Comparison based approach to Sequence Labelling for NLP Tasks

We compare two different Machine Learning approaches and their performance and accuracy using pre-trained word vectors from Glove(Stanford) and CONLL-2003 Shared task dataset (English)

Authors:

Samartha Kajoor Venkatramana

Siddharth Sharma

Chinonso Larry Okeke

UMass Lowell 2018

Abstract

We have used two machine learning algorithms, namely One-vs-All Logistic Regression and Multi-Layer Perceptron, to bring out the differences in performance and accuracy. We have tested the system by performing Sequence labeling task over the CONLL-2003 Shared Task English language dataset [1], to train and test the system. The main goal of this experiment was to perform error analysis and populate the scores over some of the more common tasks of NLP, such as Named Entity Recognition [2], Parts-Of-Speech Tagging and Chunking. Our dataset was in the form of "WORD NER-tag POS-tag Chunking-tag". We have used the external library vsmlib [3] written in python to process the pretrained vectors derived from the Glove [4], load the model and process all the vocabulary. Next, we have processed the data for fitting into the machine learning model. Finally, the results, F1 score or score was generated on the predictions for both the LR Classifier and the MLP Classifier model. We had also kept track of the running times of various operations, especially the fitting phase. We have conducted experiments with different word embedding models distinguished by different dimensions or sizes. We have also performed experimentation with different context width for preprocessing, and further, have used different hyper parameters to increase performance or efficiency.

Keywords: Natural Language Processing, NLP Tasks, scikit-learn, linear model, one-vs-rest logistic regression, machine learning, performance metrics, Multi-Layer Perceptron

Introduction

In the field of natural language processing, certain functions are commonly viewed as classification problems. These functions, namely NER, POS, and chunking, are logically simple to understand, but harder to implement on a machine [15]. Basically, we are annotating every word in a sentence of text to its respective tag. Each type of task, NER or POS or chunking have their own pre-defined tags. These tags along with the words, in turn the whole sentences, readable vertically, in large quantities, form our dataset. We acquired this data set from the CONLL-2003 Shared Task which includes both English and German data, though for our experiments, we shall only consider the English text [1]. The paper defines the shared task as language independent named entity recognition. This named entity data came in 4 parts. We have a training file, development file, test file and a large file containing un-annotated data. We have made our task simpler, using only the first 3 files to run the experiments. The types of named entities considered are person, locations, organizations, and names of miscellaneous entities not falling under the previous categories.

One of the main motivations for performing these tasks is to enable machines to understand the underlying semantics and grammar of a language. Consider an example sentence, such as

	SOCCER	**JAPAN**	**GET**	**LUCKY**	**WIN**	**CHINA**	**IN**	**SURPRISE**	**DEFEAT**
NER	NN	NNP	VB	NNP	NNP	NNP	IN	DT	NN
POS	B-NP	B-NP	B-VP	B-NP	I-NP	B-NP	B-PP	B-NP	I-NP
CHU	O	B-LOC	O	O	O	B-LOC	O	O	O

Table 1

This sentence contains a bunch of named entities and parts of speech. As seen above, we have the data in this form, where the words are placed vertically with the next three columns describing the annotations. Each word has three tags, corresponding to NER, POS, and chunking. If you notice the tags are simple to understand, describing nouns, pronouns, parts of speech and location etc. One can refer to the CONLL paper to understand what each of the tag means. There are many categories and tags to help the system understand the sentence.

The problem of natural language processing is evidently useful in today's world, where there are gigabytes of textual data being uploaded and downloaded on the internet. We have mass amounts of user related text data, essays and other corpus that can be automatically processed to determine some characteristics about the person or corpus under consideration. The applications include but aren't limited to machine translation (from one language to another), spam identification, information extraction (such as financial information from the news), text summarization, answering questions, and much more. These areas are very important in today's world, and NLP allows us to solve these problems using machines.

We approach this problem with what we have, one-vs-rest logistic regression classifier, and multi-level perceptron neural network. What we are trying to do is to understand when it comes to the internal mechanics of an application, what kind of hyperparameters and parameters matter and what exactly helps the machine understand natural language in a very concise way. Using these studies, when one is developing a similar system, he can decide very confidently which parameters he would need to keep track of to increase the accuracy scores. The biggest concern however is that there is a large amount of non-annotated data on the internet. We must develop a robust system to process all that data which becomes a daunting task if we do not know what parameters matter the most and how to use them to our advantage. This paper aims to help the future

researchers to easily proceed with the problem at hand instead of spending endless hours in figuring out the correct parameters. We provide in depth comparison between two famous approaches in machine learning. They can then improve their systems from the ground up, using the required type of dataset and annotation. The latest techniques involve using deep learning techniques to solve the problem, which has been yielding very promising results. We will not delve into deep learning as it is beyond our scope.

The dataset we use is provided from the CONLL-2003 Shared Task (English). The shared task means that both the annotated and the non-annotated data was used in some way to train the system [1]. However, we have just used the annotated data to train and test the system. The method for training on non-annotated data was beyond our scope.

We have built a program to help us test various parameters, written in python. The workflow is provided in the approach section. The program *reads* and *loads* the config file. Next, it *loads* the model word embeddings from the path provided in config. Then, it *loads* the dataset from the data folder provided in the config, carefully, by processing them into indices to words and labels, stored separately in two dictionary objects. This *load* function returns the *training set, testing set, and the validation set*, consisting of X and y values, and *a dictionary object containing two dictionaries*. The X vector holds the *ids to the words* while the y vector holds the *ids to the labels*. We use *one-hot encoding* to encode the y vector for further processing. This step occurs in the *getinpOutput* method. The X values are converted into context window based lists of indices. Next the *getX* method computes the word embeddings for each of the list of ids. Then, this whole training X and y are fitted into the classifier of choice, also included in the config file. Finally, the predictions are matched with the true values to generate accuracy or f1 score.

Background

There has been a lot of work done in Named-Entity recognition, for the English language [1]. The motivation roughly began with the Georgetown experiment, which involved the automatic conversion of 60 Russian sentences to English. The authors went on to claim that the machine translation problem will be solved within the next three to five years. However, this problem has turned out to be more complicated than previously assumed and real progress was much slower [5]. Further, after the ALPAC report was released in 1966, the ten years of experimentation had failed to produce any real results [6]. Even the funding for such programs was cut. The onset of the first statistical learning systems helped to improve the solutions, however, again not satisfying the requirements. The noticeably successful NLP systems were developed as early as 1960s, such as SHRDLU [7], and ELIZA [8], of which the latter, yielded better results, close to human-like interaction.

Further, during the 1970s there was a great deal of conceptual ontologies written by programmers, which converted the real-world data into computer parse able data. Examples of this are MARGIE [9], SAM [10], etc., which lead to the onset of chatbots, such as PARRY, Racter, etc. However, until the 1980s, most of these Natural language processing systems was based mainly on large sets of hand-written rules. However, from the 1980s onward, the introduction of machine learning created the revolution in the field of NLP, further supported by the introduction of better computation power, and moving away from the Chomskyan theories of linguistics [11]. Then, the systems improved using decision tree like machine learning, which was almost like hand-written rules implementation. Next, the HMMs, hidden Markov models, which produced soft probabilistic decisions, helped to produce better results similar to the statistical learning processes [12], all of which worked with real-valued weights, to the features in the input data. Most of the speech recognition systems nowadays rely on the cache language models as one paper describes here from the early 1990s [13].

These previous works listed above all help the current system architects decide the parameters and hyperparameters while they develop the algorithms we use nowadays. The authors of scikit learn, one of the more popular the machine learning library in python explain how they have kept both ease of use and computational efficiency in mind while developing the algorithms.

Approach

Basic approach – Window based concatenation of Word embeddings

1. Primarily, the program begins by loading the options from the config.yaml. We used a configuration yaml because we want to experiment with different tasks and our system processes the correct column from the dataset depending on the specific task, whether it is NER, POS or chunking. The yaml loader returns a python object, stored into "*options*" object for use throughout the program. The config file needs to be configured with the following fields:

path_vectors	*Path to word vectors such as GloVe*
path_dataset	*Path to the dataset on disk*
window	*Context window length for current trial – 1 through 5*
task	*Task to be considered out of NER/POS or chunking*
algorithm	*Machine Learning algorithm to be used – LRC or MLP*

Table 2 – Describing the configuration values

   The algorithm specific parameters need to be hardcoded at this point.

2. Next, our system loads the word embedding model from the path, i.e. GloVe word embeddings. We used the method

   vsmlib.model.load_from_dir(options['path_vectors'])

   to load the model, as shown above.

3. Then, the task is acquired and set from the options['task'].
4. Further, we use the load_data.load() method to load our data and the corresponding values from the labels column depending on the task. This results in the generation of *train_set, test_set, valid_set and dic*. These sets represent words and their corresponding labels from the task column. These need to be further processed to reflect word embeddings and vectors. The dic holds two dictionary objects, one for word-to-index and other for label-to-index.
5. Then, we use the getinpOutput() method to generate the word ids or indices to context window lists.
6. The next method getX() is used to convert these lists of context windows into word embeddings from those lists of indices.
7. We do this for both the training and testing dataset and to simplify our task, we have extended the training set to include the validation set as well.
8. Next, depending on the algorithm specified in config, we use either MLP classifier or LR classifier to fit and predict on the training and testing dataset.

We start by explaining a few of the methods we have created to process the data.

a. *contextwin(l, win)*

   Given a list of indices composing one sentence, this provides the list of list of indices corresponding to context windows surrounding each word in the sentence.

   Example: Consider the sentence "He went to school". If we set the context window to 2, we should have for each word, represented as a number, a list of the indices of the words before and after up to two words.

	HE	WENT	TO	SCHOOL
WORDtoIndex	445	1123	45	4040
OUT	[1123, 45]	[445, 45, 4040]	[445, 1123, 4040]	[1123, 45]

Table 3

This is just an example, the real values of these indices will vary, according to indexing and word embeddings model.

b. *getinpOutput(lex, y, win, idx2word)*

This method creates the required input and output lists for our machine learning algorithm to process. It processes the loaded training lex and testing lex, along with the y vector, to create the input and output lists for our machine learning algorithms.

c. *getX(inp, m)*

This method takes in the model and the input vector and converts it into word vectors from the model, if entry exists in the model. Else the word vector is initialized to random vector. This function makes the data from the previous method into a list of embeddings. This method also checks the tokens Out of Vocabulary and Vocabulary Cover Rate.

d. *loaddata.py – load(path, task)*

This file has the method required to process the train, test and validation dataset from the system path and then returns four objects, the ***out['train'], out['test'], out['valid'],*** and the ***dic*** object holding two dictionaries, one for **words2idx** and **label2idx**.

This is task specific loading of y vector based on tag position in the dataset, generation of the out object with the words and labels indexed accurately from the dataset which is in a format that cannot be handled without preprocessing.

Other supporting methods include:

e. *main()*

The main method encapsulates all these methods and utilizes them to generate the scores we publish in the results section.

The logistic regression algorithm we used was provided by the scikit learn library [19]. The scikit learn library describes their logistic regression classifier found in the linear_models package as

LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)

We do not set the parameters unless we really need to. We shall discuss them in the results section.

The MLP classifier class can be found in the scikit learn neural networks library. The class hold takes these constructor parameters as shown below

```
MLPClassifier(hidden_layer_sizes=(100, ), activation='relu', solver='adam',
alpha=0.0001, batch_size='auto', learning_rate='constant',
learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,
random_state=None, tol=0.0001, verbose=False, warm_start=False,
momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```

This model optimizes the log-loss function using LBFGS or stochastic gradient descent [16].

WorkFlow

Here we provide the basic workflow of the program we have composed.

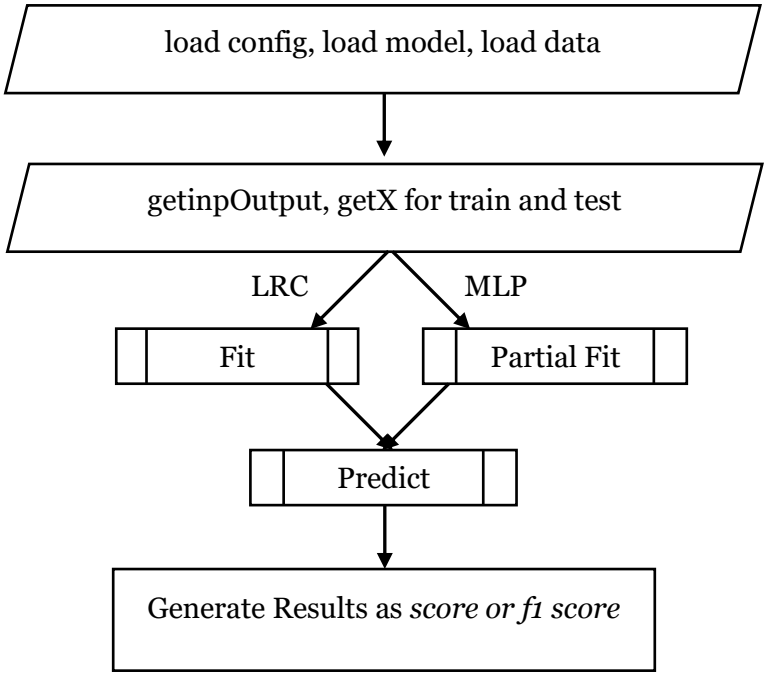


Fig 5 – Workflow diagram.

Results

Dataset - A note on CONLL-2003 Dataset

The CoNLL-2003 named entity data consists of eight files covering two languages: English and German. Each of the languages has a training file, a development file, a test file and a large file with unannotated data. The learning methods were trained with the training data. The development data could be used for tuning the parameters of the learning methods. The challenge of this year's shared task was to incorporate the unannotated data in the learning process in one way or another. When the best parameters were found, the method could be trained on the training data and tested on the test data. The results of the different learning methods on the test sets are compared in the evaluation of the shared task. The split between development data and test data was chosen to avoid systems being tuned to the test data.

The English data is a collection of Reuters Corpus. The annotation has been done by people of the University of Antwerp. Because of copyright reasons we only make available the annotations. To build the complete data sets one would have to access the Reuters Corpus.

The data files contain one word per line with empty lines representing sentence boundaries. A tag which states whether the current word is inside a named entity or not is found at each line. The tag also encodes the type of named entity. Each line contains four fields: the word, its part-of-speech tag, its chunk tag and its named entity tag.

Specifics of data used

- English data only.
- Training set merged with validation set.
- For faster processing, words converted into indices, then indices converted to list of indices based on context width, then these lists converted to lists of word embeddings for each of the index in list.
- Same procedure applied for labels of y set.

Experiments and performance evaluation

Experiments performed are by modification of config file and running the program, and are listed below

- Varying word embedding dimensions: 50d, 100d
- Varying context window size: 1,3
- Different tasks chosen from: [NER, POS, Chunk]
- Different algorithms: [Logistic regression classifier or MLP classifier]

Description of the target system

- Hardware – As shown by Windows system information details.

Processor	*Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz, 2401 MHz, 2 Core(s), 4 Logical Processor(s)*
System Type	*x64-based PC*
Physical Memory (RAM)	*16.0 GB*
Total Physical Memory	*15.9 GB*
Total Virtual Memory	*24.4 GB*
HDD	*500GB SSD*

Table 4

- Software

Language	*Python 3.6*
Libraries	*sys, numpy, sklearn.linear_model, sklearn.neural_networks, sklearn.metrics, warning, yaml, vsmlib, time*
Classes	*Logistic Regression, MLPClassifier,*
Methods	*f1_score, accuracy_score*

Table 5

Scoring metrics:

- F1 Score for NER and Chunking

  In statistical analysis of binary classification, the F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0 [20].

  Formula:

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Figure 6 – Formula for F1 score

- Accuracy score for POS

  The accuracy_score function computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions.

  In multilabel classification, the function returns the subset accuracy. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

Figure 7 – Formula for accuracy score

  Where y^ represents the predicted values, and y represents the true values. Formula derived from Sklearn [21].

Results and graphs

We did testing with different dimensions as mentioned above. With the LRC(Logistic Regression Classifier), it was very slow, but we still managed to get as much data as possible. We have used a output file to store all the results, and then used standard graphing tools to tabulate and graph the data. We have the tables as shown below:

	Embedding Size	Context Window	Time elapsed	Training Score	Test Score
NER	50	1	5.3255	0.9061	0.9009
POS	50	1	24.91232	0.790747	0.793854
Chunk	50	1	11.464	0.7432	0.7443
NER	50	3	13.903	0.9147	0.9084
POS	50	3	54.548	0.8014	0.7996
Chunk	50	3	27.417	0.762452	0.7603
NER	100	1	5.8496	0.9314	0.925
POS	100	1	28.089	0.8655	0.8586
Chunk	100	1	14.394	0.8362	0.827
NER	100	3	15.76	0.9401	0.9311
POS	100	3	69.07	0.8768	0.8623
Chunk	100	3	36.624	0.8589	0.8457

Table 6 – Logistic Regression Classifier

	Embedding Size	Context Window	Time elapsed	Training Score	Test Score
NER	50	1	1.0239	0.9751	0.9521
POS	50	1	1.2429	0.9007	0.8755
Chunk	50	1	1.3558	0.9145	0.8896
NER	50	3	1.5063	0.9869	0.9516
POS	50	3	1.7288	0.9089	0.8725
Chunk	50	3	1.6887	0.9317	0.893
NER	100	1	1.5662	0.986	0.9587
POS	100	1	1.7753	0.9387	0.9055
Chunk	100	1	1.7029	0.9452	0.9114
NER	100	3	2.6186	0.996	0.9551
POS	100	3	2.7081	0.9536	0.8988
Chunk	100	3	2.7457	0.9662	0.9114

Table 7 – Multilayer Perceptron Neural Network

The test proved that the out-of-vocabulary rate and the vocabulary cover rate for each algorithm, remains the same throughout and varies with varying context length.

	OOV	VCR	Context Length	Task
Train	6.820616	93.179384	1	NER
Test	7.800111	92.1999	1	NER
Train	6.82062	93.1794	1	POS
Test	7.80011	92.199889	1	POS
Train	6.82062	93.1794	1	CHUNK
Test	7.80011	92.1999	1	CHUNK
Train	6.82062	93.1794	1	NER
Test	7.80011	92.1999	1	NER
Train	6.82062	93.1794	1	POS
Test	7.80011	92.1999	1	POS
Train	6.82062	93.1794	1	CHUNK

Test	7.80011	92.1999	1	CHUNK
Train	13.3414	86.6586	3	NER
Test	14.9093	85.0907	3	NER
Train	13.3414	86.6586	3	NER
Test	14.9093	85.0907	3	NER
Train	13.3414	86.6586	3	POS
Test	14.9093	85.0907	3	POS
Train	13.341439	86.6586	3	CHUNK
Test	14.9093	85.0907	3	CHUNK
Train	13.341439	86.6586	3	CHUNK
Test	14.9093	85.0907	3	CHUNK
Train	13.3414	86.6586	3	POS
Test	14.9093	85.0907	3	POS

Table 8 – Out of vocabulary rate and vocabulary cover rate.

So consolidated, we have

	OOV	VCR	Context Length
Train	6.820616	93.179384	1
Test	7.800111	92.1999	1
Train	13.3414	86.6586	3
Test	14.9093	85.0907	3

Table 9 – Consolidated OOV and VCR measures.

We can see here that the out of vocabulary rate increases with increase in context width.

Next, we show the various graphs we have generated to help with our analysis.

Discussion

MLP Classifier is faster and more accurate than LR Classifier. The MLP classifier is easier to setup. Logistic regression classifier may not be well suited for this task. The decision boundary generated by the logistic regression might have overlap. Linear models do not do well predicting multiclass boundaries, with one-vs-rest. They might have multiple overlapping values, which makes it difficult for the system to predict on new data. There many chances for false positives, which makes it not a good choice of logistic regression as a great classifier.

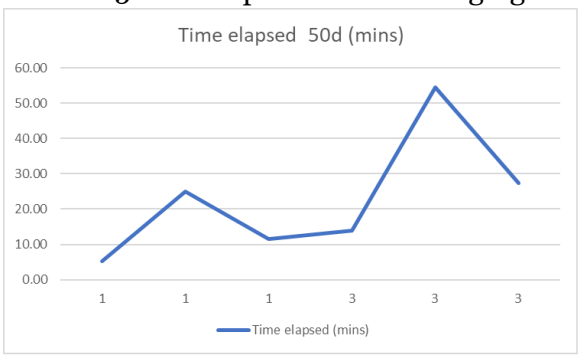
However, with the multilayer perceptron, we see that the training and test takes very less time. The accuracy is also higher. But we see that as the context width increased from 1 to 3, the accuracy scores for both the classifiers came down. We can conclude that accuracy score might not depend on the context width as expected, i.e. higher context width must generate better accuracy.

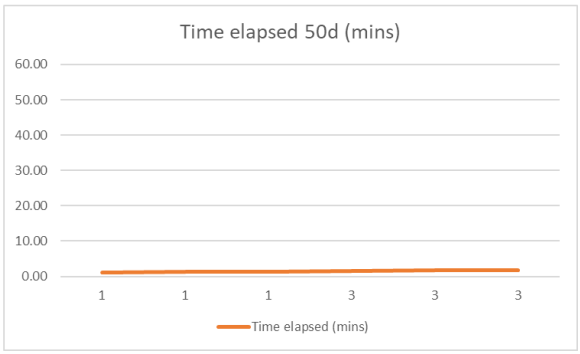
We see that the time taken is larger for LRC and smallest for MLP, and there is a clear distinction as the scale of LR classifier is in the range of 10 – 70 mins, whereas the classifier of MLP does the same task in less than 2 minutes.

Logistic Regression Classifier VS MLP Classifier

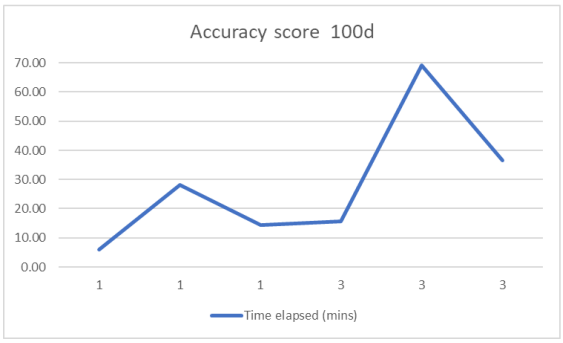
a. Total Time Elapsed

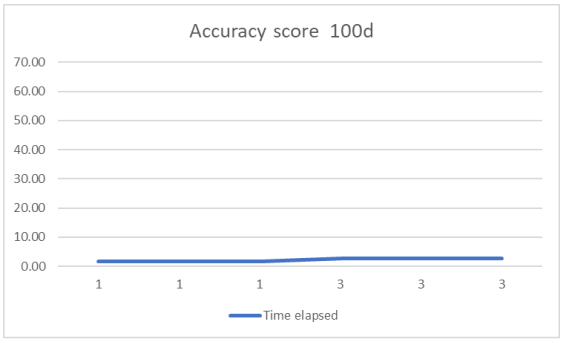
1. 50d VSM plotted with changing context width





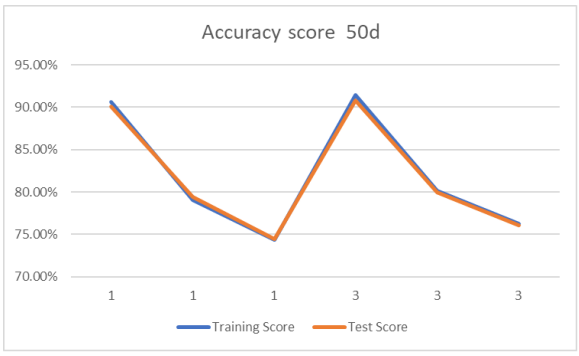
2. 100d VSM plotted with changing context width

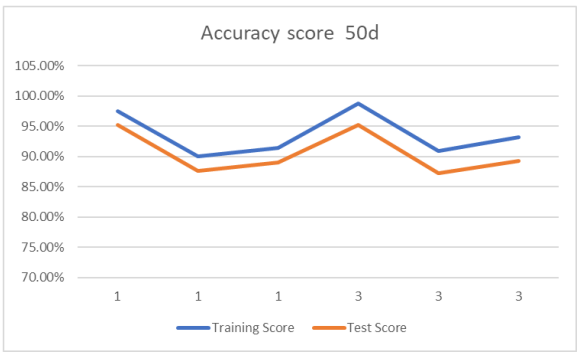




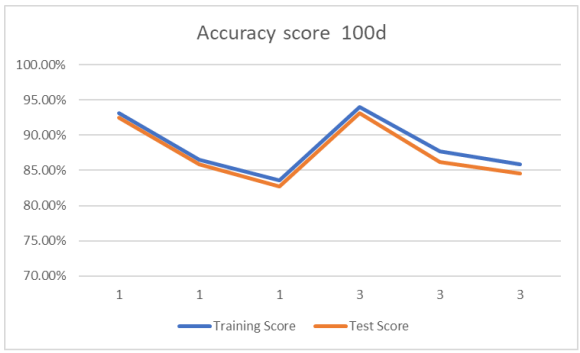
b. Accuracy score

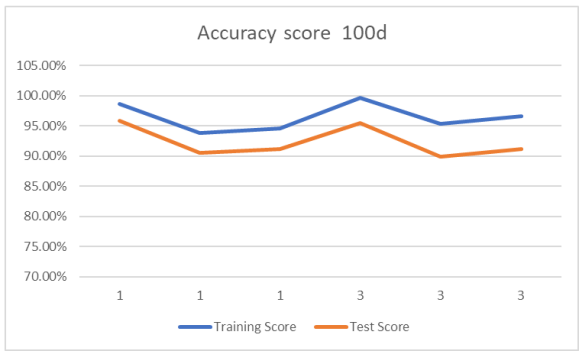
1. 50d VSM plotted changing context width





2. 100d VSM plotted changing context width





As the word embedding dimension increases, we see that the elapsed time for fitting and prediction increases by a factor of 10%. This is simple to visualize from the performance time graphs of 50d and 100d.

For the NER, POS, Chunking tasks, the time taken always peaks for POS task, as seen above. It always takes longer for chunking than NER.

The accuracy scores for NER is generally higher than the others. The difference between the accuracy scores for MLP and LRC are about 5 % with MLP having scores above 95%. The highest accuracy for LRC is just around 92%. The chunking task is having the lowest accuracy with LRC, since words are confused to be both a noun and verb.

Conclusion

After running these tests, and varying the previously mentioned properties and parameters, we conclude that the MLP Classifier is better. The MLP classifier outperforms the Logistic Regression classifier in every way. As the context length increases, the processing time increases. As the dimension of word embedding increases the accuracy and the time taken increases by a little fraction in the margin of 5% - 10%. The kind of hyperparameters we can work with are context length, word embeddings dimensions, and different algorithms to increase the performance. It is always advisable to try different methods before concluding which is the appropriate one. Hence, we did the same thing, we conclude that logistic regression classifier is not well suited for classification tasks, especially when compared to the multi-layer perceptron neural network. We verify from the results that a higher context width might make sense to us since it adds more meaning to each word, but the graphs prove that it may not be a good idea, or further testing is required. Further increasing the word embeddings seems to increase the accuracy, but the increase isn't quite evident in the margin of 5% only, whereas the running time increases by almost 10%. Logistic regression classifier is very slow compared to neural networks. This just goes to show that one algorithm can be way better for certain task than others. It might just also be that logistic regression is a very bad compared to neural networks in general. If you check the results, we can conclude that for natural language contexts, neural networks are far more better than logistic regression, which is more well suited for binary classification, whereas in our case we have too many classes. Hence, always be sure to experiment deeply before deciding on machine learning models.

References

[1] Tjong Kim Sang, E. F., & De Meulder, F. (2003, May). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4 (pp. 142-147). Association for Computational Linguistics.

[2] Karkaletsis, V., Paliouras, G., Petasis, G., Manousopoulou, N., & Spyropoulos, C. D. (1999). Named-Entity Recognition from Greek and English Texts. *Journal of Intelligent and Robotic Systems, 26*(2), 123-135. Retrieved 4 24, 2018, from http://dblp.uni-trier.de/db/journals/jirs/jirs26.html

[3] Aleksandr D., vsmlib, (2017), an open-source Python library for working with vector space models, http://vsmlib.readthedocs.io/en/latest/tutorial/basic.html#what-is-vsmlib

[4] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[5] Dostert, L. E. (1955). The Georgetown IBM experiment. 1955). Machine translation of languages. John Wiley & Sons, New York, 124-135.

[6] ALPAC, L. (1966). Machines: Computers in Translation and Linguistics, Report by the Automatic Language Processing Advisory Committee, Division of Behavioral Sciences. National Academy of Sciences, National Research Council, Publication, 1416.

[7] Leibniz, B., Boole, F., & Russell, T. History of AI.

[8] Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. Communications of the ACM, 9(1), 36-45.

[9] Roger C. Schank and Robert P. Abelson (1977). Scripts, plans, goals, and understanding: An inquiry into human knowledge structures

[10] Cullingford, R. (1986, June). Sam. In Readings in natural language processing (pp. 627-649). Morgan Kaufmann Publishers Inc.

[11] Chomsky, N. (1975). The logical structure of linguistic theory.

[12] Manning, C. D., & Schütze, H. (1999). Foundations of statistical natural language processing. MIT press.

[13] Kuhn, R., & De Mori, R. (1990). A cache-based natural language model for speech recognition. IEEE transactions on pattern analysis and machine intelligence, 12(6), 570-583.

[14] Christopher, M. B. (2016). PATTERN RECOGNITION AND MACHINE LEARNING. Springer-Verlag New York.

[15] Berger, A. L., Pietra, V. J. D., & Pietra, S. A. D. (1996). A maximum entropy approach to natural language processing. Computational linguistics, 22(1), 39-71.

[16] Multi-layer Perceptron, Neural network models (supervised), scikit learn, chapter 1.17.1., http://scikit-learn.org/stable/modules/neural_networks_supervised.html

[17] Salton, G. (1971). The SMART retrieval system—experiments in automatic document processing.

[18] Turney, P. D., & Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. Journal of artificial intelligence research, 37, 141-188.

[19] Logistic Regression, Linear models, scikit learn, chapter 1.1.11, http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

[20] F1 score, scikit learn metrics, http://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics

[21] Accuracy score, scikit learn metrics, http://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score

[22] SequenceLabelling_NLP_Tasks, Github Repository for our project https://github.com/SamarthaKV29/SequenceLabelling_NLP_Tasks/

In footnote at end of paper — list team-member roles and contributions to project, using individual team member initials!

TEAM ROLES

S K V

- Design and implement code
- Performing testing and documentation
- Generating results
- Analysis and reasoning

S S

- Coming up with ideas for further applications and research,
- Choosing context length and testing with different lengths,
- Generating the final report
- Generating the tables and graphs

L O

- Evaluating and choosing the correct model,
- Applying the concepts picked up in class to our project
- Checking the mathematical integrity
- Finding current state of the art
- Generate the tables and graphs.

Appendix A – In depth Background

One-vs-Rest Logistic Regression

The first algorithm we used is logistic regression, specifically, multinomial logistic regression. This type of regression is where the logistic regression is performed one-vs-rest classification, since simple logistic regression can only work with two classes. The scikit learn library provides us the implemented ready to use library from the sklearn.linear_model, which also performs the one-vs-rest modification for multi-class problems. There are many parameters we can set for the

As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Similarly, L1 regularized logistic regression solves the following optimization problem

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Fig 1 – Extract from scikit learn guide for logistic regression.

classifier, such as regularization penalty, solver, tolerance, multiclass type etc. When multi_class is set to multinomial, it becomes a pure multiclass logistic regression [14]. We have provided the minimizable cost function above.

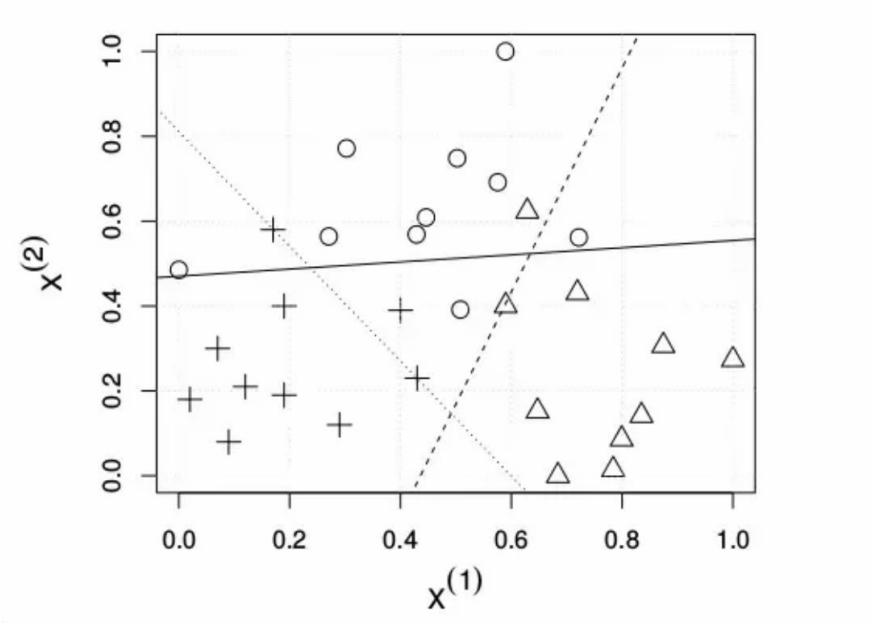


Fig 2 (a) – Multi binary classifier

Multilayer Perceptron Neural Network

The second method we use is the multi-layer perceptron neural network [16]. This is a supervised learning algorithm which is online (can learn by partial fitting the data) and can learn non-linear models. This model proved to be a more accurate and faster approach than linear models such as logistic regression. The figure below gives a more general approach,

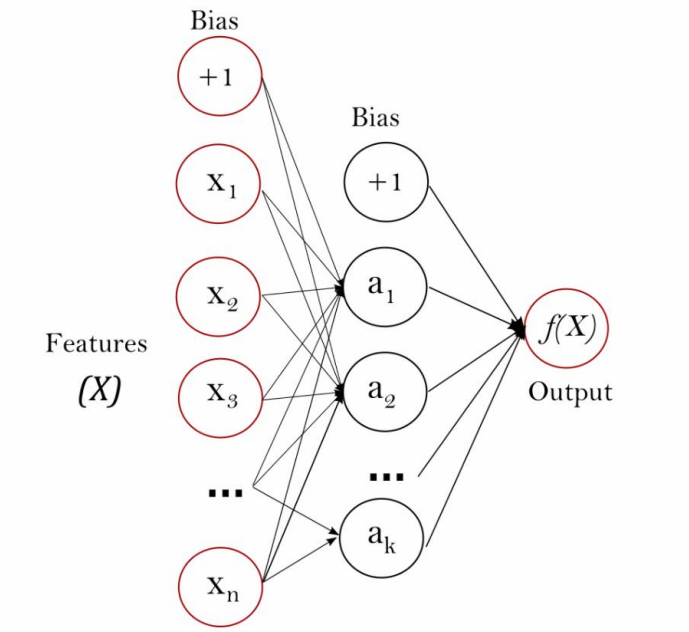


Fig 2 (b) – Extract from scikit learn guide for MLP classifiers, one hidden layer MLP with scalar output

The multilayer perceptron has leftmost layer as the input layer and the rightmost layer as the output layer. As the system trains on input data, it learns the coefficients for the hidden layers. It learns by back propagation and is a feed forward network. There is a non-linear activation function which transforms the values in each of the hidden layer neurons [16].

Both these algorithms have undergone much modification in terms of performance and accuracy over time. Each one is problem specific in that preferring to use one over the other really depends on what it can do for your problem. Easiest way to figure out which one to use by trialing different algorithms with the dataset and whichever produces the best accuracy over a large dataset and does it in certain fixed unit time of training is the one.

In our project, we could not use the standard available UCI repository datasets, as the kind of data we were looking for, annotated text data was unavailable. We have used the CONLL-2003 dataset as it was well suited for our purpose.

Word Embeddings:

Word Embeddings is the aggregate name for an arrangement of dialect modelling and highlight learning systems in characteristic dialect training in Natural Language Processing where words or expressions from the vocabulary are mapped to vectors of genuine numbers. Reasonably it includes a numerical implanting from a space with one measurement for every word to a consistent vector space with considerably higher measurement. Strategies to create this mapping

incorporate neural networks, dimensionality lessening on the word co-event matrix, probabilistic models, and unequivocal portrayal as far as the setting in which words appear. Word and expression embeddings, when utilized as the fundamental information portrayal, have been appeared to support the execution in NLP errands, for example, syntactic parsing and sentiment analysis.

We have acquired our embeddings from the highly popular Stanford GloVe [4], which stands for Global Vectors for Word Representation. The GloVe algorithm itself is an unsupervised algorithm for learning the vector representations for various words. These pre-trained vectors are very important to help our system build understanding on the meanings of words and sentences. It helps our system form the specific benchmark tasks such as NER, POS and chunking.

System	Accuracy
Shen et al. (2007)	97.33%
Toutanova et al. (2003)	97.24%
Giménez and Màrquez (2004)	97.16%

(a) POS

System	F1
Shen and Sarkar (2005)	95.23%
Sha and Pereira (2003)	94.29%
Kudo and Matsumoto (2001)	93.91%

(b) CHUNK

System	F1
Ando and Zhang (2005)	89.31%
Florian et al. (2003)	88.76%
Kudo and Matsumoto (2001)	88.31%

(c) NER

Fig 3 - State of the art systems on three NLP tasks, An F1 Score is calculated for CHUNK and NER and performance is reported with per-word accuracy in the POS.

Definition of tasks:

- Part of Speech Tagging:

  In POS we label each word with a unique tag that indicate its syntactical role, for example, noun, pronoun, verb, etc. A standard benchmark setup is described by Toutanova et al. (2003). Section 0-18 of Wall Street Journal (WSJ) data are used for training, while sections 19-21 are for validation and section 22-24 are for testing.

- Chunking:

  Chucking is used to label of the sentence with labels such as noun or verb phrases (NP or VP). Every word has only one tag, which is coded as begin-chunk (e.g., B-NP) or inside-chunk tag (e.g., I-NP).

- Named Entity Recognition:

  NER marks nuclear components in the sentence into classes, for example, "PERSON" or "LOCATION". As in the chunking undertaking, each word is relegated a tag prepended by a pointer of the start or within an element.

VSMlib in python 3.6

- Vector space models (VSMs)

Vector space models are dimensions created from words in documents depending on their occurrence. If the word occurs in the document, its vector will hold a non-zero value. There are mainly three broad classes of VSMs, of which we have used the word-context VSMs. VSMs have been around since the early 1970s, used in SMART information retrieval systems [17]. The general idea of VSMs was to represent a word in a document as a vector in a vector space. The points which end up close to each other have similarities in meaning and this can be used to process new sentences and text [18]. There are more details to include, however, we shall not go into the implementation and other details.

VSMlib for python allows us to use the pre-trained models from external sources in our project much more easily than previously achieved. The author describes it as an open source library for working with VSMs and word embeddings such as word2vec [3]. It includes a lot of helpful features while we use it map our dataset words to preexisting words in the vector. We use two measures here to compare the dataset vocabulary to the word vector vocabulary, which are:

1. Out of Vocabulary Rate – Generally described as the number of unknown words in a new sample of language, expressed as a percentage.
2. Vocabulary Cover Rate – Number of words required to cover the entire language corpus under consideration, again expressed as a percentage.

So, the library provides multiple cool features that we use to process the dataset and use the model to prepare our input data.