural language toolkit,” *arXiv preprint cs/0205028*, 2002.

[0] B. Jang, M. Kim, G. Harerimana, S.-u. Kang, and J. W. Kim, “Bi-lstm model to increase accuracy in text classification: Combining word2vec cnn and attention mechanism,” *Applied Sciences*, vol. 10, no. 17, p. 5841, 2020.

[2] L. Grunske, K. Winter, and N. Yatapanage, “Defining the abstract syntax of visual languages with advanced graph grammars—a case study based on behavior trees,” *Journal of Visual Languages & Computing*, vol. 19, no. 3, pp. 343–379, 2008.

[3] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, “Foundations of json schema,” in *Proceedings of the 25th international conference on World Wide Web*, 2016, pp. 263–273.

[5] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. Devanbu, “On the naturalness of software,” *Communications of the ACM*, vol. 59, no. 5, pp. 122–131, 2016.

[4] O. Kramer and O. Kramer, “Scikit-learn,” *Machine learning for evolution strategies*, pp. 45–53, 2016.

[6] nltkproj, *nltk*, https://www.nltk.org/api/nltk.tokenize.word_tokenize.html, 2023.

[8] nltkproj, *$nltk_padd$*, https://www.nltk.org/api/nltk.lm.preprocessing.html, 2023.

[9] nltkproj, *nltkmle*, https://www.nltk.org/api/nltk.lm.html, 2023.

[11] D. H. Klatt, "Review of the arpa speech understanding project," *The Journal of the Acoustical Society of America*, vol. 62, no. 6, pp. 1345–1366, 1977.

[12] K. Heafield, I. Pouzyrevsky, J. H. Clark, and P. Koehn, "Scalable modified kneser-ney language model estimation," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013, pp. 690–696.

[13] TensorProj, *Tensorflow pad sequences*, https://www.tensorflow.org/api_docs/python/tf/keras/utils/pad_seq, 2023.

[14] T. Project, *Tensorflow*, https://www.tensorflow.org/api_docs/python/tf/keras/Model, 2023.

[0] K. Project, *Keras sequential fit*, https://www.tensorflow.org/api_docs/python/tf/keras/Sequential, 2023.