MSc Natural Language Processing

M1 SUPERVISED PROJECT

Learning how to estimate the quality of Syntactic Parses

Supervised By:

Yannick Parmentier
Guillaume Toussaint

by

Basar Ezgi
Bouziani Meryem
Naava Namuwonge Hedwig
Yasmine Hanady

May 2024

Abstract

Dependency parsing is the process of encoding syntactic information by identifying dependency relations between words in a sentence. While current state-of-the-art dependency parsers exhibit high levels of accuracy on data similar to their training data, their performance suffers when tested on data out of their domain. There is reason to question the reliability of the syntactic annotations produced by these parsers since it is likely that a new input sentence may not fall under the domain of a given parser. This project proposes an automated way of assessing the quality of syntactic annotations. The approach taken is to train various classifiers that can evaluate syntactic annotations. Since the classifiers are trained on the output of parsers, they are intended to act as an outer layer of quality check. The trained models only achieve up to 66.15% accuracy and leave room for improvement. Additional or alternative strategies are discussed as an extension of this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Syntactic parsing refers to the process of identifying relations between lexical units in a sentence and creating graph or tree structures which captures these relations. It is a crucial task in natural language processing with applications in many areas ranging from information extraction to machine translation. (Lease et al., 2006) It is also useful for carrying out research in linguistics given that syntactic parsing algorithms have made it possible to automatically generate annotations for a large number of sentences and construct syntactically annotated treebanks and corpora which aid further linguistic analysis.

While there are many approaches to the task of syntactic parsing, constituency parsing and dependency parsing are the two main methods of syntactic parsing. This project concerns itself exclusively with the dependency parsing approach. One of the main advantages of dependency parsing over constituency parsing is that dependency structures lend themselves more easily to linear annotations. Constituent parsing generates hierarchical structures that operate on phrases whereas dependency parsing simply establishes connections between words using dependency labels. Researchers attribute the greater popularity of dependency parsing in NLP research to its conciseness and ease of annotation. (Gershenson, 2003) Being able to represent syntactic structures linearly with relative ease has proved important for this project as it faces the challenge of dealing with highly nested data structures. Another advantage of dependency parsing is that head-dependent relations allow for a more transparent way of encoding semantic relations between predicates and arguments. (Kübler et al., 2009) Such approximation makes dependency parsing preferable for downstream tasks.

As illustrated in Figure 1.1, dependency trees feature binary relations between heads and dependents and this relation comes in the form of directed and labeled arcs. Each token depends on another head token with the exception of a special token termed the root node. Every dependency tree contains only one root node with no incoming edges from other nodes. The edges are labeled to highlight the type of relation between a head and a dependent.

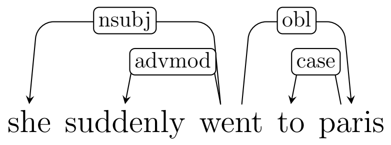


Figure 1.1: An example dependency tree (Nivre et al., 2020)

The most commonly adopted labeling scheme for dependency parsing is the Universal Dependencies (UD) scheme. (Marneffe et al., 2021) It is a framework that has made it possible to provide consistent annotations for a large number of natural languages and create morpho-syntactically annotated treebanks. The scope of this project is limited to English and it exclusively operates on English language data. A multilingual version of this project would also have been possible to realize, albeit proving to be a significantly bigger challenge as it would have to accommodate the typological differences between cross-linguistically different languages. Therefore, the number of different dependency labels is also limited to the subset of unique dependency relations found in UD English treebanks as opposed to all the possible dependency relations cross-linguistically.

The goal of this project is to implement a syntactic annotation reliability scorer that can estimate the reliability of automatically generated syntactic parses. Despite significant advances in automated parsing systems, ensuring the reliability of syntactic annotations remains an unsolved task.

Model	**LAS**	**MLAS**	**BLEX**
Stanford (Qi et al., 2018)	74.16	62.08	65.28
UDPipe Future (Straka, 2018)	73.11	61.25	64.49
HIT-SCIR (Che et al., 2018)	75.84	59.78	65.33
TurkuNLP (Kanerva et al., 2018)	73.28	60.99	66.09

Table 1.1: Dependency parsing results on the UD benchmark (Ruder, 2022)

Table 1.1 reports state-of-the-art dependency parsing results using LAS, MLAS, and BLEX evaluation metrics where LAS stands for the proportion of correct head-dependent relations with the appropriate label; MLAS is the same metric as LAS with the additional requirement of certain morphological features being correctly predicted; and BLEX is the proportion of correct head-dependent pairs with the appropriate lemmas. (Zeman et al., 2018) These results support the claim that even the best dependency parsing algorithms available are not entirely reliable and the primary goal of this project is to provide an additional layer of quality check for automated parsers.

Being able to assess the reliability of a syntactic annotation provides benefits not only for syntactic parsing tasks but also for all downstream NLP tasks utilizing the generated syntactic annotations. There are many aspects of language which provide challenges for automated parsing systems. It can be argued that the main challenge of automated parsing systems is out-of-domain parsing. Out-of-domain parsing refers to the analysis of data that substantially differs from the data found in the training set used to develop the parsing model. This difference can manifest itself in a number of ways. Out-of-domain parses can be the result of parsing text that exhibits variations in topic, genre, vocabulary, register, or modality.

	Domain A (Answers)	**Domain B (Newsgroups)**	**Domain C (Reviews)**	**Domain D (WSJ)**
ZhangNivre	76.60	81.62	78.10	89.37
UPenn	68.54	74.41	70.17	81.74
UMass	72.51	77.23	74.89	81.15
NAIST	73.54	79.83	75.72	87.95
IMS-2	74.43	79.63	76.55	86.88
IMS-3	75.90	79.77	77.61	86.02
IMS-1	78.33	83.16	79.02	90.82
CPH-Trento	78.12	82.90	79.58	90.47
Stanford-2	77.50	83.56	79.70	89.87
HIT-Baseline	80.75	85.26	81.60	91.88
HIT-Domain	80.79	85.18	81.92	91.82
Stanford-1	81.01	85.85	82.54	91.50
DCU-Paris13	81.15	85.38	83.86	89.67

Table 1.2: LAS scores reported by dependency parsing teams at the 2012 Shared Task on Parsing the Web (Petrov et al., 2012)

As demonstrated in Table 1.2, dependency parsers can show up to a 14.41% difference in performance when applied to data from different domains. The presence of such differences can pose obstacles in the way of generating reliable parses for unseen data. For example, data from web forums will be out of the domain of a syntactic parser trained on data from newspapers even if the data in question is about the same topics or events. Web forums use a more informal language whereas newspapers have a different register lending to the use of a more formal language. In the same vein, data from romance novels will be out of the domain of a parser trained on scientific articles due to register, vocabulary, and topic differences. Differences in modality, such as between written and spoken language, will also pose similar issues for syntactic parsing models. Moreover, inconsistencies in annotation guidelines between the training data and the data to be parsed can lead to significantly different parses and undermine the performance of a parsing system.

Variations in natural language data call for an investigation into methodologies for generating reliability scores for syntactic annotations. This project aims to provide a tool to automatically assess the trustworthiness of annotations performed on text containing such variations. The approach taken is to treat the task of reliability assessment as a classification task. Selected parsers are used to obtain syntactic annotations for a given set of sentences from various sources. These syntactic annotations are compared against the gold-standard in order to be tagged as right or wrong. In the end, several classifiers are trained on a dataset featuring right and wrong classes. These classifiers serve as an outer layer to automated parsing algorithms with the end goal of making reliability assessments for new annotations. The source code for the project is publicly available on a dedicated GitHub repository.[1]

[1] https://github.com/Mery-e/SyntacticParsingProject

Chapter 2

Experimental setup

2.1 Corpus

Universal Dependency (UD) [1] English is the combination of several treebanks, each introduced by a different team at a different time using different conversion tool. In the implementation of the project 6 UD treebanks were used but only the first 3 listed below formed the data and were fed to the machine learning algorithms. The used treebanks are the following: English-GENTLE, EnglishLinES, EnglishParTUT, English-GUM, English-EWT, and English-PUD.

- The English-GENTLE treebank originates from the Genre Tests for Linguistic Evaluation (GENTLE) Corpus, which serves as a repository for linguistic assessment. This corpus comprises various text genres, each meticulously annotated across multiple layers, mirroring the annotations present in the GUM corpus.

- UD EnglishLinES, constituting the English portion of the LinES Parallel Treebank, encompasses various text sources, predominantly drawn from literature, alongside segments from online manuals and Europarl data.

- TUDEnglish-ParTUT is an adaptation of a multilingual parallel treebank originating from the University of Turin. This treebank encompasses diverse text genres, ranging from talks and legal texts to Wikipedia articles and contains parallel collection of Italian, French, and English sentences.

- The English-GUM, sourced from the Georgetown University Multilayer corpus (GUM), contains diverse text types and is continually expanded through student contributions as part of a computational corpus linguistics course at Georgetown University.

[1]See https://universaldependencies.org/en/index.html .

- The English-EWT corpus is a Gold Standard Universal Dependencies Corpus that comprises words and sentences sourced from various genres of web media, including weblogs, newsgroups, emails, reviews, and Yahoo!

- The English-PUD dataset is a component of the Parallel Universal Dependencies (PUD) treebanks, crafted for the CoNLL 2017 shared task on Multilingual Parsing from Raw Text to Universal Dependencies. The sentences are drawn from news articles and Wikipedia entries and the dataset encompasses translations from German, French, Italian, or Spanish via English, with annotations conducted by Google according to universal annotation guidelines

These treebanks employed the CoNLL-X standard, which has been termed CoNLL-U [2] for annotations. ConLL-U is an annotation schema for describing linguistics features across diverse languages and for compiling these descriptions into linguistics corpora. The structure comprises three line types: word lines, blank lines indicating the end of the sentence, and comments. Word lines encapsulate annotations for each token in the sentence described into 10 fields listed below separated by individual tab characters.

1. **ID**: word's index, which starts from 1 for each new sentence.
2. **FORM**: denotes the word form or punctuation symbol.
3. **LEMMA**: indicates the word's lemma or stem which is the base form of the word.
4. **UPOS**: the universal part-of-speech tag which represents the universal representation of the word category.
5. **XPOS**: the expanded part-of-speech is the optional language-specific or treebank-specific part-of-speech, marked with an underscore if unavailable.
6. **FEATS**: list of morphological features, either from the universal feature inventory or a defined language-specific extension, also denoted by an underscore if absent.
7. **HEAD**: represents the head of the current word, denoted by a value of ID or zero.
8. **DEPREL**: indicates the universal dependency relation to the HEAD, with root signifying HEAD = 0, or a defined language-specific subtype.
9. **DEPS**: denotes the enhanced dependency graph, presented as a list of HEAD-DEPREL pairs.
10. **MISC**: a field that captures any additional annotations.

[2]https://universaldependencies.org/format.htm .

The initial step in the project was to extract linguistic data from the CoNLL-U formatted files and then this data will be processed to isolate the raw text content of each of these files, resulting in a list of sentences transformed into strings with tokens separated by spaces and written to new text files.

2.1.1 Stanza

Stanza [3] is a set of precise and effective tools for linguistic analysis and offers the most recent NLP model of several human languages and it was used in this project to establish a processing pipeline tailored to English language tasks. This pipeline configuration, includes the following linguistic processors: tokenization, multi-word tokenization, part-of-speech tagging, lemmatization, and dependency parsing. The processing of pre-tokenized input was also enabled, enhancing efficiency and flexibility in the NLP workflows. Then, the text files containing linguistic data were read and processed and the NLP-Stanza pipeline was applied to these text data, generating annotations such as tokenization, multi-word tokenization, part-of-speech tagging, lemmatization, and dependency parsing.
And finally, the annotations generated by the NLP-Stanza pipeline for the text data files were converted into the CoNLL-U format and these files were exported.

2.1.2 Trankit

The same steps were done with trankit [4] parser for English language. Trankit stands as a lightweight Transformer-based Python Toolkit designed for multilingual Natural Language Processing (NLP). With 90 pre-trained pipelines adapted to 56 languages, it provides a flexible pipeline that allows training for common NLP tasks in over 100 languages. Trankit outperforms other models in dependency parsing, part-of-speech tagging, sentence segmentation, and morphological feature tagging by utilizing advanced pre-trained language models.

Then, in this next step, data was parsed from CoNLL-U [5] formatted files for comparison within the project. Three files, the original CoNLL-U, stanza-parsed CoNLL-U, and trankit-parsed CoNLL-U for each treebank will be used. This step started by creating a list for each of these: gold (from the original), stanza, and trankit. Each file's content is read and parsed line by line, excluding comments. Empty lines signal the end of a sentence, prompting the creation of new list elements within the lists if the preceding

[3]See https://stanfordnlp.github.io/stanza/ .
[4]See https://trankit.readthedocs.io/en/latest/ .
[5]See https://universaldependencies.org/format.html .

element isn't empty. For non-empty lines, specific fields are extracted and appended to the current list element. Finally, the last list element is removed to ensure proper formatting of the parsed data. This process prepares the data from all three files for comparison within the project.

Finally, data between the gold standard and the parsed using either the Stanza or Trankit libraries were compared. And in this comparison, only 3 datasets were used: gentle-gold, lines-gold, and partut-gold. And for each dataset, if there's a discrepancy between the data in gold and the data parsed by either stanza or Trankit, the parsed data will be considered wrong and will be appended to the wrong data list. As a result, 3602 sentences were classified as wrong and 2460 as right. This data will serve as the updated gold standard for the project. A dedicated column in the final dataset will indicate whether each entry was parsed correctly (1) or incorrectly (0) with other linguistic features.

Dependency parsing generates nested structures wherein the representation of each sentence is essentially a list of dependency relations for each token. With the goal of building a classifier to assess the reliability of the generated parses at the sentence level, such nested structures have to be linearized and transformed into fixed-length vectors. The approach taken is an adapted version of the bag of words approach to text representation. (Harris, 1954) The bag of words method refers to a strategy of generating representations for raw text in which every sentence or document is represented by token frequencies over a set vocabulary. Since the data in question is not simply a list of raw sentences but rather a list of lists of dependency annotations, the vector for each sentence had to contain information that goes beyond simple term frequencies for each token. It can be argued that the task of capturing dependency relations linearly presents a challenge for feature design and various strategies to that end have been entertained. A subset of the final dataset that has been used to build the classifiers can be found below:

ADJ	ADP	ADV	AUX	CCONJ	DET	INTJ	NOUN	NUM	PART	...	X:punct:NOUN	X:root:root_pos
0	0	1	1	0	1	0	4	1	1	0	0	0
0	0	1	0	0	0	0	0	0	2	0	0	0
0	1	1	1	0	0	0	1	0	0	0	0	0
1	1	4	1	1	1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0	0	0

Table 2.1: An example of how sentence annotations are represented in the data

As exemplified in Table 2.1, the feature vector contains a fixed set of features intended to capture dependency relations in a linear fashion. The resulting vector can be said to be highly sparse with a size of 1280 features. This feature vector contains three different categories of features. The first of these is the set of unique POS tags which amounts to 17 different POS tags. Rather than counting the frequency of each token, the dataset instead represents the count of the POS tags associated with the tokens. The second category is a relation between the POS tag of a token and the POS tag of the head token with which it has been associated. This category features 220 unique attachments between a dependent and a head POS tag. The third one is the same relation as the second category, except it captures the particular kind of dependency relation by including the dependency relation label as a part of the feature. The intuition behind this approach is the observation that two tokens can be correctly labeled in terms of POS tags and attach the dependent with the appropriate head token and, yet, label this dependency relation in a number of different ways which yields significantly different parses. The size of this final category is 1043. One observation that could be made is that the feature vector exclusively contains the POS tags of the tokens rather than containing features directly for the tokens. The reason for this abstraction is that having features for tokens leads to a substantially sparser dataset. A significant drop in results have been observed in experimental settings where token features have been used instead of or in conjunction with POS features.

2.2 Evaluation Metrics

The results of our experiments are presented with multiple classifiers, including Logistic Regression (LR), Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) for the classification task. Each classifier was trained and evaluated using the same dataset:

Evaluation Metrics: Accuracy, precision, recall, and F1-score were used to evaluate each classifier's overall performance.

In addition, visualizations such as confusion matrices and ROC curves were done for each classifier to have a better understanding of its behavior. Our goal is to select the most successful model for the classification problem at hand. Defining each metric can help us have a better understanding of how to assess classifier performance.

- *Accuracy*: This metric measures the number of correctly predicted values divided by the total number of predicted values. (Badillo et al., 2020) The accuracy score is computed as follows:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{True Negatives} + \text{False Positives} + \text{False Negatives}}$$

- *False Positive Rate*: False Positive Rate (FPR) is related to precision. (Joshi, 2020) It measures the proportion of negative values predicted incorrectly. (Badillo et al., 2020) The formula for FPR is as follows:

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

- *False Negative Rate*: False negative rate (FNR) has same definition as recall (Joshi, 2020), it indicates the proportion of instances that are incorrectly classified as negative among the instances that are actually positive. (Badillo et al., 2020) The formula for FNR is as follows:

$$\text{FNR} = \frac{\text{False Negatives}}{\text{False Negatives} + \text{True Positives}}$$

- *Recall*: It is also called the true positive rate (TPR) corresponding to the ratio of correctly predicted positive values to the total number of positive values in the dataset. (Badillo et al., 2020) The recall score is calculated using the following equation:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- *Precision*: The ratio of correctly anticipated positive values to the total number of predicted positive values. (Badillo et al., 2020) The precision score is given by:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- *F1 Score* : It is also known as an F-measure. It is the harmonic mean of precision and recall. (Joshi, 2020; Flach, 2019) The F1 score is computed as follows:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- *Confusion Matrix* : It is used for two-class problems. The confusion matrix indicates how successful the algorithm was at predicting labels in a binary classification problem where labels take values 0 (called “negative”) or 1 (called “positive”) by evaluating the predicted versus the real labels. Every data point in the test set belongs to one of the four categories true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) and different measures can be derived from these numbers. (Badillo et al., 2020)

- *The receiver operating characteristics (ROC)*: The ROC curve is a two-dimensional graph with TPR as the y-axis and FPR as the x-axis. It shows the performance of a classification model at all classification thresholds and balances the benefits of true positives against the costs of false positives. (Tharwat, 2020)
- *The Area Under the ROC Curve (AUC)*: It is a metric for binary classification that is used to calculate the area under the ROC curve. It quantifies the model's ability to distinguish between the positive and negative classes across all possible thresholds, ranging from 0 to 1. (Tharwat, 2020)

2.3 Approach

In order to classify each sentence as being either correctly parsed or not, supervised learning was used. Supervised Learning is a type of Machine Learning where the algorithm is trained using labeled data. In this approach, the input data is accompanied by the corresponding output or target variable. The goal of supervised learning is to learn a mapping function that can accurately predict the output for unseen input data.

Classification plays a crucial role in this project. The objective is to categorize sentences into two classes: correctly parsed or incorrectly parsed. As the task involves binary classification using supervised learning, the training dataset comprises labeled examples, with each sentence designated as either correctly or incorrectly parsed. This labeled data is utilized to train the classification models, enabling them to accurately predict the class of each sentence.

The following subsections will provide a brief explanation of each model used in the project, along with the corresponding results obtained when applied to the dataset.

2.3.1 Support Vector Machines

Support Vector Machines (SVM) is a powerful algorithm used in classification tasks and in this project to classify sentences. SVC, or Support Vector Classifier, is an implementation of SVM for binary classification. An SVC model classifies examples by separating the classes with the decision surface that maximizes the margin between the classes. It constructs a hyper-plane in a high dimensional space and identifies support vectors, the closest data points to the decision boundary, and maximizes the margin between them for clear separation. New instances are then mapped into that space and predicted to belong to a class based on the side of the gap they fall on. An illustration of classification using SVM is shown below in Figure 2.1.

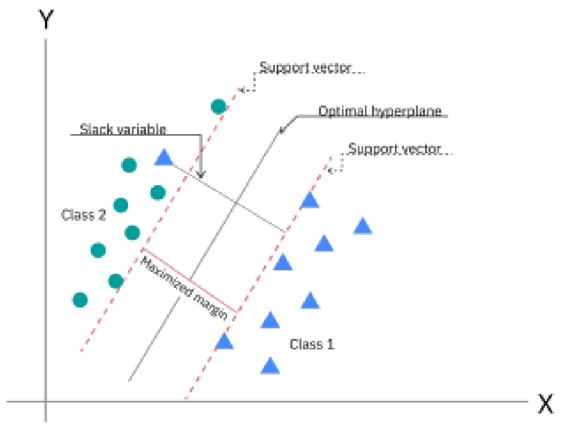


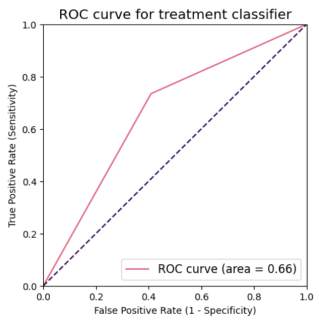
Figure 2.1: An SVC example (IBM, 2023)

In the project, a SVC model was employed, utilizing a linear kernel with enabled probability. A linear kernel implies that the decision boundary between classes is a hyperplane in the feature space, making it suitable for linearly separable data. Enabled probability estimates allow the model to provide probability scores for each class prediction, offering insights into the confidence of the model's classifications.

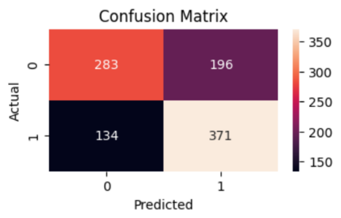
SVM achieved an accuracy of 66.46%, the highest among all models. It presents a precision of 0.68 for class 0 and 0.65 for class 1. The recall for class 0 is 0.59, while for class 1, it is 0.73. Consequently, the F1-score is 0.63 for class 0 and 0.69 for class 1. These metrics suggest that the SVM model struggles with both precision and recall for class 0, while showing relatively better performance for class 1. Additionally, the AUC of (0.66) indicates that the model has a moderate ability to distinguish between positive and negative classes.

Class	**Precision**	**Recall**	**F1-score**	**Support**
0	0.68	0.59	0.63	479
1	0.65	0.73	0.69	505
Macro avg	0.67	0.66	0.66	984
Weighted avg	0.67	0.66	0.66	984

Table 2.2: Classification Report for SVM



((a)) ROC Curve for SVM



((b)) Confusion Matrix for SVM

Figure 2.2: ROC Curve and Confusion Matrix for SVM

2.3.2 Decision Trees

Decision trees (DT) are tree structures that express independent features and a dependent feature representing a set of decisions. Thus, decision tree models classify instances by sorting them based on feature values. A typical decision tree is composed of decision nodes representing features, branches denoting decision rules, and leaves corresponding to the classification outcome. By recursively partitioning the data based on the most informative features, decision trees enable to accurately classify sentences by evaluating a sequence of decision rules. This algorithm allows to uncover important patterns and features that contribute to distinguishing between correctly parsed and incorrectly parsed sentences, enhancing the effectiveness of this project.

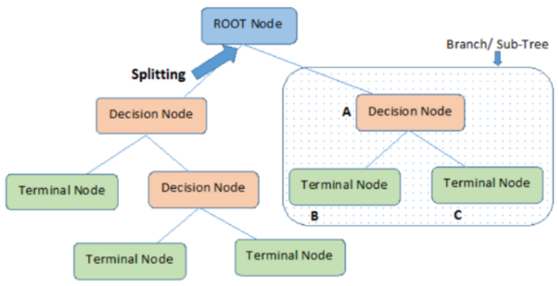
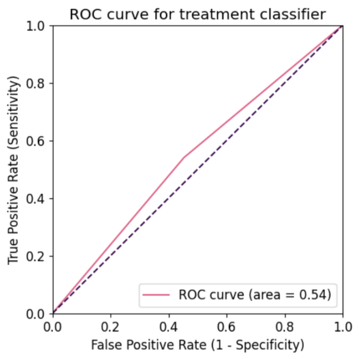


Figure 2.3: Decision Tree Algorithm (Saini, 2024)

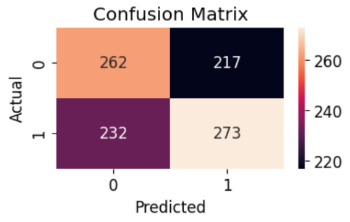
The Decision Tree classifier achieved an accuracy of 54.36% showing that it identified correctly approximately 54.36% of the instances in the dataset. The precision, recall, and F1-score for both classes 0 and 1 are relatively balanced ranging from 0.54 to 0.56. These metrics indicate that the Decision Tree model performs well in identifying both classes without any bias towards either one. The AUC (0.54) is above random chance (0.5) which indicates a moderate discrimination ability for the model.

Class	**Precision**	**Recall**	**F1-score**	**Support**
0	0.53	0.55	0.54	479
1	0.56	0.54	0.55	505
Macro avg	0.54	0.54	0.54	984
Weighted avg	0.54	0.54	0.54	984

Table 2.3: Classification Report for Decision Tree



((a)) ROC Curve for Decision Tree



((b)) Confusion Matrix for Decision Tree

Figure 2.4: ROC Curve and Confusion Matrix for Decision Tree

2.3.3 Random Forest

Random Forest (RF) is a tree-based algorithm where the classifiers' output is the majority vote among several tree classifiers, called an ensemble of trees. A subset from the full training set is randomly sampled for training each of the three classifiers. Random Forest provides several benefits, including improved accuracy, resistance to over-fitting, and the ability to handle large and complex datasets.

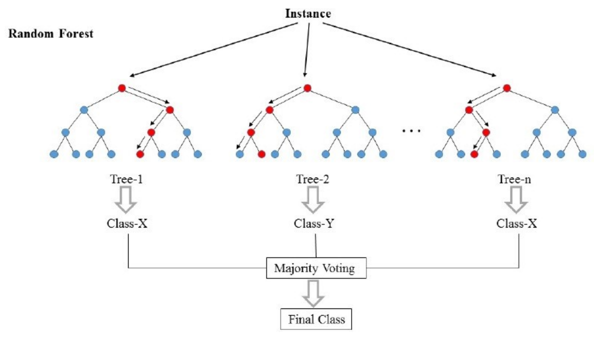
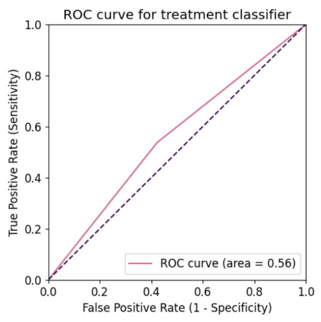


Figure 2.5: An Illustration of Random Forest algorithm (Dimitriadis et al., 2018)

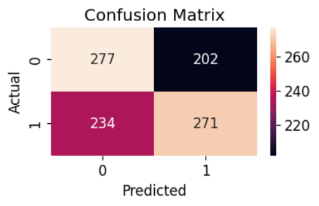
The Random Forest classifier has an accuracy score of 55.69%. Precision, recall, and F1-score for both classes 0 and 1 are relatively balanced, with values ranging from approximately 0.54 to 0.57, suggesting a good performance in identifying instances from both classes without any bias. The AUC (0.56) suggests moderate discrimination ability for the model. Overall, these results show that the Random Forest model performs moderately well in this classification task but further improvement may be explored to enhance its performance.

Class	**Precision**	**Recall**	**F1-score**	**Support**
0	0.54	0.58	0.56	479
1	0.57	0.54	0.55	505
Macro avg	0.56	0.56	0.56	984
Weighted avg	0.56	0.56	0.56	984

Table 2.4: Classification Report for Random Forest



((a)) ROC Curve for Random Forest



((b)) Confusion Matrix for Random Forest

Figure 2.6: ROC Curve and Confusion Matrix for Random Forest

2.3.4 Logistic Regression

Logistic Regression (LR) is a concept that machine learning borrows from statistics. It typically creates a linear model based on a transformed output variable or the target class. The logistic regression model is applied for binary classification problems and it works by changing the original target variable that cannot be approximated using a linear function, into a log transformation of the odd ratio. It is widely used due to its simplicity, interpretability, and efficiency. Logistic Regression provides a clear boundary between the two classes, allowing for accurate predictions in identifying whether each sentence is correctly parsed or not.

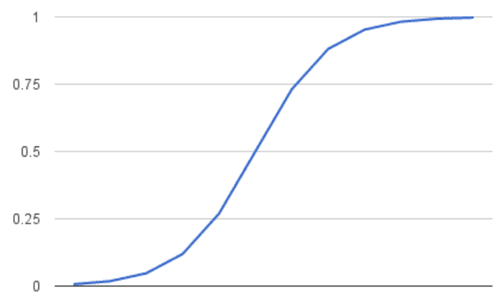
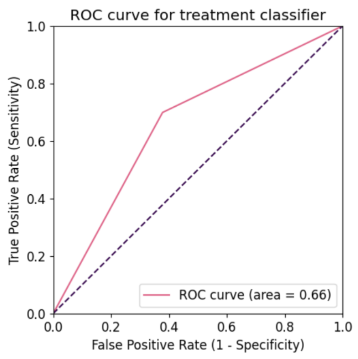


Figure 2.7: Logistic function for Logistic Regression (Brownlee, 2023)

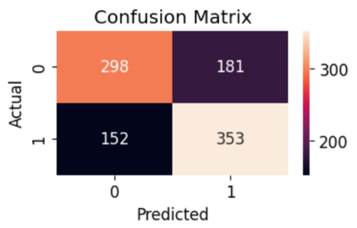
This model resulted in an accuracy score of 66.15%, meaning that it was able to correctly identify around two-thirds of the instances for each class. The model had variations between classes in terms of recall with 0.62 for class 0 and 0.70 for class 1. Precision was balanced between both classes with a score of 0.66, while the F1-score was approximately 0.64 for class 0 and 0.68 for class 1. Moreover, the AUC (0.66) showcases that the model's performance in distinguishing between positive and negative instances was moderately good.

Class	**Precision**	**Recall**	**F1-score**	**Support**
0	0.66	0.62	0.64	479
1	0.66	0.70	0.68	505
Macro avg	0.66	0.66	0.66	984
Weighted avg	0.66	0.66	0.66	984

Table 2.5: Classification Report for Logistic regression



((a)) ROC Curve for Logistic Regression



((b)) Confusion Matrix for Logistic Regression

Figure 2.8: ROC Curve and Confusion Matrix for Logistic Regression

2.3.5 K-Nearest Neighbors

K-Nearest Neighbor (KNN) is a type of instance-based classification algorithm. The learning task in such algorithms consists of simply storing the training data. When making predictions, KNN identifies the K nearest neighbors to the given data point based on a distance metric; a typical one is the Euclidean distance; and assigns the majority class label among those neighbors as the predicted label. The Euclidean distance between two points X and Y can be calculated as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Where X, Y are two points in Euclidean n-space, x_i, y_i are Euclidean vectors starting from the origin of the space of the original point, and n is n-space.

Figure 2.9 depicts a simplified example of the KNN algorithm where the green object of the unknown class would be labeled as a triangle class if k = 3 and a square if k = 5.

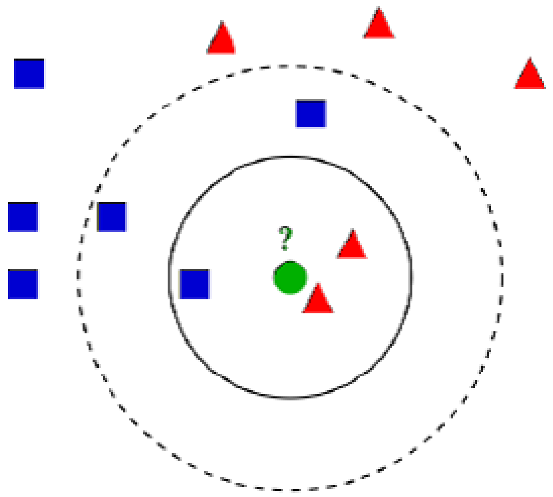


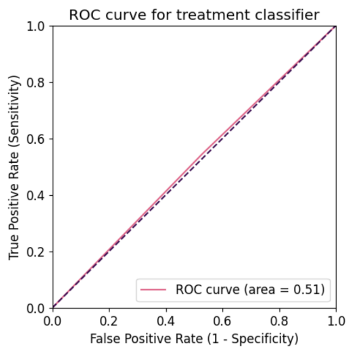
Figure 2.9: KNN example (Dimitriadis et al., 2018)

In the project, a K-Nearest Neighbors (KNN) classifier was employed, configured with a parameter specifying the number of nearest neighbors considered during classification, set to 5.

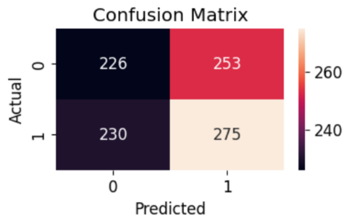
K-Nearest Neighbors achieved an accuracy of 50.91%. The model showed some predictive capability with a precision score of 0.50 for class 1 and 0.52 for class 1, a recall of 0.47 for class 0 and 0.54 for class 1, and F1-scores around 0.48 for class 0 and 0.53 for class 1. The AUC of (0.51) indicates that the discrimination ability for the model is only marginally better than random chance.

Class	Precision	Recall	F1-score	Support
0	0.50	0.47	0.48	479
1	0.52	0.54	0.53	505
Macro avg	0.51	0.51	0.51	984
Weighted avg	0.51	0.51	0.51	984

Table 2.6: Classification Report for K-Nearest Neighbors



((a)) ROC Curve for K-Nearest Neighbors



((b)) Confusion Matrix for K-Nearest Neighbors

Figure 2.10: ROC Curve and Confusion Matrix for K-Nearest Neighbors

2.3.6 Artificial Neural Network

A Neural Network (NN) is a computational or mathematical model that mimics the functions of the human brain. Neural network-based models are composed of interconnected nodes (artificial neurons) and edges that interconnect these nodes. The neurons are computational units that receive and process some input information and produce some output so that the output of one neuron is passed to become the input of another. Figure 2.11 depicts the structure of an artificial neuron.

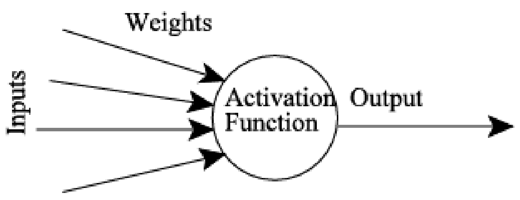


Figure 2.11: The structure of an artificial neuron (Gershenson, 2003)

The most common learning algorithm for estimating the values of weights is the Back Propagation algorithm, which involves a phase for propagation and another one for updating weights. Both phases should be repeated until the network's performance is good enough. Also, the output values are compared with the desired output to compute the value of some predefined error function, and the error is fed back through the network. Next, by using this information, the algorithm adjusts each connection's weights to decrease the value of the error function. After the repetition of this process for an enough large number of training cycles, the network will typically converge to some state with a small calculation error. When the network reaches this state, it can be considered that it has learned a specific target function.

The basic architecture of an ANN consists of three types of neuron layers: input, hidden, and output layers. In this project, an ANN was implemented using Keras Sequential API. The model consists of one dense input layer and one dense hidden layers, each with 16 neurons, employing Rectified Linear Unit (ReLU) activation functions for introducing non-linearity. Dropout layers with a rate of 0.5 are inserted after each hidden layer to mitigate over-fitting by randomly deactivating half of the neurons during training. Additionally, L2 regularization with a strength of 0.01 is applied to the weights of both input and hidden layers to further prevent over-fitting. The output layer comprises a single neuron using a sigmoid activation function, suitable for binary classification tasks, which outputs probabilities ranging between 0 and 1.

In the process of model development, various configurations were explored. Different numbers of neurons were tested in the hidden layer, and hyper-parameter tuning was performed to optimize the model's performance. This iterative approach ultimately led to the selection of the architecture described above.

Following the model definition, the neural network was compiled using the Adam optimizer, which is known for its efficiency in training deep learning models. For loss calculation, binary cross-entropy is chosen, a common choice for binary classification problems.

Additionally, metrics for evaluating the model's performance during training are specified, including accuracy, precision, recall, and area under the curve (AUC). Once compiled, the model is trained for 20 epochs with a batch size of 32. Furthermore, 20 of the training data is used as validation data to monitor the model's performance on unseen data during training, helping to detect over-fitting.

The Artificial Neural Network (ANN) achieved an accuracy score of 64.05%. Precision, recall, and F1-scores for both classes 0 and 1 are relatively balanced, ranging approximately from 0.62 to 0.67. The AUC value of (0.68) suggests a moderate discrimination ability for the model. Overall, the ANN has one of the best performances after the two best performing classifiers SVM and Logistic regression , giving a moderate prediction across different evaluation metrics.

Metric	**Value**
Accuracy	0.6405
Precision	0.6214
Recall	0.7069
AUC	0.6964
Validation Loss	0.6812
Validation Accuracy	0.6372
Validation Precision	0.6391
Validation Recall	0.6733
Validation AUC	0.6853

2.4 Model Comparison

Table 2.7 presents the results for all the classifiers in terms of four evaluation metrics: Accuracy, Precision, Recall and F1 Score.

Model	**Accuracy (%)**	**Precision**		**Recall**		**F1-score**		**AUC**
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	
Logistic Regression	66.15	0.66	0.66	0.62	0.70	0.64	0.68	0.66
K-Nearest Neighbors	50.91	0.50	0.52	0.47	0.54	0.48	0.53	0.51
Random Forest	55.69	0.54	0.57	0.58	0.54	0.56	0.55	0.56
Decision Tree	54.36	0.53	0.56	0.55	0.54	0.54	0.55	0.54
Support Vector Machine	66.46	0.68	0.65	0.59	0.73	0.63	0.69	0.66

Table 2.7: Comparison of Machine Learning Models

Figure 2.12, is a visual comparison of the classification models in terms of accuracy rate. It is noticed that SVM emerges as the best classifier scoring the highest accuracy 66.46% and Logistic regression closely follows the SVM with a competitive accuracy of 66.15%. Random Forest and Decision Tree classifiers display moderate accuracy levels of 55.69% and 54.36%, respectively, while K-Nearest Neighbors lags behind with the lowest accuracy of 50.91%.

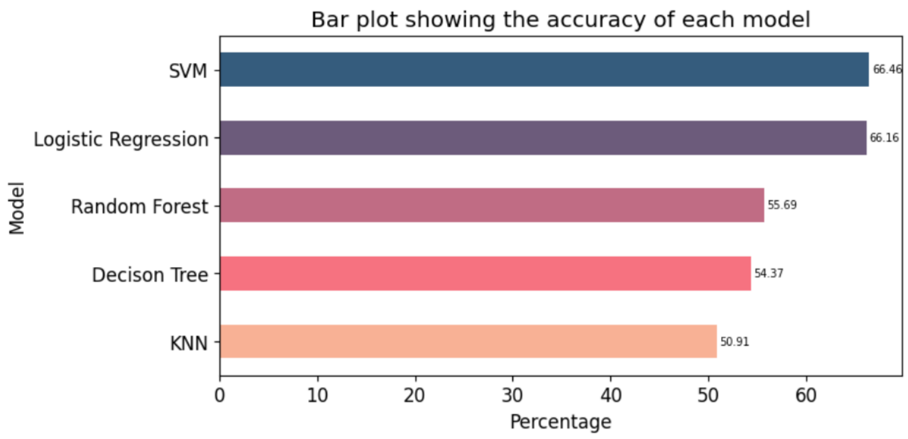


Figure 2.12: Bar plot showing the accuracy of each model

Figure 2.13 represents an ROC curve that compares visually how well each classifier distinguishes between positive and negative instances by measuring the AUC of each model, helping us identify the most effective model. It is observed that Logistic Regression demonstrates a strong ability to tell positive from negative instances with an AUC of 0.68, making it the most effective classifier among all the classifiers. SVM performs slightly worse with an AUC of 0.64, with better separation between positive and negative instances. Subsequently, the decision tree and the random forest classifiers have the same AUC score of 0.59, with a moderate discriminative ability. Lastly, the KNN classifier falls short with an AUC of 0.49, which means a weak discrimination ability.

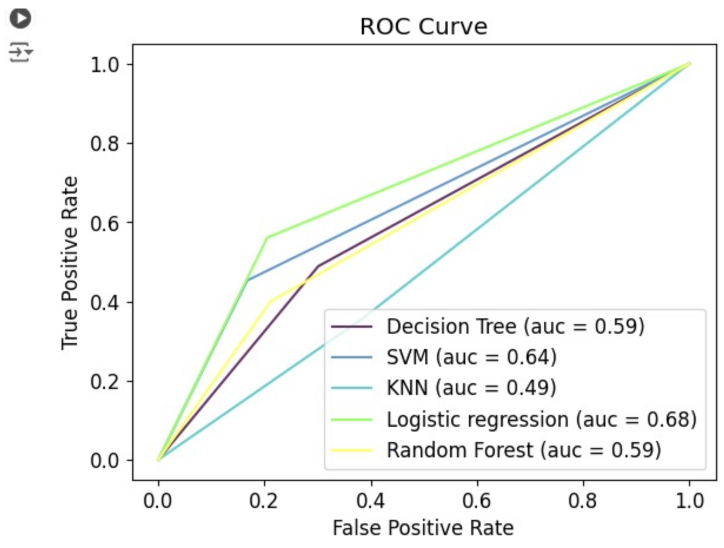


Figure 2.13: ROC curve the performance of each classifier

Chapter 3

Future Work

It can be argued that the limited size of the dataset has been a performance bottleneck and that the next logical step would be to construct a bigger dataset to improve performance. However, one of the biggest challenges faced over the course of developing this project is that more data does not necessarily lead to better results and, in some cases, more data can even lead to drops in performance. Reliability research is particularly sensitive to variation in data; therefore, incorporating data from various sources to achieve a bigger dataset comes with its own set of concerns. In order to construct a bigger dataset, each new addition should be scrutinized to make sure that it follows the same annotation guidelines. This is likely to be a very time-consuming process. The annotation guideline used to annotate a given dataset is not always publicly available and in cases where they have been published, it is usually not possible to document all the principles that have been followed given that language data is full of ambiguities.

Human annotators tend to rely on their domain knowledge and even experts often disagree on the annotation of the very same utterances. Replicability research on word sense disambiguation has pointed out that if human annotators can agree on a given annotation only some percentage of the time, it is not clear how to interpret a model's performance when it surpasses this percentage. (Kilgarriff, 1999) Annotation projects generally make use of inter-annotator agreement scores to ensure a consistently annotated dataset. However, they seldom report the agreement scores for each data point in the published dataset. It has been shown that using annotator disagreement information in the loss function of POS taggers increases performance for downstream tasks. (Plank et al., 2014) A similar approach could be adopted for the reliability estimation model for dependency parsing. If there are no dependency parsing datasets with point-wise agreement scores available, the data would have to be constructed from the ground up.

Another approach to the task of building a reliability estimator for dependency parsing would be to utilize labeled attachment scores (LAS) instead of using binary classes. Each annotated sentence would be associated with its calculated LAS. In this approach, the problem of reliability estimation could be framed as a regression task rather than a classification task due to the presence of continuous LAS values. A heuristic threshold can be employed in order to deem an annotation reliable or unreliable.

There are also some improvements that can be made to this project without taking on the bigger and more time-consuming task of constructing the aforementioned datasets. Conducting rigorous cross-validation experiments and testing the generalization of the classifiers on the datasets would be essential steps for assessing the reliability and robustness. Evaluating classifier performance across diverse datasets and real-world scenarios can provide valuable insights into the applicability and potential limitations of the trained classifiers.

Exploring deep learning architectures, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), may offer additional insights and performance gains for the classification task. Deep learning models have shown remarkable success in various domains and could potentially outperform traditional machine learning algorithms in certain scenarios.

Chapter 4

Conclusion

To summarize, state-of-the-art dependency parsers boast good results when tested on data that falls under their training domain; however, their performance drops when tested on any other data. Since it is not possible as of yet to ensure that a new and unseen input sentence will be in the domain of a given parser, there is reason to question the reliability of the syntactic annotations produced by the parsers. With that in mind, this project explored ways of assessing the quality of syntactic annotations. The approach taken was to frame it as a classification task and various classifiers were trained to tackle the problem at hand. Since the classifiers were trained on the output of parsers, they are intended to act as an outer layer of quality check on the parsers.

Based on the obtained results, Support Vector Machine (SVM) achieved the highest accuracy out of all models, demonstrating its effectiveness in accurately classifying instances. However, it struggled with precision and recall for class 0, indicating potential areas for improvement. The Logistic Regression model closely followed SVM in terms of accuracy, showcasing its robustness in classification tasks.

Additionally, Logistic Regression and SVM stood out with the highest accuracy, precision, and recall scores. Artificial Neural Networks also scored competitive precision, recall, and area under the curve (AUC) values. While Random Forest and Decision Tree classifiers displayed moderate accuracy levels, their discrimination ability, as indicated by AUC values, was also moderate. This suggests that while they perform reasonably well in classification, there is room for improvement in distinguishing between positive and negative instances. On the other hand, KNN exhibited the lowest accuracy and only marginal discrimination ability, indicating limitations in its predictive capability for the classification task examined. All in all, the trained models only achieve up to 66.15% accuracy and leave room for improvement.

As previously proposed, there are a number of alternative or additional strategies that can be employed to build on this project. Performing cross-validation experiments and different hyper-parameter tuning methods may yield performance gains. A more meticulous and controlled approach to data curation can lead to significant improvements in performance. Finally, framing the task at hand as a regression task rather than a classification task might be a promising alternative approach.

Bibliography

Alaliyat, Saleh (2008). *Video-based Fall Detection in Elderly's Houses*. URL: https://www.researchgate.net/publication/267953942_Video_-based_Fall_Detection_in_Elderly's_Houses.

Alaloul, Wesam Salah and Abdul Hannan Qureshi (2020). "Data Processing Using Artificial Neural Networks". In: *Dynamic Data Assimilation - Beating the uncertainties*. InTechOpen. DOI: 10.5772/intechopen.91935. URL: https://www.intechopen.com/chapters/71673.

Badillo, Solveig, Balazs Banfai, Fabian Birzele, Iakov I. Davydov, Lucy Hutchinson, Tony Kam-Thong, Juliane Siebourg-Polster, Bernhard Steiert, and Jitao David Zhang (2020). "An Introduction to Machine Learning". en. In: *Clinical Pharmacology & Therapeutics* 107.4, pp. 871–885. ISSN: 1532-6535. DOI: 10.1002/cpt.1796. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpt.1796.

Basar, Ezgi, Meryem Bouziani, Hanady Yasmine, and Hedwig Naava N. (2024). *Syntactic Parsing Project*. https://github.com/Mery-e/SyntacticParsingProject.

Brownlee, Jason (2023). *Logistic Regression for Machine Learning*. URL: https://machinelearningmastery.com/logistic-regression-for-machine-learning/.

Che, Wanxiang, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu (Oct. 2018). "Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Ed. by Daniel Zeman and Jan Hajič. Brussels, Belgium: Association for Computational Linguistics, pp. 55–64. DOI: 10.18653/v1/K18-2005. URL: https://aclanthology.org/K18-2005.

Dimitriadis, Stavros I. and Dimitris Liparas (2018). "How random is the random forest? Random forest algorithm on the service of structural imaging biomarkers for Alzheimer's disease: from Alzheimer's disease neuroimaging initiative (ADNI) database". In: *Neural Regeneration Research* 13.6, pp. 962–970. DOI: 10.4103/1673-5374.233433. URL: https://pubmed.ncbi.nlm.nih.gov/29926817/.

Flach, Peter (July 2019). "Performance Evaluation in Machine Learning: The Good, the Bad, the Ugly, and the Way Forward". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01. Number: 01, pp. 9808–9814. ISSN: 2374-3468. DOI:

10.1609/aaai.v33i01.33019808. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5055.

Gershenson, Carlos (2003). “Artificial Neural Networks for Beginners”. In: *CoRR* cs/0308031. arXiv: 0308031. URL: https://arxiv.org/abs/cs/0308031.

Harris, Zellig S. (1954). “Distributional Structure”. In: *WORD* 10.2-3, pp. 146–162. DOI: 10.1080/00437956.1954.11659520. eprint: https://doi.org/10.1080/00437956.1954.11659520. URL: https://doi.org/10.1080/00437956.1954.11659520.

IBM (2023). *Support Vector Machine*. IBM. URL: https://www.ibm.com/topics/support-vector-machine.

Joshi, Ameet V. (2020). “Performance Measurement”. en. In: *Machine Learning and Artificial Intelligence*. Ed. by Ameet V Joshi. Cham: Springer International Publishing, pp. 169–176. ISBN: 978-3-030-26622-6. DOI: 10.1007/978-3-030-26622-6_17. URL: https://doi.org/10.1007/978-3-030-26622-6_17.

Kanerva, Jenna, Filip Ginter, Niko Miekka, Akseli Leino, and Tapio Salakoski (Oct. 2018). “Turku Neural Parser Pipeline: An End-to-End System for the CoNLL 2018 Shared Task”. In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Ed. by Daniel Zeman and Jan Hajič. Brussels, Belgium: Association for Computational Linguistics, pp. 133–142. DOI: 10.18653/v1/K18-2013. URL: https://aclanthology.org/K18-2013.

Kilgarriff, Adam (June 1999). “95% Replicability for Manual Word Sense Tagging”. In: *Ninth Conference of the European Chapter of the Association for Computational Linguistics*. Ed. by Henry S. Thompson and Alex Lascarides. Bergen, Norway: Association for Computational Linguistics, pp. 277–278. URL: https://aclanthology.org/E99-1046.

Kübler, Sandra, Ryan McDonald, and Joakim Nivre (2009). “Introduction”. In: *Dependency Parsing*. Cham: Springer International Publishing, pp. 1–10. ISBN: 978-3-031-02131-2. DOI: 10.1007/978-3-031-02131-2_1. URL: https://doi.org/10.1007/978-3-031-02131-2_1.

Lease, Matthew, Eugene Charniak, Mark Johnson, and David McClosky (2006). “A look at parsing and its applications”. In: *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*. Vol. 21. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, p. 1642.

Marneffe, Marie-Catherine de, Christopher D. Manning, Joakim Nivre, and Daniel Zeman (July 2021). “Universal Dependencies”. In: *Computational Linguistics* 47.2, pp. 255–308. ISSN: 0891-2017. DOI: 10.1162/coli_a_00402. eprint: https://direct.mit.edu/coli/article-pdf/47/2/255/1938138/coli_a_00402.pdf. URL: https://doi.org/10.1162/coli_a_00402.

Nivre, Joakim, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman (May 2020). "Universal Dependencies v2: An Evergrowing Multilingual Treebank Collection". English. In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Ed. by Nicoletta Calzolari et al. Marseille, France: European Language Resources Association, pp. 4034–4043. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.497.

Petrov, Slav and Ryan T. McDonald (2012). "Overview of the 2012 Shared Task on Parsing the Web". In: URL: https://api.semanticscholar.org/CorpusID:11108420.

Plank, Barbara, Dirk Hovy, and Anders Søgaard (Apr. 2014). "Learning part-of-speech taggers with inter-annotator agreement loss". In: *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Ed. by Shuly Wintner, Sharon Goldwater, and Stefan Riezler. Gothenburg, Sweden: Association for Computational Linguistics, pp. 742–751. DOI: 10.3115/v1/E14-1078. URL: https://aclanthology.org/E14-1078.

Qi, Peng, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning (2018). "Universal Dependency Parsing from Scratch". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Brussels, Belgium: Association for Computational Linguistics, pp. 160–170. URL: https://nlp.stanford.edu/pubs/qi2018universal.pdf.

Ruder, Sebastian (Feb. 2022). *NLP-progress*. Version 1.0.0. DOI: 10.5281/zenodo.1234. URL: https://nlpprogress.com/.

Saini, Anshul (2024). *Decision Tree Algorithm*. URL: https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/.

Straka, Milan (Oct. 2018). "UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Ed. by Daniel Zeman and Jan Hajič. Brussels, Belgium: Association for Computational Linguistics, pp. 197–207. DOI: 10.18653/v1/K18-2020. URL: https://aclanthology.org/K18-2020.

Tharwat, A (2020). "Classification assessment methods". In: *Applied computing and informatics*. DOI: 10.1016/j.aci.2018.08.003. URL: https://www.emerald.com/insight/content/doi/10.1016/j.aci.2018.08.003/full/html..

Zeman, Daniel, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov (Oct. 2018). "CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Ed. by Daniel Zeman and Jan Hajič. Brussels, Belgium: Association for Computational

Linguistics, pp. 1–21. DOI: 10.18653/v1/K18-2001. URL: https://aclanthology.org/K18-2001.