Part-of-Speech Tagging using Genetic Algorithms

Kamaljot Singh
Department of Computer Science and Engineering
Lovely Professional University, Jalandhar, Punjab, India.
Kamaljotsingh2009@gmail.com

***Abstract* — To the best of our knowledge genetic algorithms have never been used for prediction of POS tags for Punjabi Language. In this paper, A classic Genetic Algorithm (GA) with fixed gene length is proposed for sentence-level Punjabi language tagging. It uses fixed individual size, value type encoding, Roulette wheel selection, adaptive - two point crossover (TPC) and varying mutation rate as operators in proposed work. Focusing on the relationship of tags according to context, we are proposing this technique in form of a software prototype and an algorithm. A dataset of 26,000 hand tagged words is used for proposed work and 90.63% accuracy is achieved.**

Keywords-Genetic Algorithm; Punjabi; Part of speech; Natural Language Processing;

I. INTRODUCTION

Part-of-speech (POS) tagging is a basic task in natural language processing (NLP) that aims to determine the most likely lexical tag for an occurrence of a word in a natural language sentence. As natural languages are ambiguous in nature i.e.: many tags for same word, the tagging also involves process for disambiguating the POS information for ambiguous words by taking context information into consideration. There are many NLP tasks which can be improved by applying disambiguation to the text [1] and it has very wide range of applications such as grammar checker, Speech Synthesis, Speech Recognition, on-line translation etc. [2].

For POS tagging, many researchers had proposed many techniques for POS tagging as: Kashyap, D. et.al. [2] had presented a part-of-speech tagging approach using back propagation multi-layer perceptron neural networks learning algorithm for Punjabi language. He had used a Tri-gram model for generation of feature vector for every word and 88.95% accuracy is achieved from the tagger. Sharma, S.K. et.al. [3], had proposed a Bi-gram Punjabi POS Tagger using Hidden Markov Model. He had used a Punjabi corpus containing 26,479 words and achieved an accuracy of 84.90%. Gill, M.S. et.al. [4], had proposed Punjabi Part-of-Speech rule based tagger using fine-grained tag set. Their proposed tagger had achieved 80.61%.

Various tag-sets that are proposed in Punjabi language are: Kumar, D. et.al. [5], had proposed a coarse-grained tag set for Punjabi language consisting of 38 tags. Yoonus, Mohamed M. et.al. [6] had developed a part-of-speech tagger for 12 Indian languages using hybrid approach and presented a tag set consisting of 78 tags and the performance of taggers. They trained the taggers on 80k to 85k tagged corpora for each language. They concluded that their proposed tagger's efficiency and accuracy increases on increasing the training data size but on unknown tokens the system takes more time. Sankaran, B. et.al. [7] had proposed a common 3-level hierarchal POS tag set for Indian Languages by performing in depth analysis of 8 languages from 2 major families, viz. Dravidian and Indo-Aryan. There proposed tag set constitutes of 630 fine-grained tags.

In this paper, we propose a predictive modeling technique for predicting part-of-speech tag using Genetic Algorithm(GA) by considering the context of the words in sentence. The context of the word, in which the word appears helps to decide that which is the most appropriate tag and this idea of tagging is basic for most of the tagging systems [8].

The rest of the paper is organized as in Section II, The Punjabi is over-viewed, In Section III, some domain specific concepts are described. Section IV discussed the Present Approach, Implementation, Algorithm in detailed manner. Experimental Settings are detailed in Section V. The results of the experiments are detailed in Section VI. Finally, the paper is closed with the Conclusion, Future Scope, Appendix and References.

II. DOMAIN SPECIFIC CONCEPTS

Before defining the problem, we are going to discuss some domain specific concepts as under:

- *Search Space:* It is the collection of all feasible solutions. It we are talking about the solution or individual then we are talking about one point among feasible solutions - one point in search space.
- *Population pool:* It is collection of candidate solution to a problem. Under each generation, pair of 2-2 Individuals are selected from this population pool and then crossover and mutation is applied over them, to produce the solutions or new population of population pool.

- *CORPUS:* It is a collection of large and structured set of the text. Under this problem, it is a collection of text stored as a *(word, tag)* pair.
- *Population_SIZE:* It is the count of candidate solutions in the population pool.
- *Gene_SIZE:* It is the length of sentence or words count that are under consideration for tagging.

III. Present Approach

The tagging technique that is presented in this paper performs POS tagging at sentence-level. The sentence level tagging approach performs better than word-level approach as word level allows unlike combination of tags [2]. In this research, we used the Genetic Algorithm based approach for Part-of-speech tagging of Punjabi language.

A. Genetic Algorithm

The Genetic algorithm (GA) is a stochastic search heuristic that camouflage the process of natural evolution. It belongs to a broader class of evolutionary algorithms (EA's) that generates the solution to problem using natural evolution inspired techniques. Genetic algorithm basically works with some operators like selection, crossover and mutation by *encoding* the problem into set of candidate solutions/ individuals and finding up of the most appropriate (fittest) solution out of the search space.

1) Encoding

Before applying Genetic Algorithm, we have to transform the problem data into [1] chromosomes for computational purposes. So, for this transformation we used *value* type encoding and transformed the sentence of [2]Gene_SIZE into [3]Population_TEMPLATE.

The Population_TEMPLATE is basically a collection of lists that contains possible tags for particular Gene in a chromosome. Single possible tags list contains a possible tags list for single word *(word(i))*. This list is developed in accordance to the occurrence of the word in dictionary *([4]Word_DICT)*. e.g.:

- For non-ambiguous words, the length of Possible tag list is exactly one.
- For ambiguous words, this length is strictly equals to word ambiguity in dictionary.
- For Unknown words, this length may extend to the total count of the tags. i.e.:38.

Here in this work, we are only considering the context of the word and the words itself are ignored in Population_TEMPLATE. The is due to enormous amount of Punjabi words, it is not feasible to make any statistical relationship between them.

2) Population Generation

The Population_TEMPLATE, that is generated in encoding phase is used for Population generation. This phase will generate individuals as:

- If there is only one tag in possible tag list, then this tag will remain same in all individuals.
- If there is ambiguity of a word occurrence, then a tag is selected randomly among possible tags and individuals are developed.
- If the word is unknown, i.e.: length of possible tag list is 38. Then, the tag is selected on basis of total occurrence of tag in training table using roulette wheel.

This is repeated for all the gene's to develop an individual and [5] Population_COUNT individuals are developed.

3) Fitness evaluation

It is a total correctness probability of an individual. It is computed in accordance to the individual occurrence in training table.

$$Fitness(i) = \frac{count(Individual(i), Training\_Table)}{\sum_{j=1}^{Population\_SIZE} count(Individial(j), Training\_Table)} \quad (1)$$

where, Individual is a tag sequence whose fitness is to be evaluated and the Training_Table is basically a support table which is developed from Training set containing all possible tag combinations.

4) Selection

The very first step of Genetic Algorithm is selection of individuals from population pool for crossover. During each generation, groups of two-two individuals are selected from the population pool using *roulette wheel selection*.

In roulette wheel selection, the individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness. A random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained [9].

5) Crossover and Mutation

The next step is to generate the set of individuals for next population of solutions through the process of *crossover* or recombination. Under this work, *two-point crossover(TPC)* approach is used. In two-point crossover approach, two crossover points are chosen and the contents between these points are exchanged between two mated parents [10], [11]. However, an advantage of having more crossover points is that the search space may be searched more thoroughly [12]. *Mutation* is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next.

B. Design

Initially, the main tagged corpus is divided into the training set and the testing set by a random split. The training set is then used to generate:

[1] Individuals under GA called as chromosomes.
[2] Fixed variable defining the length of the chromosome.
[3] It is a blueprint to develop the individuals.
[4] It is dictionary that contain pair of known words.
[5] It is a variable defining the number of individuals in generation.

1) Word occurrence dictionary (word_DICT)

It consists of a word, its index, frequency, ambiguity, ambiguous tags frequency. This information is maintained in the dictionary for each word of the training corpus.

2) Training Table

This file contains the different contexts of each tag, describing the support of each individual.

3) Tag occurrence table

This file includes the frequency of different tags in the training data set.

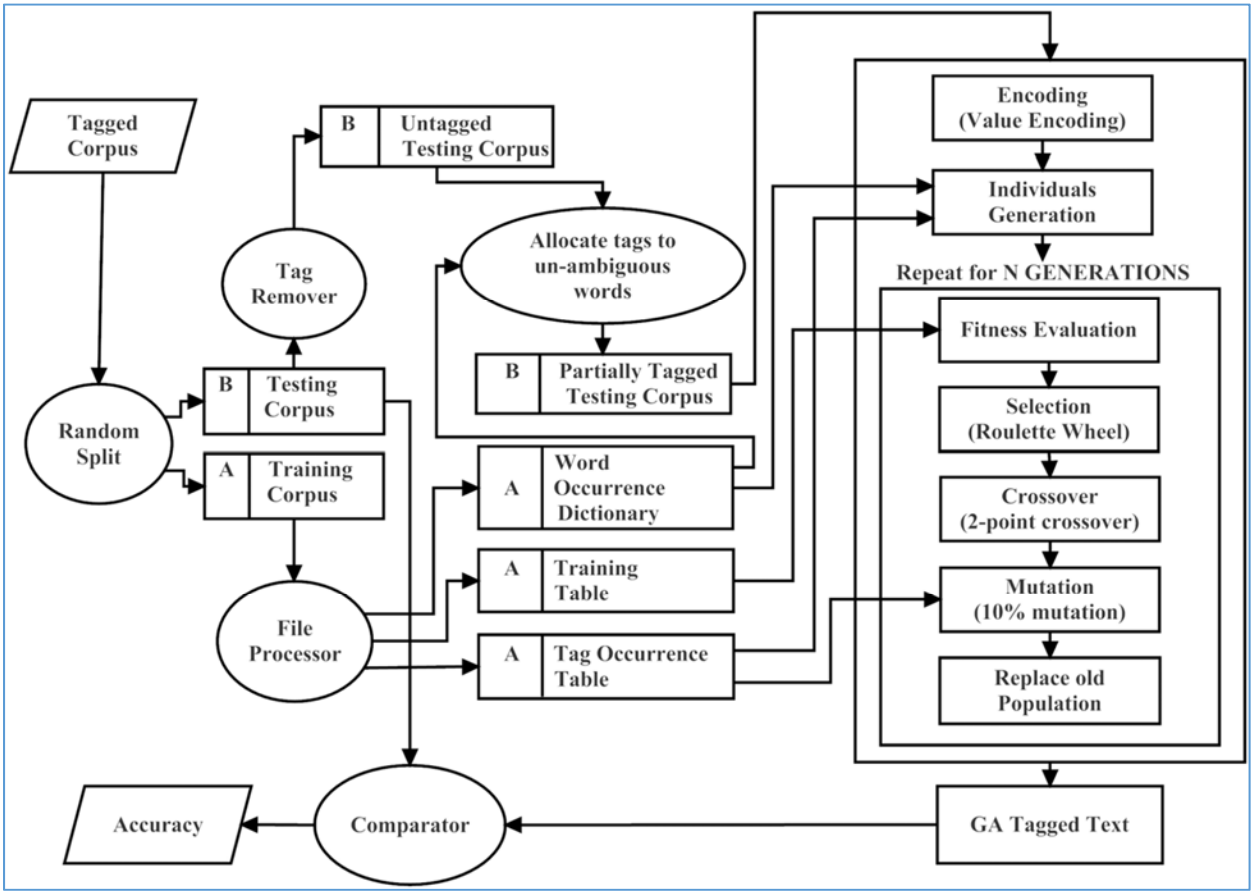

Figure 1: POS tagging Process.

The testing file is then passed to a tag remover module that removes tags from file (The original tagged testing file is also preserved at some different location on system). The purpose of removing the tags from testing file is re-tag it using GA and compare it against the original file in order to test its performance. A comparator module is developed whose work is to compare original testing set and GA tagged set for accuracy. The whole process is demonstrated in Figure 1.

C. Tagging process

We assign the tags to the words under this tagging process. The words to be tagged basically falls under three categories:

- Un-Ambiguous words (have un-ambiguous occurrence in dictionary).
- Ambiguous words (having multiple tags in dictionary).
- Unknown words (not in dictionary).

The tagging process of these three word categories are as under:

- For the un-ambiguous, the tag is directly assigned on the basis of their presence in the word occurrence dictionary.
- For the ambiguous words and Unknown words, the Population_TEMPLATE is developed and the GA procedure is called, that assigns tags to those words.

D. Implementation

This work of part-of-speech tagging had been done in JAVA. The Algorithm 1, demonstrates the whole procedure, starting with CORPUS passing to the accuracy measure. This algorithm takes CORPUS, Gene size, population count in generation and mutation rate as input parameters.

The *Step 1*, is basically splitting up of the CORPUS into 2 disjoint subsets, (1) Training set and (2) Testing set. The training set acts like a support of individuals and the testing set is to check the accuracy if the whole tagging

process. This split is done using a random number from 0 to 100, that defines the % of the training set and the remaining percent acts as testing set. This way we generated the training and testing set out of CORPUS.

Step 2, deals with the pre-processing of the training set in order to develop 3 files, which are later required for tagging, (1) Word dictionary, (2) Training table/ support table and (3) Tag dictionary. From the other tagging techniques, this technique also has an advantage that we don't need the train the machine often when the new data is available, we can pre-process the data whenever necessary and by this, the time of training is heavenly reduced.

The testing set is untagged as separate copy in *Step 3*. By this step, we can re-tag the un-tagged testing set and compare it with original testing set to find out the accuracy of the technique.

The *Step 4*, basically deals with the Un-ambiguous words tagging using the word dictionary file that is generated in *Step 1*. Only the words found in dictionary as un-ambiguous are tagged in this step, other word are marked using 2 different identifiers as ambiguous and unknown.

ALGORITHM 1: POSUSINGGA (CORPUS, GENE_SIZE, POPULATION_SIZE, GENERATION_COUNT, MUTATION_RATE)

```
STEP 1: TRAININGSET, TESTINGSET ← RANDOMSPLIT(CORPUS)
FUNCTION RANDOMSPLIT(CORPUS)
    NUM ← GENERATERANDOMNUMBER(0,100);
    TRAININGSET, TESTINGSET ← BREAKCORPUS(CORPUS, NUM %);
    RETURN TRAININGSET, TESTINGSET
END FUNCTION

STEP 2: WORD_DICT, TRAINING_TABLE, TAG_DICT ← FILE_PROCESSOR(TRAININGSET)
FUNCTION FILE_PROCESSOR(TRAININGSET)
    WORD_DICT ← GENERATEDICTFILE(TRAININGSET);
    TRAINING_TABLE ← REMOVEWORDS(TRAININGSET);
    TAG_DICT ← REMOVEWORDSANDCOUNTTAGS(TRAININGSET);
    RETURN WORD_DICT, TRAINING_TABLE, TAG_DICT
ENDFUNCTION

STEP 3: UNTAGGEDTESTINGSET ← TAGREMOVER(TESTINGSET)
FUNCTION TAGREMOVER(TESTINGSET)
    UNTAGGEDTESTINGSET:= NULL
    FORALL WORDS w(i) OF TESTINGSET
        TAG ← w(i).TAG
        WORD ← w(i).WORD
        UNTAGGEDTESTINGSET.APPEND(WORD)
    ENDFOR
    RETURN UNTAGGEDTESTINGSET
ENDFUNCTION

STEP 4: PARTIALLYTAGGEDSET ← UNAMBIGUOUSWORDSTAGGING(UNTAGGEDTESTINGSET, WORD_DICT)
FUNCTION UNAMBIGUOUSWORDSTAGGING(UNTAGGEDTESTINGSET, WORD_DICT)
    PARTIALLYTAGGEDSET := NULL
    FORALL WORD w(i) OF UNTAGGEDTESTINGSET
        IF w(i) EXIST IN WORD_DICT
            IF w(i) IS NON-AMBIGUOUS
                w(i).TAG ← ASSIGN TAG FROM WORD_DICT
            ELSE
                w(i).TAG ← 'A:AMBIGUOUS'
            ENDIF
        ELSE
            w(i).TAG ← 'UKN:UNKNOWN'
        ENDIF
        PARTIALLYTAGGEDSET.APPEND(w(i))
    ENDFOR
    RETURN PARTIALLYTAGGEDSET
ENDFUNCTION
```

```
STEP 5: GATAGGEDSET ← GATAGGING(PARTIALLYTAGGEDSET, WORD_DICT, TRAINING_TABLE, TAG_DICT, GENE_SIZE,
POPULATION_SIZE, GENERATION_COUNT, MUTATION_RATE)

FUNCTION GATAGGING(PARTIALLYTAGGEDSET, WORD_DICT, TRAINING_TABLE, TAG_DICT, GENE_SIZE,
POPULATION_SIZE, GENERATION_COUNT, MUTATION_RATE)

    GATAGGEDSET := NULL
    FOR I:=1 TO LENGTH(PARTIALLYTAGGEDSET) STEP GENE_SIZE          TRAVERSING WHOLE FILE FOR TAGGING
        POPULATION_TEMPLATE := NULL
        FOR J:=I TO I+GENE_SIZE                                    DEVELOPMENT OF POPULATION_TEMPLATE
            POSSIBLETAGS := NULL
            IF WORD w(i).TAG IN PARTIALLYTAGGEDSET == 'A:AMBIGUOUS'
                POSSIBLETAGS = GETAMBIGIOUSTAGS(WORD_DICT, w(i))
            ELSEIF WORD w(i).TAG IN PARTIALLYTAGGEDSET == 'UKN:UNKNOWN'
                POSSIBLETAGS = ALLTAGS
            ELSE
                POSSIBLETAGS = w(i).TAG
            ENDIF
            POPULATION_TEMPLATE.APPEND(POSSIBLETAGS)
        ENDFOR                                   END OF DEVELOPMENT OF POPULATION_TEMPLATE LOOP

        POPULATION := GENERATEPOPULATION (POPULATION_TEMPLATE, POPULATION_SIZE)
        CROSSOVER_CHROMOSOME := GENERATECROSSOVER_CHROMOSOME(POPULATION_TEMPLATE)

        FOR GENERATION := 1 TO GENERATION_COUNT                    GENERATIONS LOOP
            FITNESS ← EVALUATEFITNESS(POPULATION)
            FOR J := 1 TO POPULATION_SIZE/2                        SELECTION, CROSSOVER & MUTATION LOOP
                INDIVIDUAL_1 = SELECTINDIVIDUALUSINGROULETTEWHEEL(POPULATION)
                INDIVIDUAL_2 = SELECTINDIVIDUALUSINGROULETTEWHEEL(POPULATION)
                NEW_INDIVIDUAL_1, NEW_INDIVIDUAL_2 ← PERFORMCROSSOVER(INDIVIDUAL_1, INDIVIDUAL_2,
                 CROSSOVER_CHROMOSOME)

                MUTATED_NEW_INDIVIDUAL_1 = MUTATE (NEW_INDIVIDUAL_1, MUTATION_RATE)
                MUTATED_NEW_INDIVIDUAL_2 = MUTATE (NEW_INDIVIDUAL_2, MUTATION_RATE)
            ENDFOR                                 END OF SELECTION, CROSSOVER & MUTATION LOOP
            REPLACE OLD POPULATION WITH NEW
        ENDFOR                                                     END OF GENERATIONS LOOP

    FITNESS ← EVALUATEFITNESS(POPULATION)
    ASSIGN TAGS OF THE FITTEST INDIVIDUAL TO THE WORD SEQUENCE FROM i TO (i + GENE_SIZE)
    ENDFOR                                   END OF TRAVERSING WHOLE FILE FOR TAGGING LOOP
    RETURN GATAGGEDSET
  ENDFUNCTION

STEP 6: ACCURACY ← COMPARATOR(TESTINGSET, GATAGGEDSET)

FUNCTION COMPARATOR(TESTINGSET, GATAGGEDSET)
    CORRECT := 0
    FORALL WORD w(i) IN TESTINGSET & WORD G(i) IN GATAGGEDSET
        IF w(i).TAG == G(i).TAG
            CORRECT := CORRECT + 1
        ENDIF
    ENDFOR
    ACCURACY := (CORRECT/LENGTH(TESTINGSET))*100
    RETURN ACCURACY
ENDFUNCTION

STEP 7: EXIT.
```

Step 5, it is the core of the whole work. In this step, the untagged words i.e.: the ambiguous and the Un-known words are tagged using the genetic algorithm. This step includes the population pool generation, execution of generations, fitness evaluations, selection of individuals using roulette wheel, crossover using two-point crossover and mutation. This step requires a partially tagged set, word dictionary, training table, gene size, population size, generation count, and mutation rate. This step proceeds as:

- Initially, a template is generated for the development of the population pool and parallely one more Individual is generated that defines the possible crossover points known as crossover chromosome.
- Following steps are repeated for all generations:
  - Fitness of all individuals of population pool is evaluated using a fitness function provided in Equation 1.
  - Two individuals are selected from the population pool using fitness based roulette wheel selection method.
  - The crossover operator is applied to the selected individuals using crossover chromosome and set of offspring individuals is developed.
  - These off springs are then subjected to the random mutation on basis of the population template and mutation rate.

Step 6, It compares the original testing file against the Genetic algorithm tagged file and returns the accuracy of the tagger.

IV. Experimental settings

For the experiments, the proposed tag-set of [5], is used in this work. This tag-set consist of 38-tags coarse grained tags.

A hand-tagged corpus consisting of 26,000 words is used for this proposed work. The corpus data is collected from news articles, essays, stories etc. The Punjabi corpus is tagged with respect to [5]. The data is divided into 2 sets: Training set (20,000 words) and Testing set (6,000 words) by a random function corpus splitter.

The Training set is then passed to the pre-processor module that processes it and develop 3 files (word dictionary, Training Table, Tag dictionary). The testing set is untagged using a tag remover module and then again tagged by GA. The Comparator module, compares the GA tagged test set against the original test set to measure the accuracy of system. Then compared against the original testing file and accuracy is measured.

TABLE I. EXPERIMENTAL SETTINGS SUMMARY

Main corpus size	26,000
Training corpus size	20,000
Testing corpus size	6,000
Individual's Encoding type	Value Type
Crossover Type	Adaptive 2-point
Mutation Rate	10%, 20%

The GA is executed with Value type encoding, an adaptive two-point crossover and 10%, 20% mutation rate. The summary of the experimental settings is provided in IV. The tagging process is executed number of times by varying the configuration of the Genetic Algorithm. The results are presented in next section.

V. Results

Table II, demonstrates the results obtained, when GA is executed several times by varying Gene_SIZE, Population_SIZE, Generations_COUNT, and Mutation_Rate on same training and test set. Following are the outcomes of the experiment:

TABLE II. EXPERIMENTAL RESULTS.

Gene SIZE	Population SIZE	Generations Count	Mutation Rate	Accuracy
5	32	10	10%	88.33%
			20%	87.90%
		20	10%	86.05%
			20%	87.88%
	64	10	10%	88.23%
			20%	88.68%
		20	10%	88.86%
			20%	88.63%
	96	10	10%	89.13%
			20%	**89.25%**
		20	10%	88.96%
			20%	89.03%
4	32	10	10%	89.61%
			20%	90.28%
		20	10%	89.78%
			20%	89.88%
	64	10	10%	90.35%
			20%	90.36%
		20	10%	**90.63%**
			20%	90.33%
	96	10	10%	90.48%
			20%	90.53%
		20	10%	90.51%
			20%	90.43%
3	32	10	10%	90.53%
			20%	90.43%
		20	10%	90.38%
			20%	90.43%
	64	10	10%	90.95%
			20%	**91%**
		20	10%	90.80%
			20%	90.88%
	96	10	10%	90.80%
			20%	90.95%
		20	10%	90.77%
			20%	90.83%

- This is seen that the Accuracy is increasing as we decrease the Gene_SIZE.
- Accuracy is increasing as we increase the Population_SIZE.

- When we re-run the genetic algorithm for tagging, results of same configuration deviate from each other by ±0.1, due to highly dynamic nature of GA.

A. Comparasion with existing taggers

The Accuracy that is achieved on Gene_SIZE 3, consumes lots of time for computation and thus, we can't consider the Gene_SIZE 3 results. For comparison with other taggers we will only consider the Gene_SIZE 4 and Gene_SIZE 5 results. For the comparison with existing taggers, we are using the Gene_SIZE 4, Population_SIZE 64, Generation_COUNT 20, Mutation_RATE 10% configuration of GA based POS tagger.

TABLE III. COMPARISON WITH OTHER TAGGERS

Tagging technique	**Accuracy**
GA based Tagger (Proposed)	90.63%
Neural Network based Tagger	88.95%
HMM based Tagger	84.90%
Rule based Tagger	80.61%

The proposed Genetic Algorithm based Punjabi POS tagger was compared with existing Punjabi POS taggers as: a HMM based tagger [3], a rule based tagger [4] and Neural network based tagger [2]. The results showing that our proposed tagger performed better than existing taggers for Punjabi language in terms of accuracy is shown in Table III.

VI. CONCLUSION AND FUTURE WORK

The accuracy of 90.63% is achieved by using training set of 20k words and test set of 6k words. It is also seen that the sentence level tagging also works better than the word level tagging methods. This work is a preliminary work towards a future goal of Punjabi Language Speech Synthesis. Our proposed GA based POS tagger works very well and providing us with very well accuracy but their is further a room of improvement in it by using a multi-model approaches or some better classifiers.

REFERENCES

[1] C. D. Manning, H. Schu¨tze, Foundations of statistical natural language processing, MIT press, 1999.

[2] D. Kashyap, G. Josan, A trigram language model to predict part of speech tags using neural network, in: H. Yin, K. Tang, Y. Gao, F. Klawonn, M. Lee, T. Weise, B. Li, X. Yao (Eds.), Intelligent Data Engineering and Automated Learning IDEAL 2013, Vol. 8206 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 513–520. doi:10.1007/978-3-642-41278-3 62.

[3] S. Sharma, G. Lehal, Using hidden markov model to improve the accuracy of punjabi pos tagger, in: Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on, Vol. 2, 2011, pp. 697–701. doi:10.1109/CSAE.2011.5952600.

[4] M. S. Gill, G. S. Lehal, S. S. Joshi, Part of speech tagging for grammar checking of punjabi, The Linguistic Journal 4 (1) (2009) pp. 6–21.

[5] D. Kumar, G. S. Josan, Developing a tagset for machine learning based pos tagging in punjabi, International Journal of Applied Research on Information Technology and Computing 3 (2012) pp. 132–143.

[6] M. M. Yoonus, S. Sinha, A hybrid pos tagger for indian languages., Language in India 11 (9).

[7] B. Sankaran, K. Bali, T. Bhattacharya, P. Bhattacharyya, G. N. Jha, S. Rajendran, K. Saravanan, S. L. Devi, K. Subbarao, Designing a common pos-tagset framework for indian languages., in: IJCNLP, 2008, pp. 89–92.

[8] E. Alba, G. Luque, L. Araujo, Natural language tagging with genetic algorithms, Information Processing Letters 100 (5) (2006) pp. 173 – 182.

[9] H. Pohlheim, Geatbx: Genetic and evolutionary algorithm toolbox for use with matlab, H. Pohlheim, Berlin, http://www.geatbx.com.

[10] L. Booker, Improving search in genetic algorithms, Genetic algorithms and simulated annealing (1987) pp. 61–73.

[11] M. Kaya, The effects of two new crossover operators on genetic algorithm performance, Applied Soft Computing 11 (1) (2011) pp. 881– 890.

[12] Y. Kaya, M. Uyar, et al., A novel crossover operator for genetic algorithms: ring crossover, arXiv preprint arXiv:1105.0355.

Kamaljot Singh, Assistant professor at Lovely Professional University, INDIA has received his master degree from DAV University in 2015. He had received a Gold medal from NASA, Johnson Space center in 2008 for rover designing. His research interest includes data mining, wireless sensor networks, natural language processing, Nano-magnetic materials and nature inspired computations. He had several publications in reputed journals and in international conferences.