Core Relation Extraction from Natural Language

Anuroop John Abraham
ajohnabr@ucsc.edu

1 Introduction

The problem statement is to determine knowledge graph relations in Freebase schema, which are invoked in user utterances addressed to a conversational system using conventional machine learning models. Since both the user utterance and its corresponding Core Relations are given, the problem falls under supervised machine learning approach.The goal is to identify the core relationships of each utterance given by user.

2 Datasets

Since all of the relevant files where in text format, used pandas(Mckinney, 2011) library to do the data manipulation. Data from train_data.csv and test_data.csv where imported as a dataframe. The train dataset has contains 2253 user utterances and its corresponding core relations and the test data contains 981 user utterances. Each user utterance in train data is associated with one or more number of core relations. While framing the machine learning approach for this problem, "utterances",can be considered as the input and "Core relations" as output respectively.
E.g, for the utterance "who plays luke on star wars new hope" the Core relation associated to it are "movie.starring.actor" and "movie.starring.character".
The total number of unique words in the set of input utterances is 1117. In total there are 19 unique output labels and the distribution of them are given below

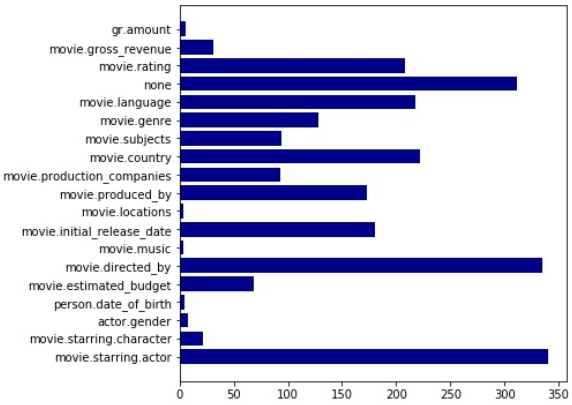

Figure 1: Frequency Distribution of Core Relations

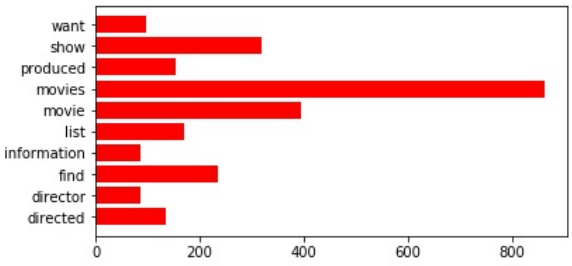

Figure 2: Frequency Distribution of Most Common Words

3 Models

3.1 Data Preprocessing

The utterances had several stop words. So in-order to remove them used NLTK(Loper and Bird, 2002) stopword corpus and removed the common words from utterance dataset and the stop word corpus. In order to reduce the number of features and to get better word mappings to the output label, each word was lematized. NLTK(Loper and Bird, 2002) builtin WordNetLemmatizer is used to get the lematized words.

3.2 Feature Extraction

Every machine learning model expects its input to be in numeric format. In order to convert the input text to vector representation, count vectorizer and tfidf vectorizer is used. A vocabulary of unique words are created at first then, its sorted and each of these words are given an index. Then for each documents in the dataframe, the words in sentences are replaced by a 1-dimensional matrix of length which

is equal to the total number words of the vocabulary. In the given problem, vocabulary size is 1117, hence each sentence in train dataset will be converted into a matrix of length 1117. For eg: for the sentence, “who plays luke on star wars new hope”, the index of each word in the vocabulary would be‘who’:1085, ‘plays’:735, ‘luke’:586, ‘on’:679, ‘star’:927, ‘wars’:1063, ‘new’:660, ‘hope’:458. So a sparse 1-D vector will be generated with 1 on the above mentioned indexes and 0’s on rest of the positions. Tfidf calculates, the term frequency-inverse document frequency of the words in each sentences. Hence the weight-age of each word can be computed. This is another way of understanding the magnitude of influence of each word in a sentence to its corresponding label.

Words	CountVectorizer Index	Tf-idf
who	1085	0.373
plays	735	0.507
luke	586	0.342
on	679	0.265
star	927	0.426
wars	1063	0.303
new	660	0.324
hope	458	0.190

3.3 Models

Deep Learning is vast domain enriched with different algorithms. Finding the algorithm, that best suits the problem is the most important part. For the given problem different deep learning approaches are used to get the best results. Few of those are

1. Multi-layer Perceptron Neural Network

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The different types of Perceptron networks are kinds of perceptron models:

1. Single-layered perceptron model.
2. Multi-layered perceptron model.

A single-layer perceptron model includes a feed-forward network depends on a threshold transfer function in its model. It is the easiest type of artificial neural network that able to analyze only linearly separable objects with binary outcomes(target) i.e. 1, and 0.

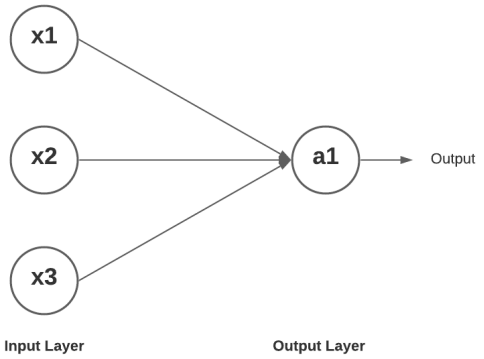

Figure 3: Single-layer Perceptron

A multi-layered perceptron model has a structure similar to a single-layered perceptron model with more number of hidden layers. It is also termed as a Backpropagation algorithm. It executes in two stages; the forward stage and the backward stages.

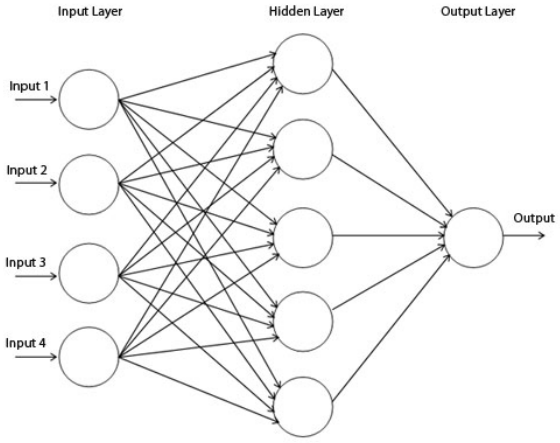

Figure 4: Multi-layer Perceptron

2. Convolutional Neural Network

The convolutional neural network, or CNN for short, is a specialized type of neural network model designed for working with two-dimensional image data, although they can be used with one-dimensional and three-dimensional data.Central to the convolutional neural network is the convolutional layer that gives the network its name. This layer performs an operation called a “convolution“. In the context of a convolutional neural network, a

convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

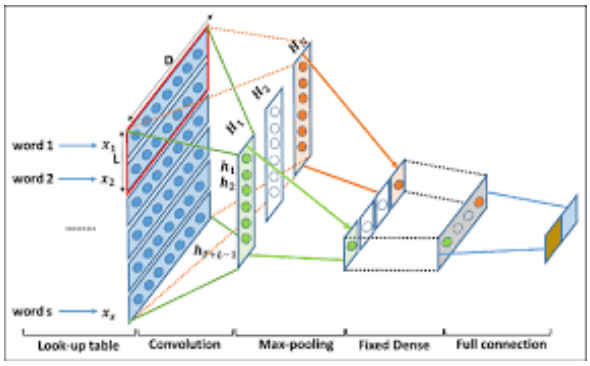

Figure 5: Convolutional Neural Network

3. Recurrent Neural Network

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence.

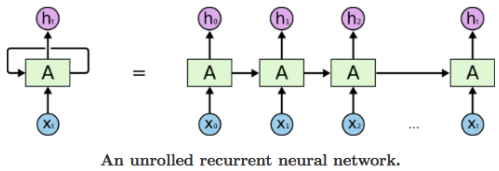

Figure 6: Recurrent Neural Network

4. Long Short-Term Memory

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells. These operations are used to allow the LSTM to keep or forget information. Now looking at these operations can get a little overwhelming so we'll go over this step by step. The core concept of LSTM's are the cell state, and it's various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. Gates contains sigmoid activations. A sigmoid activation is similar to the tanh activation. Instead of squishing values between -1 and 1, it squishes values between 0 and 1.

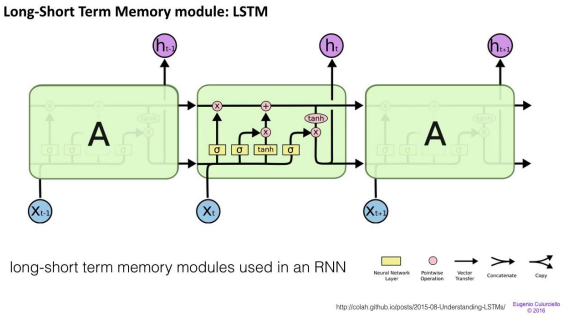

Figure 7: LSTM

5. Gated Recurrent Unit

GRU or Gated recurrent unit is an advancement of the standard RNN i.e recurrent neural network. GRUs are very similar to Long Short Term Memory(LSTM).Just like LSTM, GRU uses gates to control the flow of information. They are relatively new as compared to LSTM. This is the reason they offer some improvement over LSTM and have simpler architecture. Unlike LSTM, it does not have a separate cell state (Ct). It only has a hidden state(Ht). Due to the simpler architecture, GRUs are faster to train. There are primarily two gates in a GRU as opposed to three gates in an LSTM cell. The first gate is the Reset gate and the other one is the update gate.

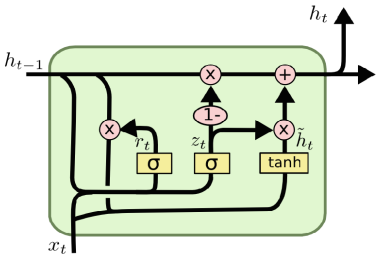

Figure 8: GRU

4 Experiments

The data-set was split into train and validation-set with 80:20 ratio. Hence the training dataset contains 1802 data samples for training and 451 data samples for testing the accuracy of the model. In-order to get few sample data of different category in both training and validation, the dataset is shuffled before splitting. Used Scikit-learn(Pedregosa et al., 2011) 'train-test-split' function to split the data. Random state was enabled, since the every time function train-test-split shuffles the data and gives different set of outputs.

4.1 Hyperparameter Tuning

[A]. Single-layer Perceptron with bag of words as feature extractor

For implementing this model, the prepossessed inputs were converted into feature vectors by bag of words approach. In bag of words, the a vocabulary dictionary of all words in the training data was created. Then for every sentence in the training corpus, the words were replaced with its corresponding index from the vocab dictionary. The labels were converted into numeric form using a custom built function. Then the feature vectors with padding was fed into a single-layered perceptron neural network. Tried different values for the parameters such as learning rate, batch size, epoch, optimizer function etc.

learning rate	epoch	batch size	val acc	val loss
0.001	100	16	0.866	0.666
0.001	50	16	0.85	0.677
0.0001	100	16	0.824	0.78
0.001	25	4	0.862	0.0.45

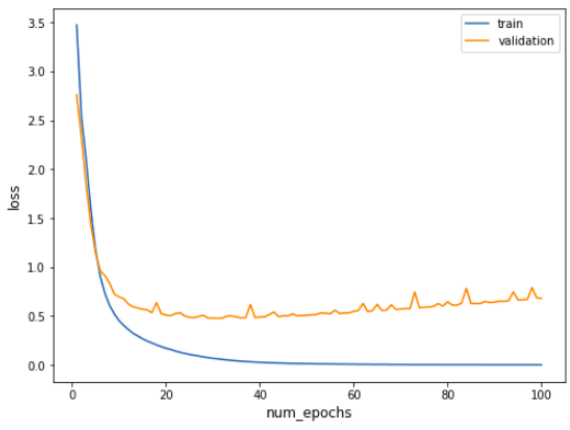

Figure 9: Single Layer perceptron - Loss

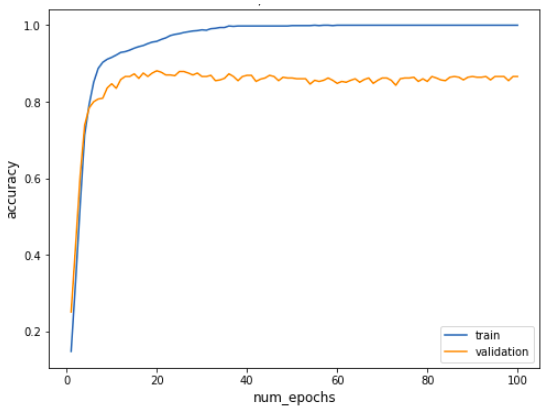

Figure 10: Single Layer perceptron - Accuracy

[B]. Single-layer Perceptron with tfidf as feature extractor

Input utterences were converted into word feature vectors using sckit-learn's tfidf. Used scikit-learn's label-binarizer to convert labels into corresponding numeric representation. Then the tfidf feature vectors were fed into a single-layered perceptron neural network. Used pytorch's cross entropy with sigmoid as the loss function since the problem demands multiple neurons in the final layer. Sigmoid function is better suited for multi-class problems. Tried different values for the parameters such as learning rate, batch size, epoch, optimizer function etc.

learning rate	epoch	batch size	val acc	val loss
0.001	100	16	0.864	0.633
0.001	25	16	0.864	0.521
0.0001	100	16	0.877	0.68
0.001	15	4	0.891	0.556

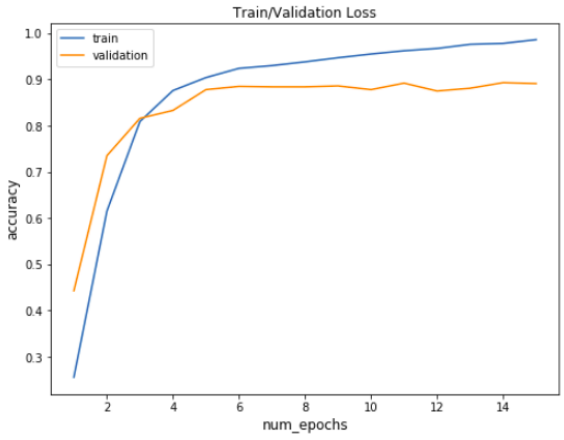

Figure 11: Single-Layer perceptron - tfidf - Loss

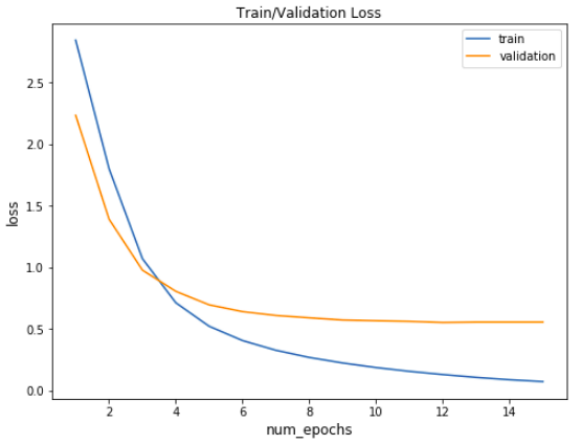

Figure 12: Single-Layer perceptron -tfidf- Accuracy

[C]. Multi-layer Perceptron with tfidf as feature extractor

Since tfidf veectorizer was giving better accuracy compared to normal bag of words approach, I plan to keep it as my feature extractor for the rest of experiments. Input utterences were converted into word feature vectors using sckit-learn's tfidf. Used scikit-learn's label-binarizer to convert labels into corresponding numeric representation. Then the tfidf feature vectors were fed into a multi-layered perceptron neural network.

no hidden layers	lr	epoch	batch size	val acc	val loss
5	0.001	25	4	0.80	2.01
4	0.001	100	16	0.78	2.24
4	0.0001	100	4	0.82	1.96
2	0.001	15	4	0.81	2.06

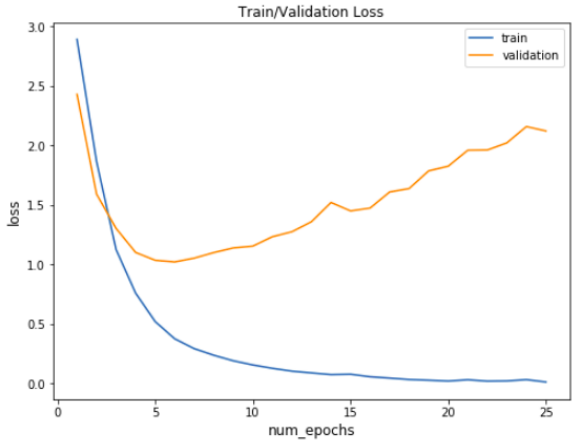

Figure 13: Multi-Layer perceptron - tfidf - Loss

Figure 14: Multi-Layer perceptron -tfidf- Accuracy

[D]. CNN with Glove Embedding

In-order to find out, whether the context between the words in an utterence can influence the models performance, used an RNN based neural network. Used bag of words as the feature extractor for converting words into its corresponding numerical form. Since RNN works on timestep basis each utterence was tokenized before passing into the embedding layer. used 100 diamentional embedding so that each word vector will be represented in 100 dimensional space. Then each of these 100 diamentional word vectors were fed into the RNN cell at each time step.

lr	epoch	batch size	val acc	val loss
0.001	100	16	0.88	0.76
0.001	100	4	0.82	0.87
0.0001	100	4	0.83	0.54
0.001	60	16	0.86	0.43

[E]. RNN with bag of words

In-order to find out, whether the context between the words in an utterence can influence the models performance, used an RNN based neural network. Used bag of words as the feature extractor for converting words into its corresponding numerical form. Since RNN works on timestep basis each utterence was tokenized before passing into the embedding layer. used 100 diamentional embedding so that each word vector will be represented in 100 dimensional space. Then each of these 100 diamentional word vectors were fed into the RNN cell at each time step.

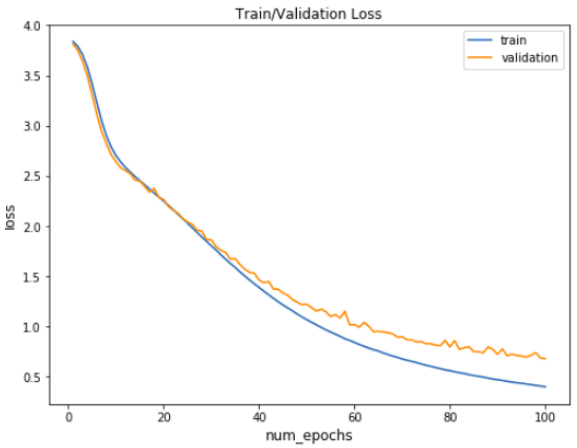

Figure 15: cnn- glove - Loss

Figure 16: cnn - glove- Accuracy

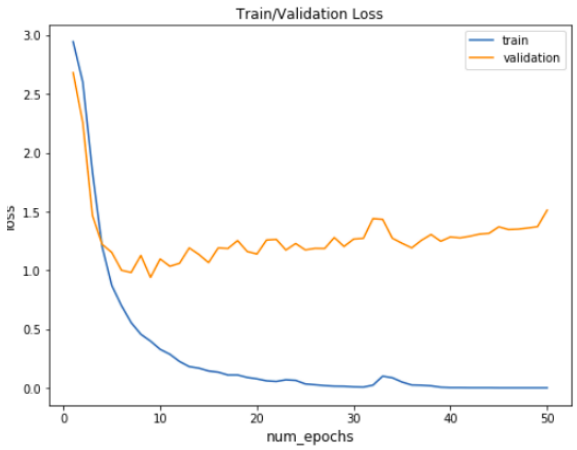

Figure 17: rnn - bag of words - Loss

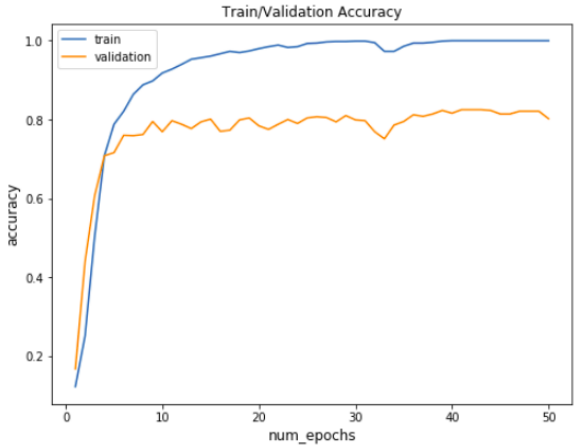

Figure 18: rnn - bag of words - Accuracy

4.2 Modelling approach

[A]. Multiclass-single Label approach

In this approach multiple labels of each sentences were computed as a single label. For eg: The label for the sentence "who plays luke on star wars new hope" - ['movie.starring.actor', 'movie.starring.character'] is treated as a single label, ie 'movie.starring.actor movie.starring.character'. Hence the total number of labels in the training data is now 47.

[B]. Multiclass-multilabel approrach

In this approach all the original 19 labels were treated as the outputs. In case of OneVsRest, for each classifier, the class is fitted against all the other classes, that means that problem of multiclass/multilabel classification is broken down to multiple binary classification problems.

[C]. Combination of multiclass-multilabel approach

Combined both the single-label and multi-label approaches to get better accuracy and generalizations. First all the inputs are passed through the multi-class multi-label model and then then the null predictions in which the model is unable to predict any values were replaced by the predictions from the single-label model.

The performance of each model was evaluated based on the results of confusion matrix, classification report and validation accuracy score. Each row of the confusion matrix represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa. Classification report however generates the precision, recall, f1score of each model.

5 Results

From the experiments, its evident that all the deep leaning neural networks under-performed compared to the shallow perceptron due to the lack

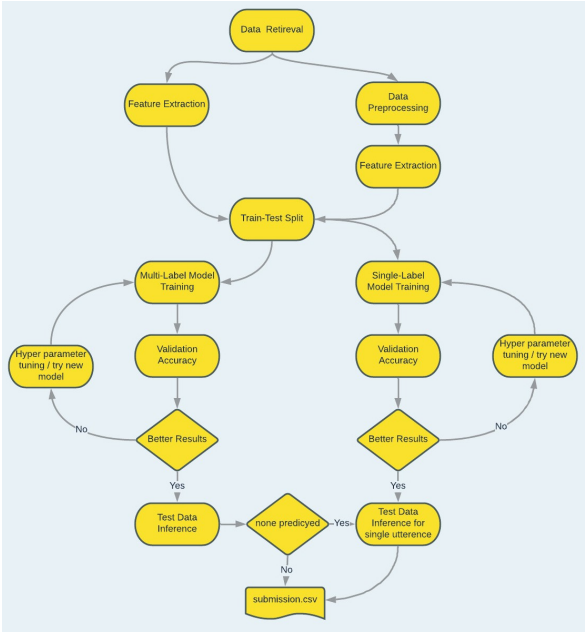

Figure 19: combination of single-label and multi-label

of sufficient amount of data.In addition to that both the single label and the multi-label approach has its on pros and cons. The single label approach gave a validation accuracy of 85% percentage, however its not always guaranteed that the same combination of labels will appear in the test data. Considering the outliers in real world scenario the total number combinations which can be formed by the labels are 19!, out of which only 47 combinations are present in the training data. Hence for the generalization purpose multilabel is best suited. The disadvantage of multi-label classifier is that, due data imbalance the model performance is very poor. For many instance the model didn't predict any labels. Hence to get the best out of both models, an approach is created with the combination of both. First the data will be passed through the multi-label model. Then the null predicted outputs were passes through the single label model to get its prediction.

Models	**Train**	**Validation**	**Test**
Single Label	0.96	0.89	0.75
Multi Label	0.97	0.81	0.72
Combined	0.97	0.89	0.76

References

Edward Loper and Steven Bird. 2002. Nltk: the natural language toolkit. *CoRR*, cs.CL/0205028.

Wes Mckinney. 2011. pandas: a foundational python library for data analysis and statistics. *Python High Performance Science Computer*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830.