UNIVERSITY OF ARIZONA

Application of Logistic Regression in Automatic Reading Comprehension and Detection of English Grammatical Structures Based on Difficulty

Author:
Ragheb AL-GHEZI

Supervisor:
Dr. Robert HENDERSON

A report submitted in fulfillment of the requirements for the degree of Masters of Science

in the

Human Language Technology

UNIVERSITY OF ARIZONA

Abstract

Faculty Name

Masters of Science

Application of Logistic Regression in Automatic Reading Comprehension and Detection of English Grammatical Structures Based on Difficulty

by Ragheb AL-GHEZI

In this study we examine the use of a probabilistic classification algorithm called Logistic Regression on two novel Natural Language Processing (NLP) tasks: Automatic Reading Comprehension and Detection of Grammatical Difficulty in English Sentences. Logistic Regression has been used extensively in machine learning and NLP research over the past decades, but it has gained popularity as it is considered the building block of Artificial Neural Network (ANN). Both tasks studied in this report show a unique use to multinomial logistic regression classifier. In automatic reading comprehension task, given a reading passage and a question, the goal is to extract an answer from the passage to the question if there exists one or abstain from answering if the passage does not have an answer. For this purpose, a multinomial logistic regressor is used on two stages: a) find the sentence that most likely contains the answer (null-answer is treated as a sentence at this stage). b) within the best candidate sentence, another logistic regressor is used to extract the constituent that most likely represent the answer. The experiment results show that using simply designed features a logistic regressor score 0.71 F1 score and 0.40 F1 in stage 1 and 2 respectively, making it as a potential competitor to other state-of-the-art advanced neural network architectures. In the second task, detection of grammatical difficulty, a multinomial logistic regressor is applied in a semi-supervised manner on low-resourced data to categorize the difficulty of English grammatical structures according to Common European Framework Reference (CEFR) guidelines. The method used shows a significant improvement from 0.67 F1 to 0.79 F1.

Acknowledgements

I would like to thank the Fulbright team, represented by the Amideast for affording me the un-imaginable opportunity to complete my study at the University of Arizona. I also would like to extend my sincerest gratitude to my adviser, Dr. Robert Henderson for his constant help and support.

I owe a special and an exceeding thanks to Dr. Mihai Surdeanu, Dr. Michael Hammond and Dr. Sandiway Fong for empowering my skills measurably in the area of Natural Language Processing

Finally, I am grateful to Stephanie Amado and Maria Alexeeva for their unfailing support and assistance.

Contents

List of Figures

List of Tables

Dedicated to my parents
For their endless love, support and encouragement . . .

Chapter 1

Introduction

1.1 Background

Lum is a company developing customized machine reading technology using machine learning and natural language processing. Their technology serves to a variety of medical, agricultural and educational domains and sectors. During a 3-month internship, the researcher was assigned two main NLP tasks that contributed to solving problems in teaching foreign languages using online distance learning. The two NLP tasks are automatic reading comprehension and detecting grammatical difficulty of English sentences according to CEFR. The goal of the first task is to design and implement a system that, given a reading passage and a set of questions, can automatically find answers to those questions from within the reading passage. The motivation for such a system is that it can help human language instructors create automatically graded assignments and quiz activities in virtual learning environments. The purpose of the second task is to create a system that can automatically categorize a given English text into three levels of grammatical difficulty: Elementary (A), Intermediate (B), and Advanced (C). In this report, the researcher investigates the possibility to use NLP and ML algorithms to solve these two problems. The report also covers the entire process: data description and analysis, literature survey, feature design, and engineering and evaluating the performance of the algorithm.

1.2 Goals

One goal of this internship report is to examine the suitability of using a machine learning linear classification algorithm called logistic regression on two novel educational tasks: automatic reading comprehension and detection of grammatical difficulty. Another goal is to explore and compare different methods of feature extraction and feature engineering to each of these tasks. Furthermore, examining the use of logistic regression in a semi-supervised fashion to bootstrap the scarcity of training data and improves the overall quality of generalization.

1.3 Outline

The first chapter gives a general background about the internship that the researcher undertook as well as the goals intended to be achieved through this internship report.

The second chapter is dedicated to reviewing the theory of machine learning in the context of natural language processing. It also defines classification as a supervised machine learning task and logistic regression as one popular linear classification algorithm. The chapter also includes the conventional methods of computational representation of human languages such as bag-of-words, term-frequency-inverse-document-frequency, and word embeddings. Finally, it concludes by showing the metrics used to evaluate classification systems.

The third chapter introduces the task of detecting grammatical errors in English sentences as a classification problem and explains the use of multinomial logistic regression in a semi-supervised fashion to solve it. The chapter also briefly surveys the most important studies in the literature that tackled similar textual classification tasks. Finally, it shows the experiments, the results of applying the algorithm of the data collected, and the conclusions.

The fourth chapter introduces the task of automatic reading comprehension and the data set used for this purpose. It also discusses the common approaches tackling these tasks in the literature. Next, the chapter explores different feature extraction methods to create a numerical representation of text. Finally, the chapter is concluded with the result and evaluation as well as sections dedicated to error analysis and future work respectively.

The fifth chapter contains the overall conclusion of the internship report.

Chapter 2

Theory

2.1 Introduction

In this chapter, we explain the fundamentals of machine learning theory, with emphasis on supervised classification tasks. We start by describing the three primary stages of the machine learning process: feature extraction, classifier training, and testing and evaluation. We also discuss two types of feature engineering methods: frequency-based and distributional in addition to dimensionality reduction methods. Next, we introduce a linear classification technique called logistic regression and explain its mathematical foundations. Finally, we discuss the metrics used to evaluate the performance of supervised classification tasks.

2.2 Classification as a Machine Learning Task

Generally, there are two types of supervised learning tasks according to the data type of the predicted output: regression and classification. When the predicted output is a continuous value, such as predicting the price of a house or the temperature, the task is called regression. A classification task, on the other hand, is when the predicted output is a discrete value such as predicting the sentiment of a document (negative or positive) or predicting the weather state (rainy, sunny, cloudy, etc).

The goal of a classification task is to select a correct class for a given input, or more generally a task of "assigning objects from a universe to two or more classes or categories" Manning & Schütze (1999, p.575). A classifier is an algorithm that quantitatively models the relationship between a set of inputs and their associated outputs such that it learns how to label new data. In the task of spam filtering, for instance, input data are texts and the output are binary labels (0 or 1) representing whether or not a document is spam.

Any classification task has two main phases: training and prediction 2.1. During training, a feature extractor is used to convert each input into a set of features which are designed to capture the basic information about each input. Next, pairs of feature sets and their corresponding labels are fed into the machine learning algorithm to generate a model. In the prediction phase, the trained model is used on unseen data to predict the labels.

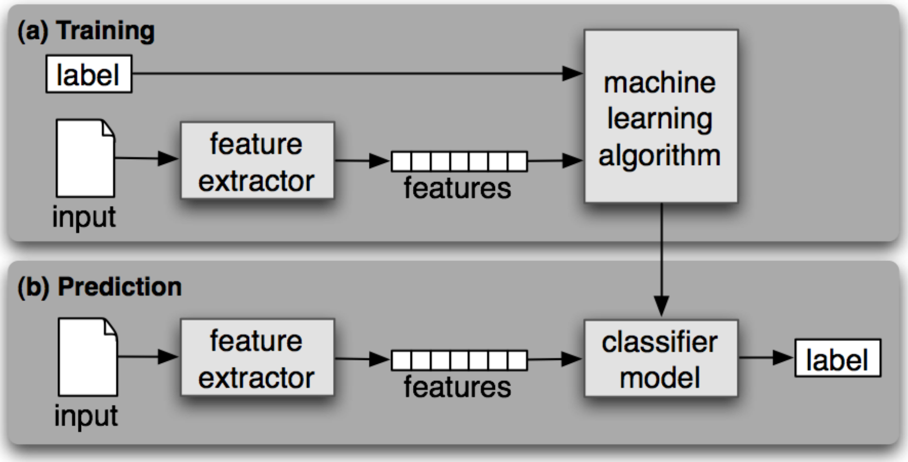


FIGURE 2.1: A diagram illustrating the process of supervised classification learning, adopted from (Bird, Klein, & Loper, 2009, p. 222)

2.3 Feature Extraction

The first step in any machine learning task is feature extraction where a text input is transformed into a set of attributes that characterize it. One example of a negative sentiment document feature may be the presence of the word *terrible*. Choosing what features to use when representing a text or a document not only has a major impact on the entire classification process, but it is also arguably the most difficult phase in any natural language processing task due to several fundamental questions on how do we represent language computationally. For example, on what level should we represent a text: *character-wise*, *word-wise* or *sentence-wise*, or even more specifically morphologically, etymologically, or phonologically? Do we treat a word e.g. *try* and all its morphological variations e.g. *tries*, *trying*, *tried* similarly or differently? How to represent and model the semantics of human language computationally? Questions like these and many others are the core of natural language processing. Nevertheless, several automatic feature extraction methods have developed over the past decades that can be categorized into frequency-based or distributional, and we briefly explore some of the most common ones.

2.3.1 Frequency-based Methods

Frequency-based methods such as Bag-of-Words (BOW) and Term Frequency Inverse Document Frequency (TF-IDF) rely on the number of token occurrences in the textJurafsky & Martin (2014). They have been used extensively in a wide range of NLP applications, and each of them has advantages and disadvantages.

TABLE 2.1: Bag-of-Words Feature Vectorization Example

	the	apple	is	better	than	orange	summer	winter
Vector 1	2	1	1	1	1	1	0	0
Vector 2	0	0	1	1	1	0	1	1

Bag-of-Words Representation

The bag-of-words (BOW) representation is the most straightforward and most intuitive representation. It represents each document by a vector of word counts in the document. To illustrate, the BOW vector representations of a data set of two documents (1) and (2) are shown in table 2.1:

1. The apple is better than the orange.
2. Summer is better than winter.

Despite the simplicity of the BOW method, it has two main disadvantages. One major disadvantage is that the BOW method pays no respect to the order of tokens in the document, and this can give similar representations to semantically different documents. For instance, the sentences *The wolf eats the goat* and *The goat eats the wolf* have identical vector representations, for they contain the exact same words. Another downside to BOW is that high frequency features are more important than low frequency ones. We can see in 2.1 that the component corresponding to the first dimension (the) of vector 1 is bigger than in that of vector 2 because document 1 has two occurrence of the word *the* while the document 2 has only one occurrence.

To address the problem of order, several studies have attempted to count the occurrence of sentences instead of words (Fuhr & Buckley, 1991; Tzeras & Hartmann, 1993). Although the sentences offer the advantage of better preserving the semantic relationship of the words, their use as features failed in practice. According to Lewis 1992, this representation is penalized by a large number of possible combinations which leads to low and highly random frequencies. A more practical approach is representation by n-grams. It consists of breaking the text into moving sequences of *n* consecutive tokens. For example, the trigram segmentation of the word "language" gives the following 3-grams: lan, ang, ngu, gua, uag, age. (Shannon, 1948), who was interested in predicting the appearance of certain characters according to the other characters introduced the notion of n-grams. Since then, n-grams have been used in many areas such as speech recognition and information retrieval. Using n-gram on a character-level or word-level gives a contextual representation of the text. A modification to BOW to overcome the bias of high frequency tokens resulted in devising a method called Term Frequency Inverse Document Frequency (TF-IDF).

TF-IDF Representation

TF-IDF has two main parts. Term Frequency is proportional to the frequency of the term in the document (local weighting)and can be used as is or in several variations (Sable & Church, 2001; Singhal, Mitra, & Buckley, 1997).

$$tf_{ij} = f\left(t_i, d_j\right)$$

$$tf_{ij} = 1 + \log\left(f\left(t_i, d_j\right)\right)$$

$$tf_{ij} = 0.5 + 0.5\frac{f\left(t_i, d_j\right)}{max_{t_i \in d_j} f\left(t_i, d_j\right)}$$

where $f(t_i, d_j)$ is the term frequency of document j. Variations include logarithmic scaling of frequency counting or normalization.

Inverse Document Frequency (IDF) measures the importance of a term throughout the collection (overall weighting). A term that often appears in the document base should not have the same impact as a less frequent term. Indeed, terms that appear in the majority of documents do not have any discriminating power to distinguish documents from each other and must, therefore, have low weightings. The IDF weighting is inversely proportional to the number of documents containing the term to be weighted. Thus, the more the term appears in several documents, the less it is discriminating, which results in being assigned a low weighting. The IDF weighting is generally expressed as follows:

$$idf\left(t_i\right) = \log\left(\frac{N}{df\left(t_i\right)}\right)$$

where $df(t_i)$ is the term frequency of feature i and N is the number of documents in the corpus.

The TF-IDF weighting combines the two weightings, TF and IDF, in order to provide a better approximation of the importance of a term in a document. According to this weighting, for a term to be important in a document, it must frequently appear in the document but rarely occur in other documents. This weighting is given by the product of the term's local weighting in the document by its overall weighting in all the documents of the corpus.

$$tfidf\left(t_i, d_j\right) = tf_{ij} \times \log\left(\frac{N}{df\left(t_i\right)}\right)$$

Frequency-based text representation methods explored so far, despite their usefulness in a variety of tasks in natural language processing, share sparsity as one problem in common. The number of features generated using these methods can easily exceed the tens of thousands, which not only negatively influences the categorization process, but it is also very computationally expensive in terms of hardware resources. In addition, the higher the dimensions of the features are, the weaker the features become. This problem is known as the *curse of dimensionality* (Bellman, 2015). To remedy this issue, techniques borrowed from the field of information theory and linear algebra have been developed to reduce the feature space dimensionality without losing much information.

2.3.2 Dimensionality reduction

There are a number of linguistic and non-linguistic ways to reduce the number of features in text data. Depending on the task, we can aggregate words of certain linguistic relation in concepts. For example, we replace co-hyponyms *dog* and *wolf* with their hypernym *canine*; or *vehicle* and *car* with *automobile*. Concepts, defined as units of knowledge, can be used as features to solve the ambiguity problem as well as the problem of synonymy. Indeed, each concept represents a unique meaning that can be expressed by several synonymous words. Similarly, a word with many meanings (senses) is found mapped to several concepts. Thus, a document containing the word *vehicle* may be indexed by other words, such as car or automobile. The transition from a word representation to a concept representation requires the use of semantic resources external to the content of documents such as semantic networks, thesauri, and ontologies. As a result, the performance of such a representation crucially depends on the semantic richness of the resources used in terms of the number of concepts and relationships between these concepts.

In some cases, we may need to treat morphologically variant words similarly. For example, words such as play, player, players, plays, and played will be replaced by *play*. Lemmatization and stemming are the two techniques used to find the canonical form of a word. Lemmatization uses a knowledge base containing the different inflected forms corresponding to the different possible lemmas. Thus, the inflected forms of a noun will be replaced by the singular masculine form while the infinitive form will replace the different inflected forms of a verb. Lemmatization requires the use of a dictionary of inflected forms of language as well as a grammar labeler. Stemming uses a knowledge base of syntactic and morphological rules and to transform words into their roots. One of the most well-known stemming algorithms for the English language is Porter's algorithm (Porter, 1980). Lemmatization is more complicated to implement since it depends on the grammatical labelers. In addition, it is more sensitive to misspellings than stemming.

Dimensionality reduction can also be done using probabilistic and linear algebraic techniques such as principal component analysis (PCA) and linear discriminant analysis (LDA) that aim to project highly dimensional data into lower dimension space. However, we limit the discussion to only two methods: clustering and Latent Semantic Allocation.

Clustering (Baker & McCallum, 1998; Sable & Church, 2001; Slonim & Tishby, 2001) can be used to reduce dimensionality. It consists of representing the documents in a new representation space other than the original one — each dimension of the new representation space groups terms that share the same meaning. Thus, the documents will no longer be represented by terms but rather by groupings of terms representing semantic concepts. This new space of representation offers the advantage of managing the synonymy since the synonymous terms will appear in the same groupings. Likewise, the fact that a term can be included in several groupings also makes it possible to manage the polysemy. Another exciting method to reduce dimensionality, proposed by Deerwester et al., 1990 is Latent Semantic Allocation (LSA). It uses singular value decomposition of the document's x term matrix to change the representation by keeping only the k axes of the strongest singular values. However, LSA is very expensive in terms of calculation time during learning as it relies on a matrix factorization method called Singular Value Decomposition (SVD), which runs computationally in cubic size. Likewise, with each new document, it is necessary to redo the whole grouping process.

2.3.3 Distributional Method

An alternative approach for text representation in natural language of processing is distributional similarity, which represents the meaning of a word using words that co-occur in the same context. For example, the words 'dog' and 'cat' have the same meaning in the following two sentences: *I have a cat*; *I have a dog*. While this idea might not be adequate to capture or account for the complexity or the sophistication of human language, it has succeeded tremendously in a variety of lexicalized NLP tasks. It has become the de facto representation method of text. The need for such a method does not only come from the need for more accurate numerical representation, but its density compared to other methods makes it less computationally expensive. Word2Vec (Mikolov et al., 2013) and GloVe (Pennington, Socher, & Manning, 2014) are two famous algorithms for creating word embedding.

The intuition of word2vec is to train a classifier on a binary prediction task: "Is word w likely to show up near X?", where X is the word to which we want to find embeddings. Then, we use the weights learned by the classifier as embeddings. Skip-gram algorithm (Mikolov et al., 2013) , first, initializes embeddings randomly. Then, it iteratively updates the embeddings of each word w to be equal to the words they occur within the same context. Finally, it injects a, k, number of non-neighbor words as negative examples.

2.4 Logistic Regression

Logistic regression is a machine learning classifier that is used for a variety of learning tasks in natural language processing and image processing. It belongs to a group of probabilistic classifiers called discriminative models which, unlike its generative counterparts like Naive Bayes classifiers, tries to distinguish the classes rather than learn to generate them. More formally, given a document, x, and a class y, a logistic regression classifier computes the conditional probability of a class, y_i, given its input x_i $P(y|x)$. There are two types of logistic regression classifiers: binary and multinomial. The binary logisitc regression is used when there are two classes of inputs to be predicted such as spam or not spam; positive or negative; malignant or benign. Multinomial or multiclass logistic regression is used when there are more than two predicted classes, such positive, neutral, or negative.

According to Jurafsky & Martin, 2014, classification using logistic regression, like any other supervised classification algorithm, has four main components:

- a feature extractor, which numerically represents language data (characters, words and sentences, etc) input, $\mathbf{X}$, as feature vector $[x_1, x_2, \ldots, x_n]^T$ where n represents the number of features in the input data;
- a classification function to compute an estimation to class $\hat{y}$ via $p(y|x)$ and determine whether the classifier is binary or multinomial;
- an objective function to minimize the error predicted label $\hat{y}$ and actual label, y, in training examples (cross entropy); and
- an optimizer to help find the minimum of the objective function (stochastic gradient descent).

Given a dataset of m, number of observations, x, and classes, y, $\mathcal{D} = \left\{ \left(\boldsymbol{x}^{(i)}, y^{(i)} \right) \right\}_{i=1}^{m}$, the goal is to learn the set of optimal weights $\hat{w}$ that maximizes the log probability of predicting a class, y, given an observation, x

$$\hat{w} = \underset{w}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P \left(y^{(i)} | x^{(i)} \right) \right] \tag{2.1}$$

The distribution type of $P\left(y^{(i)}|x^{(i)}\right)$ determines whether the type of learning is binary or multiclass. The probability distribution of binary output is Bernoulli, while the probability distribution of multiclass output is multinomial.

2.4.1 Classification Functions

What we have discussed so far gives us a method to numerically translate language from a form that is uniquely comprehensible to humans into a representation that a computer can understand, and can somehow capture the characteristics of human language. However, in order for the logistic regression to learn to classify, a classification function should be utilized. Depending on whether the classification task is binary or multinomial, there are two main functions used: Sigmoid and Softmax. Sigmoid function, used for binary classification, outputs 1 if an observation input (feature vector) is a member of a particular class e.g. (spam), and 0 otherwise. In other words, it calculates $P(y = 1|x)$ for spam text and $P(y = 0|x)$ for non-spam text. The mathematical form of the Sigmoid function or (logistic function, hence comes the name of the classifier) is as follows:

$$Sig(x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

where x represent the input vector, w is the learned weights, and b is the bias term or the intercept of the linear equation. One mathematical property of the Sigmoid function is that it maps any input between 0 and 1, so the output can be interpreted as probability.

$$P(y = 1) = \sigma(w \cdot x + b)$$

$$P(y = 0) = 1 - \sigma(w \cdot x + b)$$

The classifier declares an observation input as *Yes* (if it is a member of class spam) if the probability is greater than a certain threshold, say 0.5, and declares an observation input as *No* otherwise.

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

In the case of multinomial classification, the logic remains the same except for the classification function where Softmax is used rather than Sigmoid. Softmax works on normalizing the prediction of an observation by the values of all other classes in order to produce valid probability distribution.

$$p(y = i|x) = \frac{e^{w_i \cdot x + b_i}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

2.4.2 Objective Functions

During the learning process, we need to answer the question, *how correct is the estimated output of $\hat{y}$ of classifier from the true output y?*. Therefore, our objective *function* is to minimize the difference between the estimated output and the true one such that the weights learned during this minimization process are, hopefully, generalizable enough to correctly label observations unseen in the training phase. We can imagine an objective function to be the difference between the $\hat{y}$ and y i.e. $|\hat{y} - y|$, but for mathematical convenience we need a non-convex objective function that can be easily optimized. Thus, we use maximum likelihood estimation.

Binary classification can be seen as Bernoulli distribution since the outcome is either 0 or 1. More formally, the probability of predicting class y given input x is $p(y|x) = \hat{y}^y(1 - \hat{y})^{1-y}$. Minimizing the probability is the same as maximizing the negative log likelihood:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y)\log(1 - \hat{y})]$$

Finally, by plugging in the value of $\hat{y}$ we obtain:

$$L_{CE}(w, b) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma\left(w \cdot x^{(i)} + b\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(w \cdot x^{(i)} + b\right)\right)$$

, where m is the number of training examples.

On the other hand, multinomial classification uses the same process except that it uses a different classification function softmax and hence has different (multinomial) distribution. As a result, the loss function for a single example x is the sum of the logs of the

K output classes:

$$L_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} 1\{y = k\} \log \frac{e^{w_k \cdot x + b_k}}{\sum_{j=1}^{K} e^{w_j \cdot x + b_j}}$$

2.4.3 Optimization

The objective functions derived in the previous section are optimized using numerical methods. Several algorithms are used for this purpose, such as Stochastic Gradient Descent, which uses first derivative (gradient) information to find a minimum of a function, and Newton-Raphson algorithm, which uses Second derivative information to find a minimum of a function. Discussing the details of how these algorithms work and the mathematical derivation of gradients is beyond the scope of this work. (See (Jurafsky & Martin, 2014) for more details on the derivation of Stochastic Gradient Descent.)

2.5 Evaluation Metrics

2.5.1 Confusion Matrix

A confusion matrix is a method to visualize the results of a classification algorithm. For the binary classification, the algorithm can be used to predict whether a test sample is either 0, or 1. As a way to measure how well the algorithm performs, we can count four different metrics, where 1 defined as positive and 0 is defined as negative:

1. True positive (TP), the algorithm classifies 1, where the correct class is 1.
2. False positive (FP), the algorithm classifies 1, where the correct class is 0.
3. True negative (TN), the algorithm classifies 0, where the correct class is 0.
4. False negative (FN), the algorithm classifies 0, where the correct class is 1.

2.5.2 Precision, Recall, and F-Score

Precision and Recall are two popular measurements used to evaluate the performance of classification methods. Precision is defined as follows:

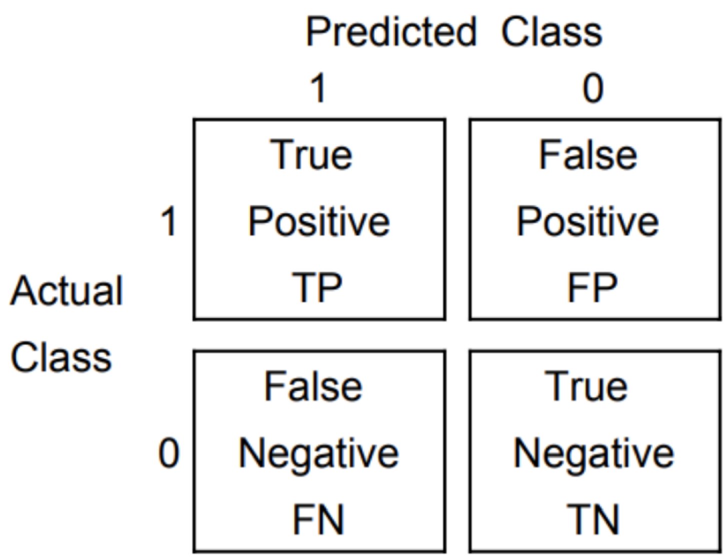
$$\text{Precision} \ = \frac{TruePostive}{TruePostive + FalsePositive}$$

; Recall is defined as:

$$\text{Recall} \ = \frac{TruePostive}{TruePostive + FalseNegative}$$

and F-score is the harmonic mean of both precision and recall:

FIGURE 2.2: Confusion Matrix Example



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

A classifier with high precision means it classifies almost no inputs as positive unless they are positive. A classifier with high recall, on the other hand, would mean that it misses almost no positive values.

2.6 Conclusion

In this chapter, we briefly introduced some fundamental topics of Machine Learning theory required for natural language processing supervised classification tasks. We discussed commonly used feature extraction methods, such as BOW, TF-IDF, and Distributional similarity. Additionally, we reviewed some techniques to reduce the dimensionality of feature space to reduce the complexity of the training and save computational resources. Finally, we discussed how logistic regression as a probabilistic algorithm can be used for binary or multinomial classification along with the evaluation metrics used to gauge its performance. In the next two chapters, we show the use of multinomial logistic regression classifiers on two novel tasks in natural language processing: grammatical difficulty categorization of English sentences and automatic reading comprehension. We also explore how logistic regression can be used in a semi-supervised manner to work on NLP tasks with limited data.

Chapter 3

The Use of Semi-supervised Learning in Classification of English Grammatical Structures

3.1 Introduction

The need to assess the difficulty of written texts is crucial in foreign language learning and teaching. Language instructors need a text difficulty metric that conforms to the guidelines of foreign language learning and teaching to find reading materials that match the proficiency level of their learners. Similarly, curricula designers and textbook publishers need such a metric to provide reliable and consistent learning resources and activities. Existing text evaluation metrics (Kincaid et al., 1975, Stenner et al., 2006, Landauer, Kireyev, & Panaccione, 2011, Graesser, McNamara, & Kulikowich, 2011) either do not conform to educational guidelines, or they are too broad to be useful for language learning and teaching purposes. Perhaps, the most popular text evaluation tools designed for educators is ETS's TextEvaluator (Napolitano, Sheehan, & Mundkowsky, 2015). It evaluates a written text based on a number of criteria that include coherence, cohesion, lexical density and syntactic difficulty. However, their syntactic difficulty feature still does not accommodate teachers' need. For example, if a teacher wants to provide their intermediate-level learners with a reading passage that features the language of cause and effect, or if-conditionals, a tool like ETS's TextEvaluator does not help much, considering the categorization features they use such as average sentence length, average number of modifiers per noun phrase, average number of dependent clauses per sentence (Sheehan et al., 2014). Therefore, in this chapter we design and implement a system that classifies a written text into three level of grammatical difficulty, are aligned with Common European Framework Reference (CEFR) (Education Committee. Modern Languages Division, 2001) language learning standards.

Common European Framework Reference (CEFR) is a standard for describing language achievement on a six-point scale from A1 for beginners up to C2 for proficient users' ability on the four primary skills (reading, writing, listening, speaking) and the two secondary skills: vocabulary and grammar. The latter two skills are considered the backbone for the four primary skills in terms of difficulty. For example, a frequently

TABLE 3.1: Example of CEFR Grammatical Rules

Level	guideword	Can-do statement	Example
A1	FORM: COMBINING TWO ADJECTIVES WITH 'AND'	Can use 'and' to join a limited range of common adjectives.	The teachers are very nice and friendly.
A2	FORM: COMBINING TWO ADJECTIVES WITH 'BUT'	Can use 'but' to join a limited range of common adjectives, after 'be'.	The weather was cloudy but fine.
B1	FORM/USE: 'THE' + NOUN + 'WHO/THAT', FOCUS	Can use defining relative clauses, 'the person who/that, the thing that, the (only) one who/that' as a focusing device.	The thing that was great is that the weather was really warm and it didn't rain.
B2	FORM/USE: 'LET'S NOT', SUGGESTION	Can use 'let's not' + base form of a main verb to make a suggestion.	Let's not lose track of each other again!
C1	FORM/USE: 'NOT ONLY … BUT ALSO' WITH INVERSION	Can use inverted auxiliary 'do' + the subject after 'not only', t o give focus.	Indeed, not only did they teach us useful knowledge, but they also organised practical exercises to ensure that we had assimilated all the information.
C2	FORM/USE: 'NEITHER'	Can use 'Neither' or 'Nor' + inverted auxiliary or 'be' + subject to add to a previous related negative clause, to focus on an additional negative factor.	Maybe he will eventually get over this terrible experience, but he's bound to be a lonelier boy than he was. Nor does Jack's future look any more promising.

uncommon lexical item like *sagacious* makes the sentence in which it appears more difficult to an English learner than a sentence with a more frequent lexical item like *wise*. Similarly, grammatical structures also play a vital role in the overall difficulty of a sentence. For instance, the word *should* conveys a different meaning in (1) *I should study for my final exams* than in (2) *This is your mission should you accept it*. The latter is considered more difficult to an English learner because the use of should as a conditional is less frequent than its use as a modal auxiliary expressing necessity according to CEFR (Education Committee. Modern Languages Division, 2001).

The English Grammar Profile (EGP) (O'Keeffe & Mark, 2017) is a finite list of (1222) grammar features compiled from the Cambridge Learner Corpus, which comprises over 250,000 scripts from Cambridge English Exams at all levels. The purpose of EGP is to establish which grammatical features characterizes learners' output at each level of CEFR. (O'Keeffe & Mark, 2017) The list contains structural features, their corresponding CEFR levels and example sentences 3.1

Our goal in this chapter is to train a multinomial logistic regression classifier on the example sentences derived from EPG to predict the CEFR difficulty level. Due to the limited number of examples sentences (around 3000), we combine each two levels (e.g. A1,A2) into one class (A), so we end up with three super-levels: A, B, and C , which correspond to elementary, intermediate, and advanced levels, respectively. We also improve the performance of the classification by applying a bootstrapping technique (Yarowsky, 1995) to fetch data from external sources to supplement the training data. Finally, we evaluate the performance of the classification before and after the bootstrapping against a random classifier.

3.2 Related Works

Sentence classification is the most common task in natural language processing. A considerable number of studies (Liu & Zhang, 2012; Pang, Lee, & Vaithyanathan, 2002; Stamatatos, 2009) have been conducted on lexicalized tasks like sentiment analysis, spam filtering, news categorization, etc. Rarely do we see work on classification sentences based on their syntactic structures. Thus, to the best of our knowledge, no work has ever tackled the task of classifying the grammatical difficulty of English sentences according to CEFR guidelines. Therefore, we will briefly survey techniques used to solve general sentence classification problems.

Proximity-based Algorithms such as Rocchio's algorithm (Rocchio, 1971) and K-nearest neighbor (Tam, Santoso, & Setiono, 2002) built vectors for each class using a training set of a document by measuring the similarity, such as Euclidean distance or cosine similarity, between the documents. Some studies (Bang, Yang, & Yang, 2006) incorporated dictionary-based methods to construct a conceptual similarity with KNN classifier, while (Chang & Poon, 2009) combined two KNN classifier with a Naive Bayes one using TF-IDF over phrases to categorize large collections of emails. While proximity-based methods perform well on document classification, yet using them on a large training set is not feasible, for computing similarities across documents is computationally and resource-wise expensive. Another disadvantage is that noise, and irrelevant data can severely degrade classification performance.

Another family of classifiers that work well in text category tasks is decision trees. Classification using this method is done via automatically creating "if-then" rules. Their use in tasks like spam filtering is widespread even with the advent of deep neural network methods (Wu, 2009). Another common powerful classifier is Naive Bayes that based on Bayes' rule, which performs surprisingly well for many real-world classification applications under some specific conditions (McCallum & Nigam, 1998; Rish, Hellerstein, & Thathachar, 2001). While Naive Bayes does not often outperform discriminative classifiers like Support-Vector Machine, it has the advantage of functioning well with small training data. (Kim et al., 2006) and (Isa et al., 2008) showed promising results by selecting Naive Bayes with SVM for text classification and clustering the documents. Using a Poisson Naive Bayes for text classification model also yielded excellent results (Isa et al., 2008).

The SVM classification method and logistic regression have shown outstanding results when used in text classification tasks (Yang & Liu, 1999) (Brücher, Knolmayer, & Mittermayer, 2002) perhaps because of their capacity of handling high-dimensional sparse data well. However, they can be relatively complex in training and require lengthy time and resource consumption.

3.3 Methods

3.3.1 Dataset: English Grammar Profile

The dataset used for this study, provided by English Grammar Profile, exhibits a typical grammar profile for each level. It consists of 3615 examples and 1215 rules 3.1, divided as follows: class A has 1194 supporting examples; class B has 1775 examples, and class C has 646 supporting examples. We merged the six levels into three supercategories in order to provide more data within each category.

The dataset provides some guidance of what characterizes each example. These rules are prepared to be used by ESL teachers, and they are not very computationally friendly.

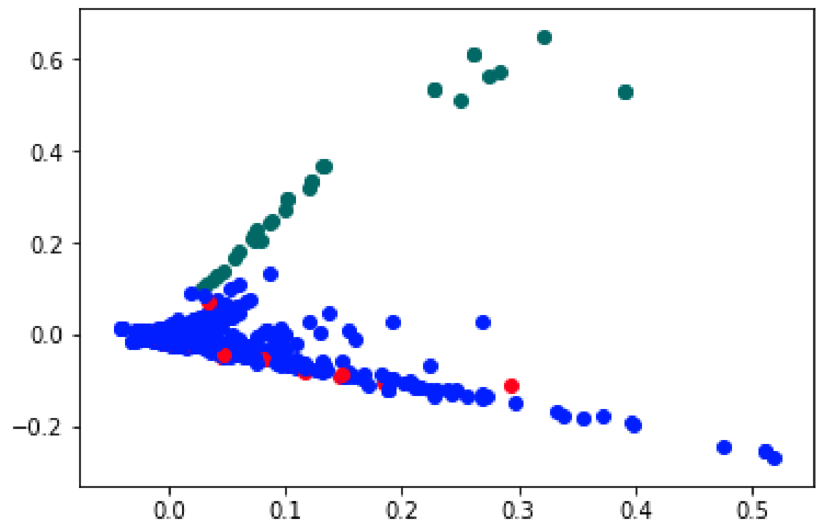


FIGURE 3.1: Two Dimensional Projection of classes A, B and C using K-means Clustering and Principal Component Analysis

For example, one rule of indicating the membership of a sentence into A2 class is having "a non-defining relative clause with ’which’ as the subject". A trained English teacher can easily spot a sentence with such a rule, yet detecting it using regular expression, for instance, is very difficult to achieve. On the other hand, the small number of examples in the dataset is not enough to build a reliable probabilistic classifier to learn to distinguish the various properties of the classes. Therefore, in order to improve the performance, we utilize a semi-supervised data augmentation technique similar to Yarawsky’s bootstrapping approach (Yarowsky, 1995).

3.3.2 Multinomial Logistic Regression

Suppose that $\mathbf{x}$ is the vectorized sentence in either of its two forms: BOW or TF-IDF, $y \in Y$ is the class label, and $\mathbf{w}_k$ and b_k are network parameters associated with class y. Then the probability of $\mathbf{x}$ belonging to the class y can be defined by the Softmax function:

$$p(y|\mathbf{x}) = \frac{1}{z(\mathbf{x})} \exp(\mathbf{w}_y^T \mathbf{x} + b_y)\,, \tag{3.1}$$

where $z(\mathbf{x}) = \sum_{j=1}^{K} \exp(\mathbf{w}_j^T \mathbf{x} + b_j)$.

Let the log-likelihood function be

$$L(\beta) = \sum_{i=1}^{N} \log p_{g_i}(x_i; \beta)$$

.

$$= \sum_{i=1}^{N} \left[\overline{\beta}_{g_i}^{T} x_i - \log \left(1 + \sum_{l=1}^{K-1} e^{\overline{\beta}_l^T x_i} \right) \right]$$

To apply the Newton-Raphson method, we need the second derivative of the log-likelihood function

$$\frac{\partial^2 L(\beta)}{\partial \beta_{kj} \partial \beta_{mn}} = - \sum_{i=1}^{N} x_{ij} x_{in} p_k \left(x_i; \beta \right) \left[l(k = m) - p_m \left(x_i; \beta \right) \right]$$

The formula for updating β_{new} for multiclass is:

$$\beta^{\text{ new}} = \beta^{\text{ old}} + \left(\tilde{\mathbf{X}}^T \mathbf{W} \tilde{\mathbf{X}} \right)^{-1} \tilde{\mathbf{X}}^T (\mathbf{y} - \mathbf{p})$$

where y is the concatenated indicator vector of dimension $N \times (K-1)$, p is the concatenated vector of fitted probabilities of dimension $N \times (K-1)$, $\tilde{X}$ is an $N(K-1) \times (p+1)(K-1)$ matrix; and Matrix W is an $N(K-1) \times N(K-1)$ square matrix. (Friedman, Hastie, & Tibshirani, 2001)

3.3.3 Feature Design

For this task, we are comparing two standard methods of feature generation in natural language processing and information retrieval: Bag-of-Words model and term-frequency inverse-document frequency (TF-IDF)

3.4 Experimental Results

In this section, we evaluate several variations of our method against a random model as our baseline model. In the following experiments, all methods are conducted using Python 3.5 and Scikit Learn Library tested on MacBook Pro laptop with Core i5 processor and 8 GB of RAM.

3.4.1 First Phase

In the first phase of our experiment, we train a logistic regression classifier, optimized by the Newton-Raphson method, on the English Grammar Profile dataset. We also introduce a feature design method to mask word categories with their part-of-speech tags in order to preserve grammatical information and avoid the lexical influence of the word. We refer to this method as *Masked* in Table 3.2. We use a combination of feature design methods such as:

- **BOW**: Apply unigram, bigram and trigram Bag-of-Words model with both word tokens, and masked tokens.

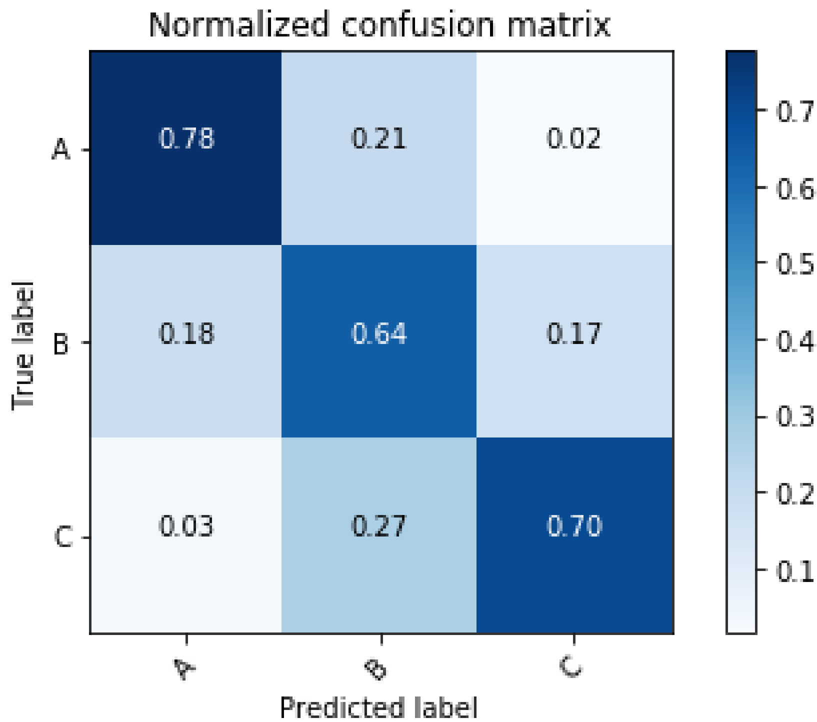


FIGURE 3.2: Confusion matrix illustrating the performance of LR-BOW-Word model with class weight to deal with data imbalance

- **TF-IDF**: Apply unigram, bigram and trigram TF-IDF model with both word tokens, and masked tokens.

From Table 3.2, we compare the performance of four variations of our method against the baseline, which is a random model. Surprisingly, we notice that our Bag-of-Words model with word tokens (BOW-LR-Words) outperform the one with the masked sequence (BOW-LR-Masked) in terms of precision and recall respectively. Besides, BOW-LR-Words also outperforms (TF-IDF-LR-Words) model in terms of recall only, while the latter has higher recall. From the confusion matrix (Fig. 3.2), we see that (BOW-LR-Words) model predicts most of the testing data correctly even though the number of training data is not equal among the classes, especially for class C. Therefore, we needed to assign specific weights to each class during the training in order to avoid this imbalance in the data.

TABLE 3.2: Comparison of Logistic Regression Performance Using Multiple Feature Extraction Methods

Model	Precision %	Recall %	F1 %
Random-Baseline	39 %	39 %	39 %
BOW-LR-Words	70 %	**69** %	**69** %
BOW-LR-Masked	66 %	62 %	63 %
TF-IDF-LR-Words	**75** %	66 %	60 %
TF-IDF-LR-Masked	66 %	64 %	61 %

3.4.2 Second Phase

The results shown in Table 3.2 does indicate that the linear model has learned the associations between sentences and their corresponding grammatical class reasonably well. However, both of our feature design techniques, namely BOW and TF-IDF have their disadvantages. The first assigns similarity based on occurrence, size and mutual words. For example, the BOW model learns that longer sentences tend to be the most difficult, while shorter ones are less difficult. Similarly, while TF-IDF treats the drawbacks of BOW model, it still ignores the usefulness of what-so-called *Stop Words* due to their commonality. In other words, it is very hard to trust these results with this amount of training data. Therefore, we propose a non-synthetic data augmentation method to provide more training examples, and thereby improve the overall performance of the model.

Using a text-only small version of the Brown Corpus (Francis, 1965), we use our BOW-LR-Words model to predict the grammatical difficulty labels of its sentences. Next, we collect the sentences predicted with high certainty (above 0.9 probability) to be fed into the original dataset. It is crucial to set a higher probability threshold, for the new data examples will serve as seed points for our prediction in the future; we do not want the classifier to *drift* from the original trajectory. (Curran, Murphy, & Scholz, 2007) This technique is reminiscent of Yarawsky's Bootstrapping (Yarowsky, 1995)(semi-supervised) algorithm, but with one difference is that unlike in Yarawsky's method, we apply this step only once.

Out of 38,400 sentences in the unlabeled corpus, only 1,428 sentences were detected with certainty above 90%. We also removed any sentence that is longer than 30 words in order to lessen the undesirable effect of BOW technique. We apply our best model from phase one to the augmented dataset of 5,043 examples under the same training conditions to get a precision of **0.80**, a recall of **0.79**, and an F1 score of **0.79**.

Comparing the confusion matrices before 3.2 and after 3.3 Bootstrapping, we notice that the performance of classification to class A does not improve much, unlike in classes B and C whose classification performance increases by F1 scores of 0.10 and 0.13 respectively.

3.5 Conclusion

In this chapter, we have presented a classification method based on simple multinomial logistic regression and Bag-of-Words model augmented with semi-supervised (Bootstrapping) method to classify English sentences based on their level difficulty level, A=Beginner, B=intermediate, and C=Advanced according to CEFR. We have also compared several common feature design techniques to encode the sentences along with a proposed masking BOW method that replaces lexical items with their grammatical category. Finally, the model achieves an overall F1 score of 0.69, and 0.79 after augmenting it with example sentences from an unlabeled corpus.

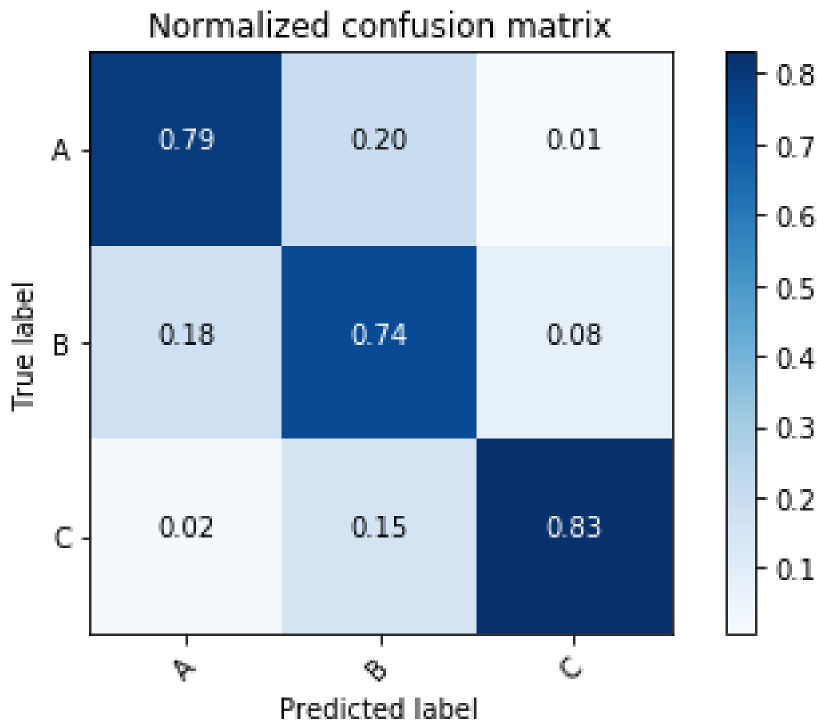


FIGURE 3.3: Confusion Matrix Illustrating the Performance of LR-BOW-Word Model after Augmentation

Chapter 4

Two-stage Classification Method for Automatic Reading Comprehension

4.1 Introduction

Teaching reading comprehension in an online learning environment is challenging to instructors who need to provide learners with reading material that is authentic, thematically diverse, and reader level appropriate. The system we developed in the previous chapter helps language instructors detect the grammatical difficulties of reading materials found online, but it still does not help eliminate the efforts they need to expend on developing practice exercises and quizzes. In order for second and foreign language learners to master a difficult skill like reading comprehension, they need a great deal of practice, which means more training exercises need to be prepared. To solve this problem, this chapter presents a system we designed and implemented that automatically answers comprehension questions given a reading text. While this system does not fully solve the problem raised, it can greatly aid in the automatic creation of reading comprehension practice activities in online learning environments.

Automatic Reading Comprehension (ARC) is the task of automatically answering comprehension questions based on information derived from short texts. Recently, it has received a special attention from the NLP community because of the second release of the Stanford Question Answering Dataset (SQuAD 2.0) (Rajpurkar, Jia, & Liang, 2018). The dataset consists of 150K questions posed by crowd workers on a set of Wikipedia articles. The answers to these questions lie within the reading passages, though some questions are unanswerable from within the reading passage. What sets SQUAD 2.0 from other data sets is that 50k of its questions are unanswerable, and this requires the learning models not only to find answers, but also to abstain from answering when necessary.

The strategy we will use for this task occurs in two stages 4.1. In the first stage, we break the reading passages into sentences, and train a multinomial logisitc regression classifier to find the sentence that most likely contains the answer. We also train the classifier to give a null-answer if an answer does not exist. Next, we divide the best candidate sentence from stage one into its constituent phrases, and train another multinomial logisitc regression classifier to find the constituent phrase that is most likely to be the answer to the question. This dual stage method has advantages over other competing models when it comes to speed and consumption of computational resources.

4.2 Related Work

Numerous studies (Bahdanau et al., 2017; Wang & Jiang, 2016; Xiong, Zhong, & Socher, 2016) have attempted to solve traditional Machine Comprehension tasks, where answers exist within the passage. The difficulty of SQUAD 2.0 task, though, lies in abstaining from answers when no answer span exists in the passage. Thus, in this section, we review models that address this constraint. It is prudent to note that all of the systems described in this section are complicated neural network architectures, so discussing their details is beyond the limitation of this report (See (Chen, 2018) for more details).

Seo et al. (2016) introduced bi-directional attention flow (BiDAF) that uses a recurrent neural network to encode contextual information in both question and passage along with an attention mechanism to align parts of the question to the sentence containing the answer and vice versa. The model offers context representation at multiple levels of granularity: character-level, word-level, and contextual embedding. What distinguishes this work from others is that it does not represent the context paragraph into a fixed-length vector. Instead, it dynamically computes a vector at each time step, combines it with the one from the previous layer, and allows flow through to the next layers. The model outputs confidence scores of start and end index of all potential answers. One problem with this model is that it is not designed to handle unanswerable questions. Levy et al. (2017) extended the work by assigning a probability to null-answers to account for the questions whose answers do not exist in the corresponding paragraph, achieving a 59.2% ExactMatch (EM) score and a 62.1% F1 score.

Hu et al. (2018) proposed a read-then-verify system that can abstain from answering when a question has no answer in a given passage. They introduce two auxiliary losses to help the neural reader network focus on answer extraction and no-answer detection respectively, and then utilize an answer verifier to validate the legitimacy of the predicted answer. One essential contribution is answer-verification. The model incorporates a multi-layer transformer decoder to recognize the textual entailment that supports the answer found in and passage. This model achieves an EM score of 71.6% and an F1 score of 74.23% on SQuAD 2.0.

Wang, Yan, & Wu (2018) proposed a new hierarchical attention network that mimics the human process of answering reading comprehension test questions. It gradually focuses attention on the part of the passage containing the answer to the question. The modal comprises of three layers. The **encoder layer** builds representations to both the question and the passage using a concatenation of word-embedding representation (Pennington, Socher, & Manning, 2014) and a pre-trained neural language modal (Peters et al., 2018). The **Attention layer**'s function is to capture the relationship between the question and the passage at multiple levels using self-attention mechanisms. Finally, the bilinear **Matching layer**, given a refined representation for both question and passage, detects the best answer span for the question. This method achieved state-of-the-art results as of September 2018. Their single model had a 79.2% EM and an 86.6% F1 score, while their ensemble model achieved 82.4% EM and 88.6% F1 Score, respectively.

Current models that have shown significant improvement on Machine Comprehension tasks, and similar tasks owe their success to a new neural architecture called transformer networks (Vaswani et al., 2017). It has become the *de facto* in recent sequential learning tasks, eschewing recurrence. This architecture creates global dependencies between input and output using only attention mechanisms. An even more successful model is Bidirectional Encoder Representations from Transformers BERT (Devlin et al., 2018). It is a task-independent pretrained language model that uses a deep transformer network to create rich and context-specific vector representation. Using a method called Masked Language Model, the goal is to randomly mask tokens from the input and predict the vocabulary id of the masked word based only on its context. BERT has shown significant improvement on several tasks, one of which is SQUAD.

While these models achieve impressive results, they are very difficult to implement and very resource-intensive. We argue that the simplifying linguistic assumptions we followed in our proposed strategy can greatly eliminate the need to such complexity of design, yielding competing results.

4.3 Stage One: Selecting the Best Candidate Sentence

At this level, the classification task is to predict the sentence in the paragraph that contains the correct answer or declare that the question is unanswerable. Therefore, we trained a multiclass logistic regression classifier that takes a question and a reading passage as inputs and outputs the sentence that most likely contains the answer. The classifier is trained with L2 regularization and optimized using the Newton- Raphson method. The best candidate sentence has the following three criteria. First, it shares more words with the question than other sentences. Second, it has a higher cosine similarity with the question than other sentences. Finally, it shares a syntactic similarity with the question. We are using these criteria as features for the classifier.

4.3.1 Feature Extraction

- **Cosine Similarity**: for every sentence in the paragraph as well as the question a word vector representation is created via InferSent (Conneau et al., 2017), which is a pre-trained sentence embeddings method that provides semantic representations for English sentences. InferSent is an encoder based on a bi-directional LSTM architecture with max pooling, trained on the Stanford Natural Language Inference (SNLI) dataset. Cosine distance score is calculated for each sentence-question pair.

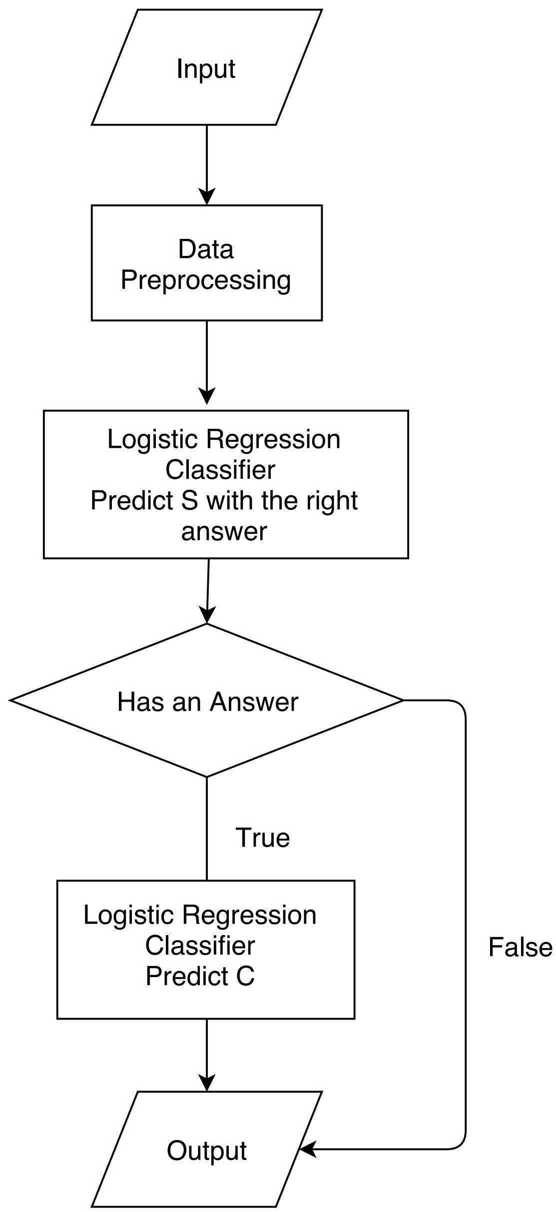


FIGURE 4.1: Flowchart illustrating the two-stage classification approach

TABLE 4.1: This table shows the results of running a multinomial regularized logistic regression. The class column represents the index of sentences within the paragraph, and -1 represents the unanswerable question case. Unpredicted classes are removed.

class	precision	recall	F1-score
-1	1	0.99	0.99
0	0.47	0.68	0.56
1	0.63	0.54	0.58
2	0.62	0.55	0.58
3	0.62	0.53	0.57
4	0.58	0.47	0.52
5	0.57	0.48	0.52
6	0.57	0.49	0.53
7	0.46	0.33	0.38
8	0.56	0.38	0.45
9	0.46	0.37	0.41
10	0.48	0.32	0.39
11	0.35	0.21	0.26
avg / total	0.72	0.71	0.71

- **Word Overlap**: calculates the Jaccard score between each sentence-question pair. Jaccard index is a method of computing the explicit similarity between two sets as follows:

$$J(Q,S) = \frac{|Q \cap S|}{|Q \cup S|}$$

  where Q and S are sets of words in question and sentence respectively.

- **POS Overlap**: computes the Jaccard score over the part-of-speech-tag representation of sentences. In other words, instead of the word tokens, it checks similarity over POS tokens. We use the default POS-tagger in the SpaCy library of Python programming language to obtain the POS representation for the sentences and questions alike.

Using the three features above, every question-sentence pair will have three scores and an additional binary feature indicating whether or not the question is answerable.

4.3.2 Training and Result

We get the results shown in 4.1. Numbers in the class column represents the index of the sentence in the paragraph containing the answer, and -1 indicates that the question has no answer in the paragraph. We also limit the number of sentences to 10. The results show that with simple features, we get an F1 score of 0.71.

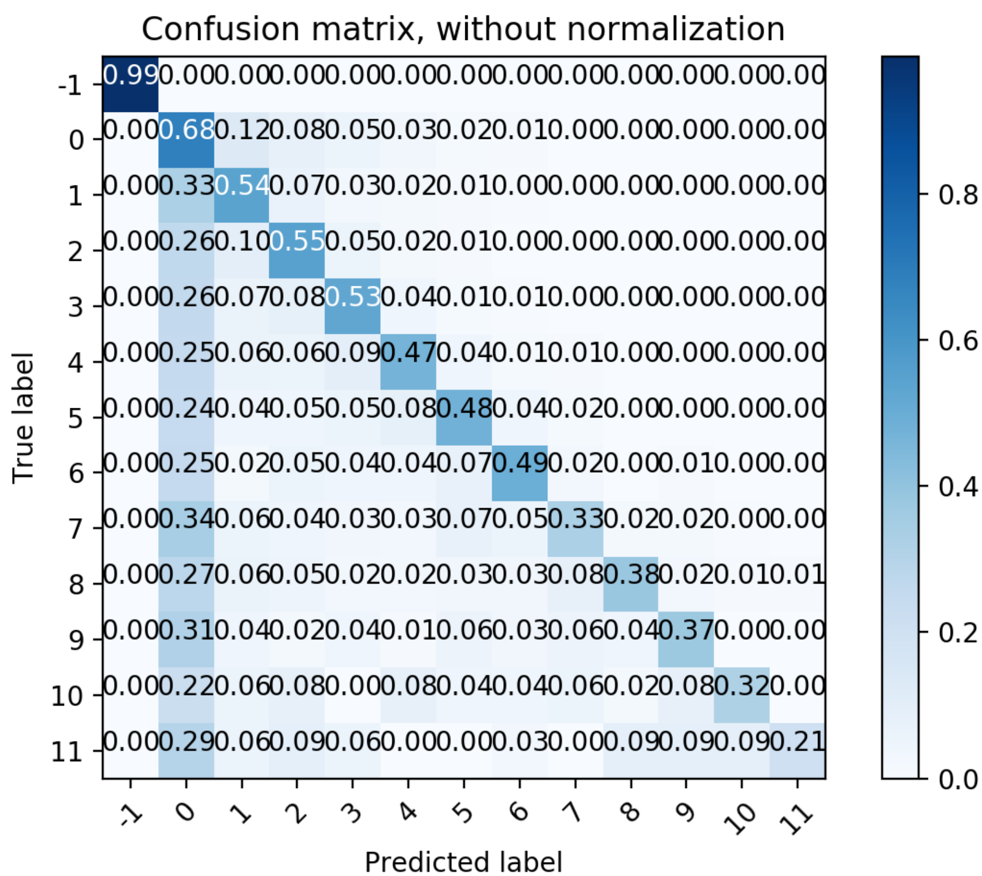


FIGURE 4.2: Confusion matrix shows which classes were predicted correctly in Stage One

4.4 Stage Two: Predicting the Answer Span

To select the most plausible answer span from the candidate sentence, we design a number of features, some of which are based on constituent analysis. A constituency parser is used to analyze a sentence into its constituent phrases. For example, the sentence "The quick brown fox jumps over the lazy dog." has the following syntactic (constituency) tree:

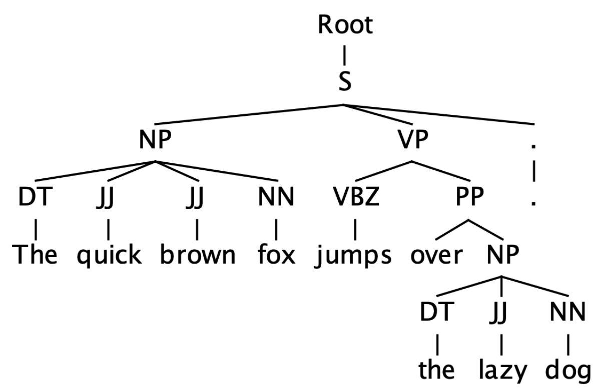


FIGURE 4.3: Example of Constituency Parse Tree

The intuition is an answer span is one of constituents of the best candidate sentence. We use a constituency parser (Kitaev & Klein, 2018) to obtain all constituents in a candidate sentence. These constituents will be the classes from which we pick the right span. For example, the sentence in 4.3 has the following classes: NP, the quick brown fox; VP, jumps over the lazy dog; VBZ, jumps; PP, over the lazy dog; and NP, the lazy dog. More generally, we design the following features for this stage:

- **Contextual Overlap**: Constituents sharing context with the original question are potential candidates to be the correct answers. So we measure the cosine similarity between each constituent and the question:

$$similarity = \frac{\sum_{i=1}^{n} C_{|w|} Q_i}{\sqrt{\sum_{i=1}^{n} C_{|w|}^2} \sqrt{\sum_{i=1}^{n} Q_i^2}}$$

  where w is the number of slide window around the candidate constituent. For our purposes features of size 2 and 3 are used.

- **Constituent Label**: Constituency parse tree label of the span is combined with wh-word. For example, for the question "Who designed the iPhone prototype?", and its answer "Steve Jobs designed the first prototype for the iPhone.", a constituent label feature is who-NP, who-V, who-VP, who-PP, etc.

- **Distributional Distance**: We measure the distributional cosine similarity between the sum of all words in the contextual window and the question using Glove (Pennington, Socher, & Manning, 2014).

- **Matching Word Frequencies**: Sum of the TF-IDF of the words that occur in both the question and the sentence containing the candidate answer.
- **Lengths**: Number of words to the left and the right of the span.

4.5 Experiment and Error Analysis

In this stage, we have done three different training attempts. In the first attempt 4.4, we used only two features "contextual overlap" and "constituent label". We also limited the number of classes (constituents) to 30. Using the same training settings and configurations from stage one, the logistic regression achieves a 0.25 F1 score. The confusion matrix shows a clear case of data imbalance. Classes at the beginning have more supporting points, and therefore fewer errors the others. However, our analysis of the errors indicates the wrong answers are partially correct most of the times. Class 2 was identified as class 1 34% of the time. For example, the answer to the question *At what age did Beyonce meet LaTavia Robertson?* is predicted as *At age eight* when the correct answer is *age eight*. Similarly, class 12 is predicted as class eleven 16%. The answer to the question *Who supervised the design and implementation of the iPod user interface?* is *Steve Jobs*, but it is predicted as *Of Steve Jobs*. Clearly, the predicted answers are not very far from the true ones. A harsher assumption such as keeping only Noun Phrase constituents could have resolved some of these issues, but we will leave this for future studies.

Other types of errors are due to literal word matching between the questions and phrases. For example, the answer to the question *How much was public expenditure on the island in 2001-2002?* is predicted to be *Public expenditure* when the true answer is *£10 million*. In this case, the phrase *public expenditure* appears in the question, which makes it a stronger candidate given the features designed. Another example is the answer to *What roles were women recruited for in the 1950s?* is *in medicine, communication*, but it is predicted as *logistics, and administration*. Given the features we have designed, it is very difficult for the classifier to recognize this subtle difference between the classes. In the second attempt 4.5, we have added the three remaining features: Distributional Distance, Matching Word Frequencies, and Length, while keeping the same parameters from the first attempt, resulting in an F1 score of 0.35. Looking at the confusion matrices of the first and second attempts, it is apparent that the new features do greatly improve the performance.

In our last experiment, we replaced logistic regression classifier in the second attempt with a 3-layer, feed-forward neural network of 128, 64 and 32 nodes respectively, optimized by Adam optimizer on a 100 epochs . The neural network achieves an F1 score of 0.42. Using a neural network does indeed increases the performance, but its use is not very practical given its configuration. Each node in this neural network is equivalent to one logistic regression classifier, so using a feed-forward neural network with a total of 224 nodes is equal in computational power to 224 logistic regression.

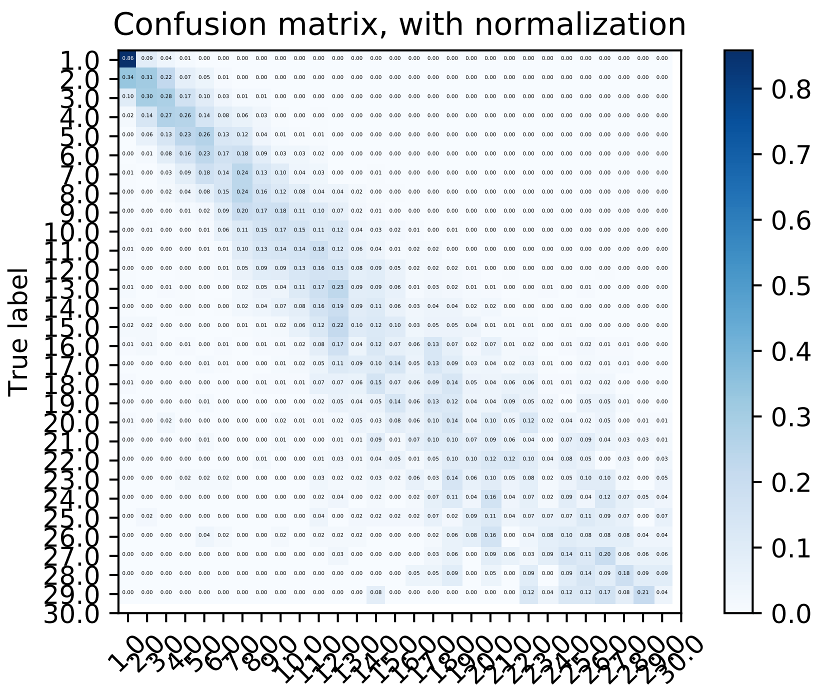


FIGURE 4.4: Confusion Matrix Illustrating the Prediction of Answer Span - First Attempt

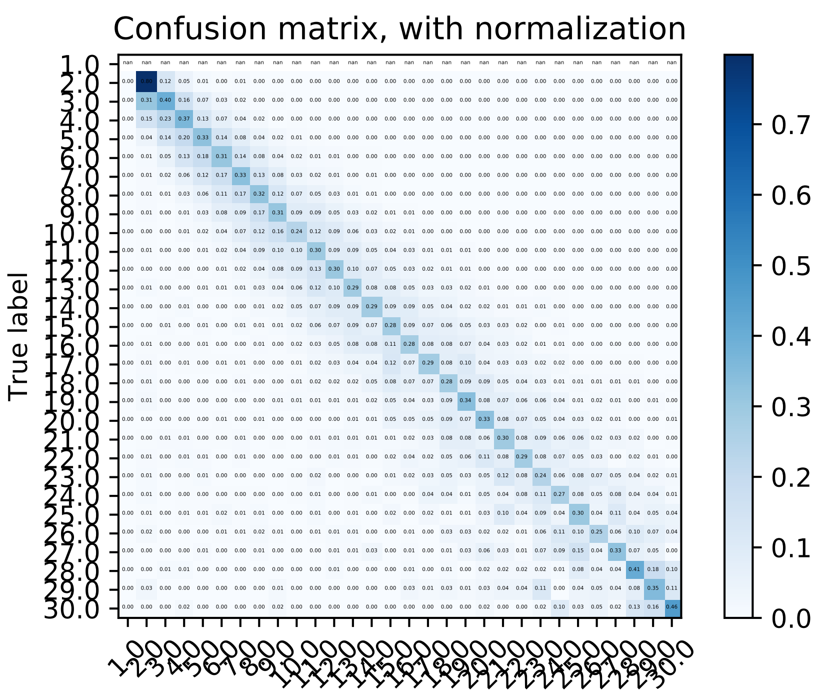


FIGURE 4.5: Confusion Matrix Illustrating the Prediction of Answer Span - Second Attempt

4.6 Future Studies and Recommendations

Looking at errors produced during the three attempts, we can improve the performance by injecting more linguistic information from Named Entity Recognizers, Semantic Role Labelers, and Dependency Parsers. A Named Entity Recognizer can help answer questions about locations, persons or entities (where, who and which questions). Information derived from a dependency parser and a semantic role labeler can, for example, help resolve cases when the answers are the subjects or the objects of the sentences. Another solution to investigate is to eliminate all constituents, but NP. This idea comes from the assumption that the majority of answers to Wh-questions in SQUAD 2.0 dataset are Noun Phrases. This assumption can help reduce the number of classes to be predicted, so there will be no need to limit the number of outputs to a certain number as we did.

4.7 Conclusion

We introduced a two-step classification method for automatic reading comprehension via the SQUAD 2.0 dataset. Our stage one classifier managed to find whether or not a question is answerable within a given passage and find the sentence containing the right answer with an F1 score of 0.71. Our stage 2 classifier manages to detect the exact span with an F1 score of 0.35 even though the predicted answer is not distant from the exact answer. In order to improve the performance of our approach, future studies should investigate the usefulness of features generated from Named Entity Recognition, Semantic Role Labeling and Dependency Parsing processes, which are expected to be potential solutions to the problems encountered in this research.

Chapter 5

Conclusion

Applications of Machine Learning in education are ubiquitous, from learning analytics to machine translation to summarization. One field of education that can greatly benefit from the applications of natural language processing is distance education. Teaching foreign languages by means of distance learning requires tremendous efforts from instructors and course developers to prepare online materials, learning activities, and resources. This report examines the applicability of a supervised classification technique called logistic regression in two educational scenarios: detection of grammatical difficulty and automatic reading comprehension. The simplicity and practicality of the proposed solutions to each of these problems make them uniquely useful in onsite and online teaching environments.

As we pointed out in Chapter 2, representing a written text is a crucial step in the machine learning pipeline, and to do so, we have explored some of the most common methods such as Bag-of-Words, TF-IDF, and word embeddings. We have also identified the advantages and disadvantages of each method in addition to techniques of dimensionality reduction such as clustering, latent semantic allocation, and stemming and lemmatization. Moreover, we have explained the workflow of classification as a supervised machine learning, starting from feature extraction through classification to evaluation of performance.

In Chapter 3, we examined the use of multinomial logistic regression classifier to classify a written text according to how difficult a foreign language learner of English sees it using the educational standards of the Common European Framework Reference (CEFR). We also showed how a logistic regression classifier could fit within a semi-supervised framework such as Bootstrapping to work on low resources data.

In Chapter 4, we investigated another use for logistic regression classification in method on a novel educational task, automatic reading comprehension. We have demonstrated that using logistic regression on two stages can compete with relatively advanced and complex neural network architecture commonly used for this purpose. The main catalyst for this success comes from using a set of linguistically rich features such as constituency parser and sentential embeddings. The performance of the model, the analysis of error made, as well as the simplicity and efficiency of the model make it feasible to be applied to onsite or virtual learning environments.

5.1 Summary of Contributions

The research offers three main contributions. First, to the best of our knowledge, we are the first to propose an automated solution to classify a written English text by following pedagogic standards (CEFR). Second, we have *cautiously* applied a bootstrapping technique to supplement the training data from unlabeled corpora, which led to 10% F1 increase in performance. Finally, we created a novel, two-stage system based on multinomial logistic regression that not only answers comprehension questions given reading passages but it also abstains from answering if a question is unanswerable within the reading passage.

5.2 Recommendations and Future Work

We have seen that each of the tasks we tackled in this paper can help solve a particular educational scenario a foreign language instructor may encounter in a face-to-face or online learning settings. However, there still exist some challenges that need the attention of NLP researchers. The solution proposed in Chapter 4 finds answers to comprehension questions, but it still requires the questions as input. This means that teachers and material developers must create the questions. Thus, future studies should investigate the process of automatically creating different types of comprehension questions given reading passages. Such a system, complemented by the system we proposed, can make an automatic tool of learning activity creation that is ready to be deployed in Learning Management Systems (LMS) or Courseware Platforms. As for improving the performance of the current automatic reading comprehension system, we recommend incorporating more linguistic features such as information derived from Named Entity Recognition, Dependency Parsing, and/or Semantic Role Labeling.

Due to the lack of training examples on each of the six classes in the grammatical detection task, we merged every two classes into one super-class to be the output. However, to provide more specificity to error detection, we recommend applying more iterations of cautious bootstrapping to obtain more training examples on each of the six classes (A1, A2, B1, B2, C1, and C2). More training examples can also reduce the state of data imbalance experienced in this task, and thereby eliminating the need for adjusting the class weights. The features we used are based on Bag-of-Words and TF-IDF, and we recommend using word embeddings or the sentence embeddings technique we have used for automatic reading comprehension task in Chapter 4. Another recommendation is to use easy-to-interpret classification algorithms like decision trees. This brings the advantage of providing "if-then" classification rules that humans can read and understand. Furthermore, as the task involves grammatical structures, we recommend designing features using constituency and dependency parsers. Finally, we encourage future studies to investigate how close will be the rules generated from the last two recommendations to the ones CEFR experts devised.

Appendix A

Auxiliary Functions for Codes in Ch3&4

LISTING A.1: Helper Functions for Tokenization, Projection and Plotting

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Dec  8 01:11:27 2018

@author: raghebal-ghezi
"""
import nltk
import matplotlib.pyplot as plt
import numpy as np
import itertools
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.manifold import TSNE

def posify(sent, flag='mixed'):
    #returns a pos-tagged sequences, or pos-word sequence.

    tokenized = nltk.tokenize.word_tokenize(sent)

    pos = nltk.pos_tag(tokenized)

    mixed_list = list()
    for t in pos:
        if t[1] in ['CC','IN','RB']:
            mixed_list.append(t[0])
        else:
            mixed_list.append(t[1])
    if flag == 'words': # returns the sentence intact
        return ' '.join(tokenized)
    if flag == 'POS': # returns POS tagged sequence instead
        return ' '.join([t[1] for t in pos])
    if flag == 'mixed': # a mixed of both
        return ' '.join(mixed_list)

def convert(label):
```

```
    dic = {"A":1,"B":2,"C":3}
#    dic = {"A1":1,"A2":2,"B1":3,"B2":4,"C1":5,"C2":6}
    return dic[str(label)]


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()


def project(texts_list, num_clusters):
    # project and plot. adapted from http://www.itkeyword.com/doc
        /7191219466270725017/how-do-i-visualize-data-points-of-tf-idf-vectors-
        for-kmeans-clustering
    num_seeds = 10
    max_iterations = 300
    labels_color_map = {
        0: '#001eff', 1: '#006865', 2: '#ff001a', 3: '#005073', 4: '#4d0404',
        5: '#ccc0ba', 6: '#4700f9', 7: '#f6f900', 8: '#00f91d', 9: '#da8c49'
    }
    pca_num_components = 2
    tsne_num_components = 2

    # texts_list = some array of strings for which TF-IDF is being computed

    # calculate tf-idf of texts
    tf_idf_vectorizer = TfidfVectorizer(analyzer="word", use_idf=True,
        smooth_idf=True, ngram_range=(2, 3))
```

```
    tf_idf_matrix = tf_idf_vectorizer.fit_transform(texts_list)

    # create k-means model with custom config
    clustering_model = KMeans(
        n_clusters=num_clusters,
        max_iter=max_iterations,
        precompute_distances="auto",
        n_jobs=-1
    )

    labels = clustering_model.fit_predict(tf_idf_matrix)
    # print labels

    X = tf_idf_matrix.todense()

    #
        ________________________________________________________________________________________________


    reduced_data = PCA(n_components=pca_num_components).fit_transform(X)
    # print reduced_data

    fig, ax = plt.subplots()
    for index, instance in enumerate(reduced_data):
        # print instance, index, labels[index]
        pca_comp_1, pca_comp_2 = reduced_data[index]
        color = labels_color_map[labels[index]]
        ax.scatter(pca_comp_1, pca_comp_2, c=color)
    plt.show()


#
#     # t-SNE plot
#     embeddings = TSNE(n_components=tsne_num_components)
#     Y = embeddings.fit_transform(X)
#     plt.scatter(Y[:, 0], Y[:, 1], cmap=plt.cm.Spectral)
#     plt.show()


def plot_pres_recall(classifier,X_train, X_test, y_train, y_test):

    y_score = classifier.fit(X_train, y_train).decision_function(X_test)

    # Compute Precision-Recall and plot curve
    precision = dict()
    recall = dict()
    average_precision = dict()
    for i in range(3):
        precision[i], recall[i], _ = precision_recall_curve(y_test[i],
                                                            y_score[i])
        average_precision[i] = average_precision_score(y_test[i], y_score[i])

    # Compute micro-average ROC curve and ROC area
    precision["micro"], recall["micro"], _ = precision_recall_curve(y_test.
       ravel(),
        y_score.ravel())
    average_precision["micro"] = average_precision_score(y_test, y_score,
                                                         average="micro")
```

```
# Plot Precision-Recall curve
plt.clf()
plt.plot(recall[0], precision[0], label='Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall example: AUC={0:0.2f}'.format(
    average_precision[0]))
plt.legend(loc="lower left")
plt.show()

# Plot Precision-Recall curve for each class
plt.clf()
plt.plot(recall["micro"], precision["micro"],
         label='micro-average Precision-recall curve (area = {0:0.2f})'
               ''.format(average_precision["micro"]))
for i in range(3):
    plt.plot(recall[i], precision[i],
             label='Precision-recall curve of class {0} (area = {1:0.2f})
                '
                  ''.format(i, average_precision[i]))

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Extension of Precision-Recall curve to multi-class')
plt.legend(loc="lower right")
plt.show()
```

Appendix B

Complete Code for Grammatical Detection System

LISTING B.1: Helper Functions for Tokenization, Projection and Plotting

```
import pandas as pd
import nltk
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, accuracy_score,
    classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_validate
import re
from aux_functions import *

# importing datset with merged categories
english_profile = pd.read_csv('./merged_labels_fixed.csv', \
    sep='\t', index_col=0)

def trainer(english_profile, vec='count', classifier='lr', \
    feature='words', binarize=True, class_w=True, res=True):
    '''
    Input: pandas dataframe of sentences and labels
    Param:
        vect: count = 'Bag-of-words' word vectorization model
              tfidf = tfidf
        classifier = 'dummy' for random classifier or 'lr' logisitic
            regression

        feature: words = tokenized word, mixed = sentence with
        some of their words replaced with POS tags (masked)
        ######## the 3 param. below are used to make the function usable in
            both phases.
        binarize: forces datatype of label to be integer.
        class_w: applies weight to certain gold_label classes to avoid data
            imbalance
        res: show plots
    '''
```

```
    #tokenizing the sentences in the dataframe
    english_profile['tokens'] = english_profile.Sentence.apply(posify, args=(
        feature,))
    #Applying Bag-of-Words Model of 1-gram, 2-gram, 3-gram
    if vec == 'count':
        vectorizer = CountVectorizer(lowercase=False, ngram_range=(1, 3))
    if vec == 'tfidf':
        vectorizer = TfidfVectorizer(lowercase=False, ngram_range=(1, 3))

    mat = vectorizer.fit_transform(english_profile['tokens'])
    # binarize gold label
    if binarize:
        english_profile['label'] = english_profile.Level.apply(convert)
    else:
        english_profile['label'] = english_profile['label'].apply(lambda x:
            int(x))
    # spliting data into training and test
    X_train, X_test, y_train, y_test = train_test_split(mat, english_profile[
        'label'],\
     test_size=0.33, random_state=42)
    # fitting logisitc regression model
    if classifier=='dummy':
        mul_lr = DummyClassifier()
    if classifier=='lr':
        if class_w:
            mul_lr = LogisticRegression(multi_class='multinomial', \
                                        solver='newton-cg',n_jobs
                                            =-1, random_state=42,\
                                        class_weight
                                            ={1:0.1,2:0.1,3:0.8})
        else:
            mul_lr = LogisticRegression(multi_class='multinomial', \
                                        solver='newton-cg',n_jobs
                                            =-1, random_state=42)

    mul_lr.fit(X_train, y_train)
    if res:
        print("Test Accuracy of % Train : ".format(classifier),
            accuracy_score(y_train, \
             mul_lr.predict(X_train)))
        print("Test Accuracy of % Train : ".format(classifier),
            accuracy_score(y_test, \
             mul_lr.predict(X_test)))
        print(classification_report(y_test, mul_lr.predict(X_test)))
        cnf_matrix = confusion_matrix(y_test, mul_lr.predict(X_test))
        np.set_printoptions(precision=2)
        # Plot normalized confusion matrix
        plt.figure()
        plot_confusion_matrix(cnf_matrix, classes=['A',"B","C"], normalize=
            True,
                      title='Normalized confusion matrix')

        plt.show()
    return vectorizer, mul_lr

# vectorize, train, test, plot confusion matrix
```

```
trainer(english_profile, vec='count', classifier='lr', feature='words',
    binarize=True)

# projecting using K-means and PCA. function definition in utils.py
project(english_profile.Sentence,3)

# loading the unlabelled corpus
with open("brown_corpus.txt") as file:
     txt_file = file.read()

# removing weird encoding characters
cleaned = re.sub('_.+?_|#.+|\n', '', txt_file)

# tokenize sentences
sents = nltk.tokenize.sent_tokenize(cleaned)

# tokenize words within each sentences. flag can take: words or mixed.
sents_posified = [posify(s, flag='words') for s in sents if len(s.split('␣'))
     < 30]

# res is set to false because I don't to show plots. I only need to the
    vectorizer
# to transform my data for prediction.
# Similarly, I need the classifier object to make inferences on the
    unlabelled data.
vectorizer, mul_lr = trainer(english_profile, vec='count', classifier='lr', \
                              feature='words', binarize=True, res=False)

#transforming my new data to match the dimensions of BoW matrix
brown_corpus = vectorizer.transform(sents_posified)

# create new dataframe to contain the sentences from the unlabelled sentences
    ,
# and their predicted label.
corpus_df = pd.DataFrame([sents_posified, mul_lr.predict(\
    brown_corpus).tolist()]).transpose()

# I also use the LR classifier to output the classification probability of
    each predicted label
# and use them as confidance score.
corpus_df['confidence'] = mul_lr.predict_proba(brown_corpus).tolist()

indivd_conf_score = list()
idx = corpus_df[1].tolist()
for i,j in enumerate(corpus_df.confidence):
    indivd_conf_score.append(j[idx[i]-1])

corpus_df['conf_score'] = indivd_conf_score

# filtering out low-prob scores
filtered_df = corpus_df.loc[corpus_df['conf_score'] > 0.9]

d1 = pd.DataFrame(english_profile[['Sentence','label']], columns=['Sentence',
    'label'])
d2 = pd.DataFrame(filtered_df[[0,1]])
d2.columns = ['Sentence','label']

combined_dataset = d1.append(d2)
```

```
# training a TOTALLY NEW logisitic regression classifier on the combined (
    augmented) dataset
# Please note that this is a new training process. The function, defined
    above, has its own vectorization \
# methods as well as training and testing
trainer(combined_dataset, vec='count', classifier='lr', \
    feature='words',binarize=False, class_w=False)
```

Appendix C

Complete Code for Automatic Reading Comprehension

LISTING C.1: Helper Functions for Tokenization, Projection and Plotting

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
from textblob import TextBlob
import torch
import spacy
from sklearn.metrics.pairwise import cosine_similarity as cosim
en_nlp = spacy.load('en_core_web_sm')

#loading the data
train = pd.read_json("dev-v2.0.json")

# pre-processing
contexts = []
questions = []
answers_text = []
answers_start = []
is_impossible = []
for i in range(train.shape[0]):
    topic = train.data[i]['paragraphs']
#    topic = train.iloc[i,0]['paragraphs']
    for sub_para in topic:
        for q_a in sub_para['qas']:
            questions.append(q_a['question'])
            try:
                answers_start.append(q_a['answers'][0]['answer_start'])
                answers_text.append(q_a['answers'][0]['text'])
            except IndexError:
                answers_start.append("NA")
                answers_text.append("NA")

            contexts.append(sub_para['context'])
            is_impossible.append(q_a['is_impossible'])

# Building a structured Dataframe
df = pd.DataFrame({"context":contexts, "question": questions,"is_impossible":
    is_impossible,
                   "answer_start": answers_start, "text": answers_text})
```

```
# encoding 'is_impossible' into binary
df.is_impossible = df.is_impossible.map(lambda x:0 if x == False else 1)

# creating a list of paragraphs
paras = list(df["context"].drop_duplicates().reset_index(drop= True))

# Sentence-tokenization
blob = TextBlob(" ".join(paras))
sentences = [item.raw for item in blob.sentences]

# loading sentence embeddings
infersent = torch.load('./InferSent/infersent.allnli.pickle', map_location=
    lambda storage, loc: storage)
infersent.set_glove_path("./InferSent/dataset/GloVe/glove.840B.300d.txt")
infersent.build_vocab(sentences, tokenize=True)

# Create a dictionary of each sentence in the paragraph and its embeddings
dict_embeddings = {}
for i in range(len(sentences)):
    dict_embeddings[sentences[i]] = infersent.encode([sentences[i]], tokenize
        =True)


# Create a dictionary of each question and its embeddings
questions = list(df["question"])
for i in range(len(questions)):
    dict_embeddings[questions[i]] = infersent.encode([questions[i]], tokenize
        =True)

def get_target(x):
     #input: pandas dataframe containing tokenized sentences and correct
         answers
        #returns: the index of sentence containing right answer or -1 if the
            question is unanswerable
    idx = -1
    for i in range(len(x["sentences"])):
        if x["text"] in x["sentences"][i]: idx = i
    return idx


def cosine_sim(x):
    li = []
    for item in x["sent_emb"][0]:
        li.append(cosim(item,x["quest_emb"][0][0]))
    return li
def fetch_dep(string):
    # returns a word, dependency tag pair for each sentence
    lst = []
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(string)
    for sent in doc.sents:
        for token in sent:
            lst.append((token.text,token.dep_))
    return lst

def fetch_ner(string):
    #returns a word, ner tag pair for each sentence.
```

```
    lst = []
    nlp = spacy.load('en_core_web_sm')
    doc = nlp(string)
    for ent in doc.ents:
        lst.append((ent.text, ent.label_))
    return lst

def jaccard(q,s):
    # returns jaccard score between two lists (texts or tags)
    if type(q) == list:
        a = set(q)
    else:
        a = set(str(q).split(" "))
    if type(s) == list:
        b = set(s)
    else:
        b = set(str(s).split(" "))

    return len(a.intersection(b)) / len(a.union(b))

def jac_index(x):
    li = []
    for item in x["sentences"]:
        li.append(jaccard(item,x["question"]))
    return li

def process_data(train):
    '''
        input: pandas dataframe
        returns: pandas dataframe with more columns (information)
        '''
    print("step 1") #append word_tokenized sequences to the dataframe
    train['sentences'] = train['context'].apply(lambda x: [item.raw for item
        in TextBlob(x).sentences])

    print("step 2") # gold labels
    train["target"] = train.apply(get_target, axis = 1)

    print("step 3") # append sentence embeddings for each sentence in the
        paragraph
    train['sent_emb'] = train['sentences'].apply(lambda x: [dict_embeddings[
        item][0] if item in\
                                                         dict_embeddings
                                                             else np.zeros
                                                             (4096) for item
                                                              in x])
    print("step 4") # append sentence embeddings for the question
    train['quest_emb'] = train['question'].apply(lambda x: dict_embeddings[x]
         if x in dict_embeddings else np.zeros(4096) )
    print("step 5") # append a list of jaccard indices of each question-
        sentence pair
    train["word_overlap"] = train.apply(jac_index, axis = 1)
    print("step 6") # append a list of cosine of each question-sentence pair
    train.quest_emb.apply(lambda x: x.reshape((4096,)))
    train["cosine_sim"] = train.apply(cosine_sim, axis = 1)

    print("step 7") # append the index of sentence with highest cosine sim
    train["pred_idx_cos"] = train["cosine_sim"].apply(lambda x: np.argmax(x))
```

```
    print("step␣8") # append the index of sentence with highest jaccard
    train["pred_idx_wrdovlp"] = train["word_overlap"].apply(lambda x: np.
        argmax(x))

    return train


new_df = process_data(df)
```

LISTING C.2: Helper Functions for Tokenization, Projection and Plotting

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  2 14:56:45 2018

@author: raghebal-ghezi
"""
import pandas as pd
import re
import numpy as np
from sklearn import linear_model
from sklearn import metrics
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
def clean(x):
    return [float(i) for i in re.findall("\s\d.\d*",str(x))]


#load modified data
dataframe = pd.read_csv("./with_pos_overlap_score.csv",dtype={"word_overlap":
    list})

#extract relevent columns for features
train2 = dataframe[["is_impossible","cosine_sim","word_overlap","
    pos_tag_ovrlap","target"]]

#De-stringifying columns
train2["is_impossible"]= train2["is_impossible"].apply(lambda x:1 if x==True
    else 0)
train2["cosine_sim"]= train2["cosine_sim"].map(clean)
train2["word_overlap"]= train2["word_overlap"].apply(clean)
train2["pos_tag_ovrlap"]= train2["pos_tag_ovrlap"].apply(clean)
# Using a small portion of the data for easy computation
small_partion = train2.iloc[:]

#generating features for classification
t = pd.DataFrame()
for i,j in enumerate(small_partion.cosine_sim):
    for k,l in enumerate(j):
        if k < 11:
            t.loc[i, "column_cos_"+"%s"%k] = l

for i,j in enumerate(small_partion.word_overlap):
    for k,l in enumerate(j):
```

```
        if k < 11:
            t.loc[i, "column_wordOverlap_"+"%s"%k] = 1

for i,j in enumerate(small_partion.pos_tag_ovrlap):
    for k,l in enumerate(j):
        if k < 11:
            t.loc[i, "column_POSOverlap_"+"%s"%k] = 1
t["is_impossible"] = small_partion["is_impossible"]
t["target"] = small_partion["target"]

# clean null values
subset1 = t.iloc[:,:10].fillna(1)
subset2=t.iloc[:,11:].fillna(0)
train_final = pd.concat([subset1, subset2],axis=1, join_axes=[subset1.index])

#saving only feature vectors to disk
# train_final.to_csv("feature_set.csv")

#Classifiication

scaler = MinMaxScaler()
X = scaler.fit_transform(train_final.iloc[:,:-1])
train_x, test_x, train_y, test_y = train_test_split(X,train_final.iloc[:,-1],
     train_size=0.8, random_state = 5)
mul_lr = linear_model.LogisticRegression(multi_class='multinomial', solver='
    newton-cg')
mul_lr.fit(train_x, train_y)

print("Multinomial Logistic regression Train Accuracy : ", metrics.
    accuracy_score(train_y, mul_lr.predict(train_x)))
print("Multinomial Logistic regression Test Accuracy : ", metrics.
    accuracy_score(test_y, mul_lr.predict(test_x)))
print(classification_report(test_y, mul_lr.predict(test_x), labels=np.unique(
    mul_lr.predict(test_x))))
print(confusion_matrix(test_y, mul_lr.predict(test_x)))
```

LISTING C.3: Helper Functions for Tokenization, Projection and Plotting

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 22 12:33:09 2018

@author: raghebal-ghezi
"""
import itertools
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt


#pickled Pandas DataFrame that has the following cols:
# ['candid', 'const_tags', 'constituents', 'cosine_2',
```

```
# 'cosine_3', 'left_span', 'question', 'right_span',
# 'span_length', 'text', 'wh+tag', 'target', 'tfidf_sum',
# 'tfidf_sum_2', 'tfidf_sum_3', 'glove_cos_2']
df = pd.read_pickle('serialized_df.pkl')


df_feature = pd.DataFrame() #to store the features
#try:
print("Generating features ...... (1 of 5)")
for i,j in enumerate(df.cosine_2): # prepare features for Contextual Overlap
    with window size 2
    for k,l in enumerate(j):
        if k < 31:
            df_feature.loc[i, "column_cos2_"+"%s"%k] = l
print("Generating features ...... (2 of 5)")
for i,j in enumerate(df.cosine_3): # prepare features for Contextual Overlap
    with window size 3
    for k,l in enumerate(j):
        if k < 31:
            df_feature.loc[i, "column_cos3_"+"%s"%k] = l
print("Generating features ...... (3 of 5)")
for i,j in enumerate(df.tfidf_sum_2):# sum of tfidf values for Contextual
    Overlap with window size 2
    for k,l in enumerate(j):
        if k < 31:
            df_feature.loc[i, "column_tfidf2_"+"%s"%k] = l
print("Generating features ...... (4 of 5)")
for i,j in enumerate(df.tfidf_sum_3): # sum of tfidf values for Contextual
    Overlap with window size 3
    for k,l in enumerate(j):
        if k < 31:
            df_feature.loc[i, "column_tfidf3_"+"%s"%k] = l
print("Generating features ...... (5 of 5)")
for i,j in enumerate(df.glove_cos_2): # distributional cos sim
    for k,l in enumerate(j):
        if k < 31:
            df_feature.loc[i, "column_gloveCos_"+"%s"%k] = l

# Appending the remaining features from original dataframe
df_feature['wh+tag'] = df['wh+tag']
df_feature['left_span'] = df['left_span']
df_feature['right_span'] = df['right_span']
df_feature['span_length'] = df['span_length']
df_feature['target'] = df.target

# restricting the number of target constituents to 30
train_final = df_feature[df_feature['target'] < 31]

# filling in missing values with zeros
train_final = train_final.fillna(0)

# enforcing the datatype to 'wh+tag'
train_final['wh+tag'] = train_final['wh+tag'].map(lambda x:str(x).lower())
#fixing the indices of dataframe
train_final = train_final.reset_index(drop=True)

# encoding the 'wh+tag' to numerical values
le = preprocessing.LabelEncoder()
```

```
train_final['wh+tag'] = le.fit_transform(train_final['wh+tag'])

# assigning X and y
X = train_final.drop('target',axis=1)
y = train_final['target']

# splitting and Shuffling
train_x, test_x, train_y, test_y = train_test_split(X,y, train_size=0.7,
    random_state = 5,shuffle=True)
# Using LR with Newton methods optimizer
mul_lr = linear_model.LogisticRegression(random_state=0, solver='newton-cg',
    multi_class='multinomial',n_jobs=-1)
mul_lr.fit(train_x, train_y)

# Printing Classification Report
print(classification_report(test_y, mul_lr.predict(test_x), labels=np.unique(
    mul_lr.predict(test_x))))


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black", fontsize=2)

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig('fig_1.pdf')


# Compute confusion matrix
cnf_matrix = confusion_matrix(test_y, mul_lr.predict(test_x), labels=[i for i
     in range(0,30)])
np.set_printoptions(precision=2)
```

```
# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=np.unique(mul_lr.predict(test_x)),
                      title='Confusion␣matrix,␣with␣normalization', normalize
                          =True)
```

Bibliography

Bahdanau, Dzmitry et al. (2017). "Learning to Compute Word Embeddings On the Fly". In: *CoRR* abs/1706.00286. arXiv: 1706.00286. URL: http://arxiv.org/abs/1706.00286.

Baker, L Douglas & Andrew Kachites McCallum (1998). "Distributional clustering of words for text classification". In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 96–103.

Bang, Sun Lee, Jae Dong Yang, & Hyung Jeong Yang (Mar. 2006). "Hierarchical document categorization with k-NN and concept-based thesauri". In: *Information Processing & Management* 42.2, pp. 387–406. ISSN: 0306-4573. DOI: 10.1016/j.ipm.2005.04.003. URL: http://www.sciencedirect.com/science/article/pii/S0306457305000579 (visited on 12/09/2018).

Bellman, Richard E (2015). *Adaptive control processes: a guided tour*. Vol. 2045. Princeton university press.

Bird, Steven, Ewan Klein, & Edward Loper (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc."

Brücher, Heide, Gerhard Knolmayer, & Marc-André Mittermayer (2002). "Document classification methods for organizing explicit knowledge". In:

Chang, Matthew & Chung Keung Poon (June 2009). "Using phrases as features in email classification". In: *Journal of Systems and Software* 82.6, pp. 1036–1045. ISSN: 0164-1212. DOI: 10.1016/j.jss.2009.01.013. URL: http://www.sciencedirect.com/science/article/pii/S0164121209000089 (visited on 12/09/2018).

Chen, Danqi (2018). "Neural Reading Comprehension and Beyond". PhD thesis. Stanford University.

Conneau, Alexis et al. (2017). "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *CoRR* abs/1705.02364. arXiv: 1705.02364. URL: http://arxiv.org/abs/1705.02364.

Curran, James R, Tara Murphy, & Bernhard Scholz (2007). "Minimising semantic drift with mutual exclusion bootstrapping". In: *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*. Vol. 6. Citeseer, pp. 172–180.

Deerwester, Scott et al. (1990). "Indexing by latent semantic analysis". In: *Journal of the American society for information science* 41.6, pp. 391–407.

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Education Committee. Modern Languages Division, Council of Europe. Council for Cultural Co-operation. (2001). *Common European Framework of Reference for Languages: learning, teaching, assessment*. Cambridge University Press.

Francis, W Nelson (1965). "A standard corpus of edited present-day American English". In: *College English* 26.4, pp. 267–273.

Friedman, Jerome, Trevor Hastie, & Robert Tibshirani (2001). *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York.

Fuhr, Norbert & Chris Buckley (1991). "A probabilistic learning approach for document indexing". In: *ACM Transactions on Information Systems (TOIS)* 9.3, pp. 223–248.

Graesser, Arthur C, Danielle S McNamara, & Jonna M Kulikowich (2011). "Coh-Metrix: Providing multilevel analyses of text characteristics". In: *Educational researcher* 40.5, pp. 223–234.

Hu, Minghao et al. (2018). "Read + Verify: Machine Reading Comprehension with Unanswerable Questions". In: *CoRR* abs/1808.05759. arXiv: 1808.05759. URL: http://arxiv.org/abs/1808.05759.

Isa, D. et al. (Sept. 2008). "Text Document Preprocessing with the Bayes Formula for Classification Using the Support Vector Machine". In: *IEEE Transactions on Knowledge and Data Engineering* 20.9, pp. 1264–1272. ISSN: 1041-4347. DOI: 10.1109/TKDE.2008.76.

Jurafsky, Dan & James H Martin (2014). *Speech and language processing*. Vol. 3. Pearson London.

Kim, Sang-Bum et al. (Nov. 2006). "Some Effective Techniques for Naive Bayes Text Classification". In: *IEEE Transactions on Knowledge and Data Engineering* 18.11, pp. 1457–1466. ISSN: 1041-4347. DOI: 10.1109/TKDE.2006.180.

Kincaid, JP et al. (1975). "Flesch-Kincaid Grade Level". In: *Memphis: United States Navy*.

Kitaev, Nikita & Dan Klein (2018). "Constituency Parsing with a Self-Attentive Encoder". In: *CoRR* abs/1805.01052. arXiv: 1805.01052. URL: http://arxiv.org/abs/1805.01052.

Landauer, Thomas K, Kirill Kireyev, & Charles Panaccione (2011). "Word maturity: A new metric for word knowledge". In: *Scientific Studies of Reading* 15.1, pp. 92–108.

Levy, Omer et al. (2017). "Zero-Shot Relation Extraction via Reading Comprehension". In: *CoRR* abs/1706.04115. arXiv: 1706.04115. URL: http://arxiv.org/abs/1706.04115.

Lewis, David Dolan (1992). "Representation and learning in information retrieval". PhD thesis. University of Massachusetts at Amherst.

Liu, Bing & Lei Zhang (2012). "A survey of opinion mining and sentiment analysis". In: *Mining text data*. Springer, pp. 415–463.

Manning, Christopher D & Hinrich Schütze (1999). *Foundations of statistical natural language processing*. MIT press.

McCallum, Andrew, Kamal Nigam, et al. (1998). "A comparison of event models for naive bayes text classification". In: *AAAI-98 workshop on learning for text categorization*. Vol. 752. 1. Citeseer, pp. 41–48.

Mikolov, Tomas et al. (2013). "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*, pp. 3111–3119.

Napolitano, Diane, Kathleen Sheehan, & Robert Mundkowsky (2015). "Online readability and text complexity analysis with TextEvaluator". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pp. 96–100.

O'Keeffe, Anne & Geraldine Mark (2017). "The English Grammar Profile of learner competence". In: *International Journal of Corpus Linguistics* 22.4, pp. 457–489.

Pang, Bo, Lillian Lee, & Shivakumar Vaithyanathan (2002). "Thumbs up?: sentiment classification using machine learning techniques". In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, pp. 79–86.

Pennington, Jeffrey, Richard Socher, & Christopher Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Peters, Matthew E. et al. (2018). "Deep contextualized word representations". In: *Proc. of NAACL*.

Porter, Martin F (1980). "An algorithm for suffix stripping". In: *Program* 14.3, pp. 130–137.

Rajpurkar, Pranav, Robin Jia, & Percy Liang (2018). "Know What You Don't Know: Unanswerable Questions for SQuAD". In: *arXiv preprint arXiv:1806.03822*.

Rish, Irina, Joseph Hellerstein, & Jayram Thathachar (2001). *An analysis of data characteristics that affect naive Bayes performance*. Tech. rep.

Rocchio, Joseph John (1971). "Relevance feedback in information retrieval". In: *The SMART retrieval system: experiments in automatic document processing*, pp. 313–323.

Sable, Carl & Kenneth W Church (2001). "Using bins to empirically estimate term weights for text categorization". In: *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*.

Seo, Min Joon et al. (2016). "Bidirectional Attention Flow for Machine Comprehension". In: *CoRR* abs/1611.01603. arXiv: 1611.01603. URL: http://arxiv.org/abs/1611.01603.

Shannon, Claude Elwood (1948). "A mathematical theory of communication". In: *Bell system technical journal* 27.3, pp. 379–423.

Sheehan, Kathleen M. et al. (2014). "The TextEvaluator Tool: Helping Teachers and Test Developers Select Texts for Use in Instruction and Assessment". In: *The Elementary School Journal* 115.2, pp. 184–209. DOI: 10.1086/678294. eprint: https://doi.org/10.1086/678294. URL: https://doi.org/10.1086/678294.

Singhal, Amit, Mandar Mitra, & Chris Buckley (1997). "Learning routing queries in a query zone". In: *ACM SIGIR Forum*. Vol. 31. SI. ACM, pp. 25–32.

Slonim, Noam & Naftali Tishby (2001). "The power of word clusters for text classification". In: *23rd European Colloquium on Information Retrieval Research*. Vol. 1, p. 200.

Stamatatos, Efstathios (2009). "A survey of modern authorship attribution methods". In: *Journal of the American Society for information Science and Technology* 60.3, pp. 538–556.

Stenner, A Jackson et al. (2006). "How accurate are Lexile text measures?" In: *Journal of Applied Measurement* 7.3, p. 307.

Tam, V., A. Santoso, & R. Setiono (Aug. 2002). "A comparative study of centroid-based, neighborhood-based and statistical approaches for effective document categorization". In: *Object recognition supported by user interaction for service robots*. Vol. 4, 235–238 vol.4. DOI: 10.1109/ICPR.2002.1047440.

Tzeras, Kostas & Stephan Hartmann (1993). "Automatic indexing based on Bayesian inference networks". In: *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, pp. 22–35.

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *CoRR* abs/1706.03762. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

Wang, Shuohang & Jing Jiang (2016). "Machine Comprehension Using Match-LSTM and Answer Pointer". In: *CoRR* abs/1608.07905. arXiv: 1608.07905. URL: http://arxiv.org/abs/1608.07905.

Wang, Wei, Ming Yan, & Chen Wu (2018). "Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering". In: *CoRR* abs/1811.11934. arXiv: 1811.11934. URL: http://arxiv.org/abs/1811.11934.

Wu, Chih-Hung (Apr. 2009). "Behavior-based spam detection using a hybrid method of rule-based techniques and neural networks". In: *Expert Systems with Applications* 36.3, Part 1, pp. 4321–4330. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2008.03.002. URL: http://www.sciencedirect.com/science/article/pii/S0957417408001772 (visited on 12/09/2018).

Xiong, Caiming, Victor Zhong, & Richard Socher (2016). "Dynamic Coattention Networks For Question Answering". In: *CoRR* abs/1611.01604. arXiv: 1611.01604. URL: http://arxiv.org/abs/1611.01604.

Yang, Yiming & Xin Liu (1999). "A Re-Examination of Text Categorization Methods". In: *SIGIR*.

Yarowsky, David (June 1995). "Unsupervised word sense disambiguation rivaling supervised methods". In: *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, Massachusetts, USA: Association for Computational Linguistics, pp. 189–196. DOI: 10.3115/981658.981684. URL: http://www.aclweb.org/anthology/P95-1026 (visited on 12/09/2018).